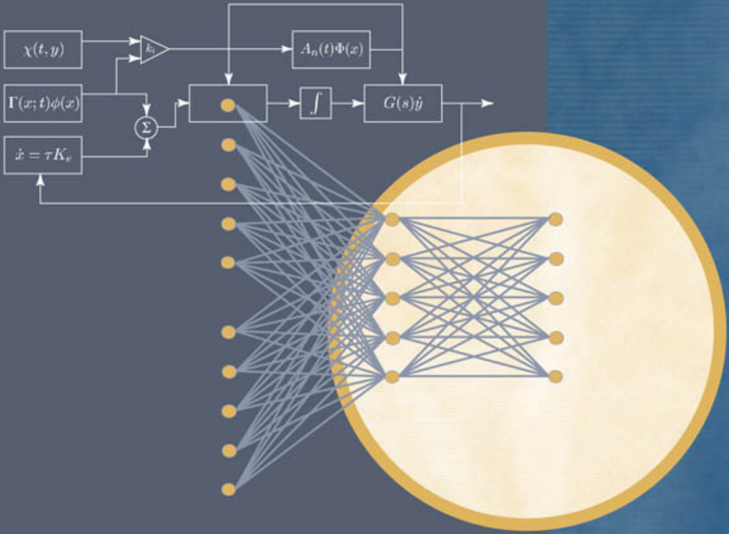
Neural Engineering

Computation, Representation, and Dynamics in Neurobiological Systems

Chris Eliasmith and Charles H. Anderson



Neural Engineering

Neural Engineering
Computation, Representation, and Dynamics in Neurobiological Systems

Chris Eliasmith and Charles H. Anderson

A Bradford Book
The MIT Press
Cambridge, Massachusetts
London, England

This book was typeset in Times by the authors using LyX and LaTeX and was printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Eliasmith, Chris.
Neural engineering : computation, representation, and dynamics in neurobiological systems / Chris Eliasmith and Charles H. Anderson.
p. cm. – (Computational neuroscience)
"A Bradford book."
Includes bibliographical references and index.
ISBN 0-262-05071-4 (hc.)
1. Neural networks (Neurobiology) 2. Neural networks (Computer science) 3. Computational neuroscience. I. Anderson, Charles H. II. Title. III. Series.

QP363.3 .E454 2002
573.8'5–dc21

2002070166

10 9 8 7 6 5 4 3 2 1

To Jen, Alana, Alex, and Charlie

and

To David Van Essen

This page intentionally left blank

Contents

Preface

This book is a rudimentary attempt to generate a coherent understanding of neurobiological systems from the perspective of what has become known as 'systems neuroscience.' What is described in these pages is the result of a five year collaboration aimed at trying to characterize the myriad, fascinating neurobiological systems that we encounter every day. Not surprisingly, this final (for now) product is vastly different from its ancestors. But, like them, it is first and foremost a *synthesis* of current ideas in computational, or theoretical, neuroscience. We have adopted and extended ideas about neural coding, neural computation, physiology, communications theory, control theory, representation, dynamics, and probability theory. The value of presenting a synthesis of this material, rather than presenting it as a series of loosely connected ideas, is to provide, we hope, both theoretical and practical insight into the functioning of neural systems not otherwise available. For example, we are not only interested in knowing what a particular neuron's tuning curve looks like, or how much information that neuron *could* transmit, we want to understand how to combine this evidence to learn about the possible function of the system, and the likely physiological characteristics of its component parts. Attempting to construct a general framework for understanding neurobiological systems provides a novel way to address these kinds of issues.

Our intended audience is quite broad, ranging from physiologists to physicists, and advanced undergraduates to seasoned researchers. Nevertheless, we take there to be three main audiences for this book. The first consists of neuroscientists who are interested in learning more about how to best characterize the systems they explore experimentally. Often the techniques used by neuroscientists are chosen for their immediate convenience—e.g., the typical 'selectivity index' calculated from some ratio of neuron responses—but the limitations inherent in these choices for characterizing the systemic coding properties of populations of neurons are often serious, though not immediately obvious (Mechler and Ringach 2002). By adopting the three principles of neural engineering that we present, these sorts of measures can be replaced by others with a more solid theoretical foundation. More practically speaking, we also want to encourage the recent trend for experimentalists to take seriously the insights gained from using detailed computational models. Unfortunately, there is little literature aimed at providing clear, general methods for developing such models at the systems level. The explicit methodology we provide, and the many examples we present, are intended to show precisely how these three principles can be used to build the kinds of models that experimental neuroscientists can exploit. To aid the construction of such models, we have developed a simulation environment for large-scale neural models that is available at `http://compneuro.uwaterloo.ca/`.

The second audience consists of the growing number of engineers, physicists, and computer scientists interested in learning more about how their quantitative tools relate to the brain. In our view, the major barrier these researchers face in applying proven mathematical techniques to neurobiological systems is an appreciation of the important *differences* between biological and traditionally engineered systems. We provide quantitative examples, and discuss how to understand biological systems using the familiar techniques of linear algebra, signal processing, control theory, and statistical inference. As well, the examples we present give a sense of which neural systems are appropriate targets for particular kinds of computational modeling, and how to go about modeling such systems; this is important for those readers less familiar with the neurosciences in general.

Our third audience is the computational neuroscience community; i.e., those familiar with the kind of approach we are taking towards characterizing neurobiological systems. Because we claim to develop a general approach to understanding neural systems, we suspect that researchers already familiar with the current state of computational neuroscience may be interested in our particular synthesis, and our various extensions of current results. These readers will be most interested in how we bring together considerations of single neuron signal processing and population codes, how we characterize neural systems as (time-varying nonlinear) control structures, and how we apply our techniques for generating large-scale, realistic simulations. As well, we present a number of novel models of commonly modeled systems (e.g., the lamprey locomotor system, the vestibular system, and working memory systems) which should provide these readers with a means of comparing our framework to other approaches.

Computational neuroscience is a rapidly expanding field, with new books being published at a furious rate. However, we think, as do others, that there is still something missing: a general, unified framework (see section 1.6 for further discussion). For instance, past books on neural coding tend to focus on the analysis of individual spiking neurons (or small groups of neurons), and texts on simulation techniques in neuroscience focus either at that same low level or on higher-level cognitive models. In contrast, we attempt to bridge the gap between low-level and high-level modeling. As well, we do not focus on models of a specific neural system as a number of recent books have, but rather on principles and methods for modeling and understanding diverse systems. Furthermore, this work is not a collection of previously published papers, or an edited volume consisting of many, often conflicting, perspectives. Rather, it presents a single, coherent picture of how to understand neural function from single cells to complex networks. Lastly, books intended as general overviews of the field tend to provide a summary of common single cell models, representational assumptions, and analytical and modeling techniques. We have chosen to present only that material relevant to constructing a unified framework. We do not want to insinuate that these various approaches are not essential; indeed we draw very heavily on much of this work. However, these are not attempts to provide a *unified* framework—one which *synthesizes* common models, assumptions, and techniques—for understanding neural systems. We, in contrast, have this as a central goal.

The fact that there are so many book-length discussions of the topics we address should serve as a warning: we are not providing a comprehensive overview of computational neuroscience. But, given the prevalence of recent calls for a neuroscientific theory, we think that trying to construct a unified approach is a useful undertaking. There are points in the book where we make strong claims regarding the ability of the framework (and underlying theory) to unify our understanding of neural systems. Often we do this for rhetorical reasons: stating the strongest position is often a way to make the position clear. When pushed, however, we refer to this framework as a 'zeroth-order guess' about how neural systems function. We think, then, that there is lots of work left to do (see `http://compneuro.uwaterloo.ca/` for a long list). Nevertheless, we feel that even a zeroth-order guess is better than no guess at all, and is likely, once articulated, to lead to better guesses. As a result, even if we turn out to be terribly wrong, we do think this framework is useful for organizing ideas about neurobiological systems. It has proven immensely useful to us, we sincerely hope others will find some utility in it as well.

This page intentionally left blank

Using this book as a course text

The purpose of this book is to introduce our framework to currently active researchers in neuroscience, and to those who may be interested in moving into the field (e.g., engineers, physicists, or biologists). However, because there are not overly many textbooks covering this material, and because we attempt to integrate many current ideas into a coherent picture of neurobiological systems, we have found that this material is well-suited for use as a course text. As such, we have developed a curriculum and programming tools for a graduate-level course in computational neuroscience based on this book. These materials include problem sets and solutions, course notes, examples, and a code library written in MatLab®. This material is available at `http://compneuro.uwaterloo.ca/`.

Students who completed the course have had diverse backgrounds and included psychologists, physicists, engineers, mathematicians, and neuroscientists. Because this book is intended for a wide audience, we often provide a brief introduction to topics relevant to an analysis or simulation we are presenting. The inclusion of such introductory material helps the book function as a course text. It also means that the prerequisites for a course based on the book can be kept minimal. We suggest that students be familiar with linear algebra and calculus. As well, familiarity with neuroscientific methodology, Fourier transforms, Laplace transforms, and some basic MatLab skills serve to make the course more manageable, but are not mandatory. Lastly, if using the book as a text, we suggest changing the order in which the chapters are presented. This allows students to begin familiarizing themselves with simulation techniques earlier in the course. As well, it allows students to understand basic transformations before being confronted with complex representations. For these reasons we prefer to present the chapters in the following order in a course setting: 2, 4, 5, 6, 3, 7, 8, 9.

This page intentionally left blank

Acknowledgments

We are indebted to a great many people who have helped shaped our efforts—in many different ways.

I am confident that this book would never have been finished without the support, inspiration, and patient ears of my friends and family: Jerry and Janet Elias, Steve, Kim, Michael, and Jessica Elias, Barb, Gord, and Jodi Smith, and Jen, Alana, and Alex Eliasmith. Thank you all so much. For their contributions 'above and beyond the call of duty,' special thanks to Janet Elias and Jen Eliasmith. As well, the unerring guidance and support provided by William Bechtel and Paul Thagard made this book a possibility. Finally, valuable interactions with Chris Lee, Pete Mandik, Whit Schonbein, Chase Wrenn, and Tadeusz Zawidski have significantly helped shape the final product.

—Chris Eliasmith

The encouragement and thoughtful suggestions of David Van Essen over the years of our collaboration have significantly influenced the development of the ideas presented in this book. My thanks to him for his support and for his friendship. I am pleased to acknowledge the important assistance of dedicated and enthusiastic graduate students (in chronological order): Bruno Olshausen, Subrata Rakshit, Hayit Greenspan, Michael Barber, Shahin Hakimian, Chris Eliasmith, Brandon Westover and Brian Fischer. In addition, the members of the Computational Neuroscience Research Group (CNRG) at Washington University, Professor John Clark, Qingfeng Huang, Zoran Nenadic, Ali Mahani, Harris Nover, and Libby Buchholtz, provided many hours of discussion, dedicated work, and significant intellectual contributions. Finally, I wish to thank Pat Anderson for her support over the many years we have been together.

—Charles H. Anderson

This book is by no means the result of the work of just two people. Many individuals have contributed directly through their hard work or indirectly through challenging and thought-provoking discussions or, in many cases, both. It is evidence of the interdisciplinary nature of this kind of work that their ranks include physicists, mathematicians, engineers, neuroscientists, and philosophers. Here they appear together: Kathleen Akins and the members of the McDonnell Project in Philosophy and the Neurosciences, Dora Angelaki, William Bechtel, Andy Clark, Carson Chow, Pat Churchland, Greg DeAngelis, Peter Dayan, Jay Hegde, Steve Highstein, Brody Johnson, Dan Marcus, John Miller, Bruno Olshausen, Alex Pouget, Jesse Prinz, Subrata Rakshit, Dave Redish, Sebastian Seung, Larry Snyder, David Tank, Tom Thach, Dave Touretzky, Todd Troyer, Rachel Wong, and Rich Zemel.

Much of the work that underlies various sections of this book has been done by members of the CNRG. In particular, we would like to thank Michael Barber (sections 9.2 and 3.2), Libby Buchholtz (sections 6.4 and 8.5), Shahin Hakimian (section 6.3), Qingfeng Huang (section 5.2), Ali Mahani (section 9.5.1), and Brandon Westover (sections 4.3, 7.3, 8.5, and 9.4). Without their dedication and valuable discussion, much of this material would not have developed as it has.

Few of the pieces of this project would have fallen into place in such a timely manner were it not for the assistance of MIT Press, especially Barbara Murphy, Sara Meirowitz, and Mike Rutter. As well, many of the pieces would not have been there to fall into place without the guidance or data provided for various sections by Kathleen E. Cullen, Steve Elias, Bard Ermentrout, Boris Gutkin, Christof Koch, and Hugh Wilson. Finally, we would not have had the time and resources to even begin the book without the generous funding of the National Science Foundation, the McDonnell Project in Philosophy and the Neurosciences, the McDonnell Center for Higher Brain Function, the Social Sciences and Humanities Research Council of Canada, and the Mather's Foundation.

Despite these many helping hands, we have no doubt managed to insert various errors. These alone are solely our responsibility.

—*Chris Eliasmith* and *Charles H. Anderson*, August 15, 2002

1 Of neurons and engineers

Neurons are fascinating, wildly diverse devices. Some, like the giraffe primary afferent, have axons that are 15 feet long. Others, like the typical granule cell, have axons that are only about 100 microns long. Some, like the common pyramidal cells, produce and transmit stereotypical action potentials (neural spikes) down their axon. Others, like the retinal horizontal cells, communicate without such spikes. Some send action potentials at speeds over 400 km/h. Others transmit spikes at a mere 2 km/h. Some, like the purkinje cells, have about 200,000 input connections. Others, like the retinal ganglion cells, have only about 500 such connections. There are literally hundreds and hundreds—by some estimates thousands—of different *kinds* of neurons in the brain.[1] And, taken together their numbers are staggering: 10^{10} neurons in the human brain; at least 10^{13} synapses; 45 miles of fiber; 100 different kinds of neurotransmitters. Such facts simply impress most people, but they can be exceedingly daunting to those interested in explaining and understanding neural systems.

How are we to get an explanatory grip on any system that is this complex? In this book we pursue the idea that the quantitative tools typically associated with the field of engineering hold great promise for understanding complex neural systems. Many of these tools have their roots in theoretical physics, mathematics, and even pure logic, but it is typically in the hands of engineers that they are applied to physical systems like the brain. Once we consider brains as purely physical devices, it becomes perfectly natural to reach into the engineer's toolbox and grab ahold of information theory, control theory, signal processing theory, and a host of other such formalisms in order to understand, characterize, and explain the function of such systems. It is, after all, a *functional* understanding of the brain that neuroscience is after—at least part of what neuroscientists want to know is how neurons cooperate to give rise to complex behavior—and it is characterizing function that is the engineer's primary concern. Thus, the theoretical tools of engineering are likely well-suited to helping us figure out what all of those neurons are doing.

Of course, these tools cannot be *blindly* applied to characterize neurobiological systems. For instance, merely reproducing, at some abstract level, a function of the brain (such as playing chess) is not enough for neuroscience (although it might be for artificial intelligence). Neuroscientists are interested in knowing how *neurons* give rise to brain function. For their purposes, the vast amounts of knowledge they have culled in recent decades must be respected when trying to understand neural systems; such neurobiological constraints are some of the best constraints we have when trying to understand how brains give rise to behavior. To illustrate, it is standard practise in communications theory to assume that

1 There are at least 23 kinds of amacrine cells alone.

incoming signals are evenly sampled over the range of the signal. A number of optimal decoding strategies have been devised that take advantage of this particular kind of sampling. However, this kind of sampling is clearly *not* present in neural systems. Sampling densities often change over the range of the signal, and the particular sample points can be quite different even between two members of the same species. This is because neurons are diverse, unlike the typical components of artifactual communications systems. Thus, looking to the actual constraints respected (or not) by neurobiological systems is important for applying various mathematical tools in the appropriate manner.

In what follows, we are committed to devising a neuroscientifically respectable approach to modeling neurobiological systems. Being concerned with actual neural systems means embracing all of the ‘messiness’ of such systems. Such messiness exists because brains are *natural* physical systems whose functioning is exquisitely shaped by real-world environments. Unlike their artifactual counterparts, they have not been designed to function like theoretical computational systems such as Turing machines. This does not mean that computational theory isn’t useful, it just means that it shouldn’t be taken at face value. We need to adopt *and adapt* the engineer’s tools in order to successfully quantify the function of neural systems. So, while brains are first and foremost *implemented* computational systems, and there are extensive tools that have been developed for quantifying such computational systems, brains respect a set of design constraints significantly different from those typically respected by engineered systems. Like mother nature, an engineer is a practical scientist who has to build things that work. But mother nature, unlike an engineer, does not rely on carefully manufactured, fault tolerant, and nearly identical parts.

These considerations will come as no surprise to some. Indeed, engineering tools have been successfully applied to neural systems for many years (Fitzhugh 1958). Information theoretical analyses of spike trains (Rieke et al. 1997), basis function characterizations of neural representations (Poggio 1990), and control theoretic characterizations of neural systems (Wolpert and Kawato 1998), have all resulted in sophisticated, biologically constrained computational models of neurobiological systems. However, much of this work has been carried out in relative isolation: those using information theory do not seem to be talking to those doing basis function research or using control theory, and vice versa. One of our main goals in this book is to provide a *synthesis* of these approaches to understanding the brain. Although determining how such seemingly disparate research programs are related often requires significant extrapolation, we are not trying to provide new tools. Rather, we are trying to articulate a *new way* to use them together, and we are hoping to make them available to *new users*.

In order to accomplish these goals we adopt the terminology of both engineers and neuroscientists, but we assume minimal background in either engineering or neuroscience on the part of the reader. So, when introducing our framework, or discussing specific examples of its application, we provide: 1) the relevant neuroscientific details needed to understand the new way of applying these tools to neurobiological systems; *and* 2) the relevant technical details needed to introduce these tools to new users. We have taken this approach because we think the communication between neuroscientists and engineers needs to be strengthened. We think there are likely benefits to both parties: engineers may come to understand novel computational strategies and sharpen existing analytical tools; and neuroscientists may come to a more precise understanding of neurobiological function and be better able to direct future experimentation.

These are goals that can be accomplished without a perfect understanding of neurobiological systems; which is why we have some hope of success. As necessary, we make simplifications, approximations, and guesses that, in the long run, destine our models to function differently than the real neural systems they are intended to model.[2] These assumptions are generally due either to a lack of neuroscientific knowledge, or a limitation of current engineering tools. This is, we take it, further evidence that neuroscience and engineering are in an excellent position to be mutually beneficial. In particular, making assumptions has clear benefits to both engineers and neuroscientists as they can guide experiments to gain the facts that are missing, help determine what kinds of facts are the most useful, tell us what kinds of analytical tools are needed, and help us better understand how to apply the tools available.

In sum, what we present in this book is a principled, unified approach to understanding neurobiological systems that employs the quantitative tools of engineers, respects neuroscientific results, and should be of mutual benefit to neuroscientists and engineers. In the remainder of this chapter, we provide an overview of our approach. In the remainder of the book, we provide the details and examples that demonstrate how the approach described in this first chapter is supposed to work.

1.1 Explaining neural systems

Neural systems are amazingly adept at solving problems. Seagulls open shellfish by dropping them on rocks, or even in front of moving cars (Grandin and Deesing 1998). Bees excel at finding their way through natural environments, and communicating what they learned on the trip (Gould 1975). Rats have an excellent sense of direction, even in complete darkness (Redish 1999). Primates are so good at dealing with problems that they have learned to fashion tools to help themselves cope (McGrew 1992). Being interested in neurobiology means being interested in providing explanations of how these diverse

2 This too, may be considered in the spirit of engineering. As the apocryphal bumper sticker famously notes: "Engineers are approximately perfϖct".

kinds of nervous systems give rise to such interesting, proficient behaviors. Of course, neuroscientists and others have been providing such explanations, whether good or bad, for decades. One fairly typical aspect of these explanations is that they invoke the notion of *representation*. Representations, broadly speaking, serve to relate the internal state of the animal to its environment; they are often said to 'stand-in for' some external state of affairs. At least since Hubel and Wiesel (1962), neuroscientists have talked of neurons *representing* their environment. Current discussions in neuroscience continue to rely heavily on the notion (see, e.g., Desimone 1991; Felleman and Essen 1991).[3]

Neuroscientists are by no means the only scientists to use the term 'representation' in this way. Indeed, the notion of a 'mental representation' has a two thousand year history in Western thought (Watson 1995). The explanatory power of representations comes from the fact that they can be manipulated internally without manipulating the actual, external, represented object. The benefits of being able to do this range from simply saving energy to being able to generate life-saving predictions. In general, then, representations help solve problems. So, it is hardly surprising that this notion is used by neuroscientists to explain the behavior of neurobiological systems.[4]

While representations are an important ingredient of the kinds of explanations that neuroscientists are interested in, they are not the only ingredient. There would be little interesting behavior produced by a neurobiological system that was simply 'filled' with representations. Rather, we also need to understand how representations are manipulated, exploited, related, updated, and so on. That is, we need to characterize how representations are *transformed* such that they are useful. In order to recognize an object, for example, the retinal image must, in some sense, be compared to past experiences (hence '*re*-cognize'). Those past experiences are not stored as retinal images. Instead, they have been transformed into a form that aids recognition; irrelevant features are ignored, noise is eliminated, and metrics along various dimensions are extracted. As well, similar transformations must be carried out on the current retinal image in order for such a comparison to occur. All of this work can be understood in terms of transformations from one representation into another. Consider also a more behavior-oriented example; reaching for an object. In order to know where the object is, an animal must rely on information regarding the orientation

3 A search of PubMed abstracts using "representation and neurons" results in over two thousand hits (see `http://www.ncbi.nlm.nih.gov/PubMed/`).

4 Of course, representational explanations are not the *only* kinds of explanations that neuroscientists are interested in. There are numerous questions concerning how neurobiological systems develop, what kinds of chemicals affect neural functioning, how the basic biological mechanisms supporting neuron activity function, and so on. These kinds of questions may need different kinds of explanations. However, the focus of this book is on what is often called 'systems' neuroscience, the part of neuroscience concerned with explaining the behavior of the (average, adult) neurobiological system in terms of its components. In this context, representational explanations so far seem to be the best kind of explanation available.

of the eyes, head, and body. In this case we can think of the retinal image as presenting the initial 'eye-centered' information regarding the position of the object, which needs to be eventually translated into arm-centered information in order to give rise to the appropriate reaching action. Again, this kind of change of coordinate systems can be understood in terms of the transformation of internal, neural representations.

So, we take the central problem facing neuroscientists to be one of *explaining how neurobiological systems represent the world, and how they use those representations, via transformations, to guide behavior*. As a result, we have divided the book into two main parts. The first part, consisting of chapters 2–5, is concerned with characterizing neural representations. The second part, consisting of chapters 6–9, is concerned with understanding transformations of such representations. In the next two sections we provide a brief overview of our approach to this problem. As will become clear, despite this preliminary distinction between representation and transformation, they are intimately related.

1.2 Neural representation

The main problem regarding mental representation, both historically and for contemporary philosophers of mind, is to determine the exact nature of the representation *relation*; that is, to specify the relation between, and representationally relevant properties of, things 'inside the head' and things 'outside the head'. The traditional approach to solving this problem is to consider the data (broadly construed) available from metaphysics, introspection, psychology, and more recently neuroscience, in order to deduce: 1) the representation relation; and 2) ways of determining what things are representations and what things are represented. However, we do not take this approach. Rather, we define outright the representational relationship and its relata, and see if our definition does the explanatory work that is needed. We happen to think that, as a matter of fact, our definition of representation does speak to the problems of mental representation that arise when adopting the traditional approach, but showing that is beyond the scope of this book (see Eliasmith 2000 for a detailed discussion). So, we will not discuss, except in passing, how to determine what things are representations and what things are represented. Rather, we take the standard kinds of representational claims that neuroscientists make more-or-less at face value, and show how our account makes those claims precise. So, for example, we take it (at least initially) that neural firings represent the stimulus that causes them, and we presume that neural populations represent the external objects or variables that the population activity is correlated with. In both cases, we use our definition to give an explanatorily and predictively useful account of what it means to make those claims. And, in both cases, we adjust the claims to make them more theoretically sound.

In order to precisely define the representation relation we turn to the quantitative tools of engineering. Specifically, we believe that there is a close tie between neural representations as understood by neuroscientists and *codes* as understood by communications engineers (Reza 1961). Codes, in engineering, are defined in terms of a complimentary *encoding* and *decoding* procedure between two *alphabets*. Morse code, for example, is defined by the one-to-one relation between letters of the Roman alphabet, and the alphabet composed of a standard set of dashes and dots. The encoding procedure is the mapping from the Roman alphabet to the Morse code alphabet and the decoding procedure is its inverse (i.e., the mapping from the Morse code alphabet to the Roman alphabet).

Representations, in neuroscience, are seldom so precisely defined but there are some commonalities between this notion of a code and the typical use of the term 'representation'. For instance, the term 'representation' is usually used to denote one or more neural firings from one or more neurons. For example, a neuron is said to represent a face at a certain orientation if it fires most strongly when the animal is presented with a face at that orientation (Desimone 1991). However, neural representations tend to be *graded* representations; i.e., a neuron fires more or less strongly depending on the nearness of the stimulus to what is called its 'preferred' stimulus (that is, the stimulus that causes it to have the highest firing rate). Neuroscientists are interested in characterizing the relation between this graded representation and stimuli that evoke it. In fact, this standard description of neural behavior maps quite nicely onto the notion of *encoding* in engineering: neural firings encode properties of external stimuli.

However, in order to characterize the representation relation in a manner similar to the way communications engineers characterize codes, we must also identify the decoding procedure. This is a surprisingly natural constraint on representation in neural systems. If there is a neuron that initially seems to fire most strongly to a face at a certain orientation, then it must be *used* by 'downstream' processes in order for the claim that it actually *represents* that orientation to make any sense. If the previously described neural firing was, in fact, used by the system to determine eye orientation (and it just so happened that in the initial experiment face orientation and eye orientation covaried because the eyes were always facing forward), then the best case could be made for it representing eye orientation (given subsequent, better controlled experiments), and not face orientation. Of course, what it means for a downstream system to *use* these neural firings is that the downstream system can extract (i.e., *decode*) that particular information (e.g., eye orientation) from those neural firings. This holds as well for graded representations as for any other kind of representation. So decoding is as important to characterizing neural representations as encoding (we return to these issues in chapter 6).

Now, in order to fully characterize neural representation as a code, we need to specify the relevant alphabets. A problem arises for neural representation here because there are so many different possible alphabets that can be identified. For instance, if we are interested in the representations of individual retinal ganglion cells, it seems that something like light intensities at certain retinal locations and spike trains of single neurons would be the relevant alphabets. In contrast, if we are interested an entire cortical area, like the primary visual cortex, it seems that something more like color, spatial frequency, intensity, etc. over the whole visual field and spike trains of large populations of neurons would be the relevant alphabets. The alphabets in these cases are extremely different. Nevertheless, if we are interested in a general understanding of neural representation (i.e., a general means of finding encoding and decoding rules), we *can* find general commonalities between these alphabets. Namely, we can understand these behaviors as relating *neural responses* (alphabet 1) and *physical properties* (alphabet 2).

In fact, we think it is possible to be a bit more specific. Neuroscientists generally agree that the basic element of the neural alphabet is the neural spike (see, e.g., Martin 1991, p. 334). However, it may be that the neural alphabets that are actually used include the average production rate of neural spikes (i.e., a rate code), specific timings of neural spikes (i.e., a timing code), population-wide groupings of neural spikes (i.e., a population code), or the synchrony of neural spikes across neurons (i.e., a synchrony code). Of these possibilities, arguably the best evidence exists for a combination of timing codes (see Rieke et al. 1997 for an overview) and population codes (see Salinas and Abbott 1994; Seung and Sompolinsky 1993; and Abbott 1994 for an overview).[5] For this reason, we take these two kinds of basic coding schemes to comprise the alphabet of neural responses (i.e., alphabet 1).

However, it is much more difficult to be any more specific about the nature of the alphabet of physical properties. We can begin by looking to the physical sciences for categories of physical properties that might be encoded by nervous systems. Indeed, we find that many of the properties that physicists traditionally identify do seem to be represented in nervous systems; e.g., displacement, velocity, acceleration, wavelength, temperature, pressure, mass, etc. But, there are many physical properties not discussed by physicists which also seem to be encoded in nervous systems; e.g., red, hot, square, dangerous, edible, object, conspecific, etc. Presumably, all of these latter 'higher-order' properties are *inferred* on the basis of (i.e., are the results of transformations of) representations of properties more like those that physicists talk about. In other words, encodings of 'edible' depend, in some complex way, on encodings of more basic physical properties like wavelength, temperature, etc. Given this assumption, our focus is (as it is in neuroscience, generally) on trying to understand how nervous systems encode the more basic physical properties. Eventually,

5 For a demonstration that rate codes are a specific instance of timing codes see Rieke et al. (1997). For empirical evidence that synchrony codes are not used by nervous systems see Kiper et al. (1996) and Hardcastle (1997).

however, we need to be able to explain how the property of being edible, or being an object is encoded by neurons. We think that this framework is flexible enough to help with this problem, but for the time being we focus our attention on characterizing more basic physical properties, where we believe successes can be more convincingly demonstrated.

This, then, is how 'neural representation' in neuroscience can be defined analogously to how 'codes' are defined in communications engineering. To recap, neurons encode physical properties into population-temporal neural activities that can be decoded to retrieve the relevant information. However, there are also important differences between engineered codes and neural representations. Perhaps most importantly, the former are *specified* whereas the latter are *discovered*. So there can be (and should be) a lot of debate concerning what is *actually* represented by a given neuron or neural population. Even without wading into the philosophical quicksand that surrounds such representational claims, we can discover something important about the proposed definition of representation. That is, there is bound to be some *arbitrariness* to the decoding we pick and thus the claims that we make regarding representation in neural populations. This is especially true at the beginning of our explorations of neural systems.

There are two sources of this seeming arbitrariness. First, since knowing what is represented depends in part on how it is subsequently used, it seems like we already have to know how the system works in order to know what it represents. But, of course, how the system works is precisely what we are trying to figure out when we are talking about representation. This problem seems much less of an obstacle once we realize that many difficult explanatory problems are resolved by making guesses about how things work and then testing those guesses (this is just the idealized 'scientific method'). So, although the 'fact of the matter' about what is represented will only be resolved once we have a fairly comprehensive understanding of what is actually represented in the brain, this should not be taken to be an insurmountable difficulty, or viciously circular. Specifically, it does not mean that our interim hypotheses are wrong, just that they might be (which is why they are *hypotheses*). In effect, what is 'really' represented is whatever is taken to be represented in a more-or-less complete, coherent, consistent, and useful theory of total brain function—a theory at least many decades away. But this does not mean that it is pointless to make representational claims now; in fact, making such claims is an essential step towards such a theory.

The second source of seeming arbitrariness stems from the fact that information encoded by a neural population may be decoded in a variety of ways. To see why, consider a neural population that encodes eye velocity. Not surprisingly, we can decode the information carried by this population to give us an estimate of eye velocity. However, we can also decode that same information to give us an estimate of a *function of* eye velocity (e.g., the square). This is because we can essentially 'weight' the information however we see fit when decoding the population activity; different weightings result in different decoded functions. Since representation is defined in terms of encoding *and* decoding, it seems that we need a way to pick which of these possible decodings is the relevant one for defining *the* representation in the original population. We resolve this issue by specifying that what a population represents is determined by the decoding that results in the quantity that all other decodings are functions of. We call this the '*representational decoding*'. Thus, in this example, the population would be said to represent eye velocity because eye velocity and the square *of eye velocity* are decoded. Things are not quite so simple because, of course, eye velocity is also a function of the square of eye velocity. This problem can be resolved by recalling considerations brought to bear on the first source of ambiguity; namely that the right physical quantities for representation are those that are part of a coherent, consistent, and useful theory. Because physics (a coherent, consistent, and useful theory) quantifies over velocities (and not squares of velocities), so should neuroscience (as this renders science as a whole more coherent, consistent and useful).

While these considerations may seem distant from empirical neuroscience, they play an important role in specifying what is meant by claims that a neuron or neural population is representing the environment; claims that empirical neuroscientists make all the time. So, throughout the course of the book we revisit these issues as various examples and analyses are presented that provide greater insight into the nature of neural representation.

In sum, there are good reasons to think that neuroscientists can and should rigorously define the notion of neural representation along the lines that engineers have defined the notion of codes.[6] Specifically, both encoding and decoding are important for defining neural representation, and the relevant alphabets are neural activities and physical properties. There are important differences between neural representation and engineered codes that raise important theoretical issues regarding the nature of neural representation. Nevertheless, considerations from engineering provide an excellent starting point for characterizing neural representation. In the next two sections we present more detail on how coding theory can play this role.

1.2.1 The single neuron

To adopt an engineering perspective regarding neural function, we can begin by asking: What kind of physical devices are neurons? Fortunately, there is some consensus on an answer to this question; neurons are electro-chemical devices. Simpler yet, the behavior of single neurons can be well-characterized in terms of their *electrical* properties. In fact, detailed, quantitative, and highly accurate models of single cell behavior have been around

6 We are by no means the first to suggest this (see, e.g., Fitzhugh 1958; Rieke et al. 1997; Abbott 1994; Seung and Sompolinsky 1993).

for about 50 years, and continue to be improved (see Bower and Beeman 1995 for a history and some recent models). By far the majority of vertebrate neurons can be understood as physical devices that convert an ‘input’ voltage change on their dendrites into an ‘output’ voltage spike train that travels down their axon. This spike train is the result of a highly nonlinear process in the neuron’s soma that relies on the interaction of different kinds of voltage-gated ion channels in the cell membrane. These spike trains then cause further electrical changes in the dendrites of receiving (i.e., postsynaptic) neurons.

The dendrites themselves have active electrical properties similar to those in the soma, which result in a current flowing to the soma from the dendrites.[7] The soma voltage itself is determined by the current flowing into the soma from the dendrites, as well as active and passive membrane resistances and a passive membrane capacitance. Typically, the passive membrane time constant in the soma (determined by the passive resistance and capacitance) is on the order of about 10 ms. This time constant effectively characterizes the time-scale of the ‘memory’ of the soma with respect to past signals. Thus, the signal generation process only ‘remembers’ (or is sensitive to) dendritic inputs that occurred in the very recent past—perhaps as long ago as 200 ms, but more typically in the tens of milliseconds.[8] So, considered as electrical devices, neurons have highly nonlinear input processes (at the dendrites), highly nonlinear output processes (in the soma, resulting in voltage spike trains), and a fairly short signal memory (in both dendrites and at the soma).

Because neurons can be characterized as electrical devices that transmit signals in this way, it is fairly natural to analyze their behavior using the tools of signal processing theory. In other words, given this characterization neurons can be directly analyzed as information processing devices. Adopting this perspective leads to the insight that neurons are effectively *low-precision* electrical devices that transmit about 1–3 bits of information per neural spike, or a few hundred to a thousand bits per second (see section 4.4.2). This means that modeling their output using 16 bit floating point numbers updated at hundreds of *mega*hertz—as those in the artificial neural net community tend to do—is not a good way to characterize real, *neurobiological* representation. We are concerned with neural representation only from this combination of a biological and engineering perspective; i.e., understanding real neurons as electrical devices.

7 Although we begin by assuming that dendritic contributions are linear, we discuss a means of modeling the likely dendritic nonlinearities in section 6.3.

8 Of course, the dendrites themselves have a longer memory, commonly characterized by a synaptic weight (which can change due to learning). However, even in the dendrites, there is a sense in which the memory of recent electrical input decays within about 200 ms. Specifically, if we distinguish the synaptic weight (which changes slowly) from the postsynaptic current (which changes rapidly as spikes are received), as most models do, then the memory of activity is short (i.e., on the order of the time constant of the postsynaptic current). Because synaptic weights change slowly, they can be considered fixed on the time-scale of the synaptic currents and somatic spikes. Given the large difference in time scales, synaptic weights can therefore be considered separately from dendritic postsynaptic currents. In particular, the weights can be treated as a simple multiplier, or gain, on a stereotypical postsynaptic response.

Nevertheless, it is useful to examine how real neurons, considered as information processors, are analogous to transistors on silicon computer chips. Like transistors, neurons are: 1) electrical devices; 2) highly nonlinear; and 3) signal/information processors. These similarities make it very natural to use the kinds of mathematical tools that have been used by engineers in the past to understand transistor signal processing in order to understand neurons. However, we must bear in mind that, unlike transistors, neurons: 1) have very short memories; 2) output voltage spikes; 3) are heterogeneous; and 4) are biological (i.e., not manufactured). These differences again make it clear that we have to be careful exactly how we apply engineering tools to understanding neurobiology. In chapter 4, we discuss a means of characterizing neurons as signal processors with these differences in mind and using both simple and realistic neural models. Because of the diversity of models we consider, it becomes clear that our approach does not rely on the functioning of specific kinds of neurons (or neural models). This helps make the approach a general one. And, more importantly, the insights gained by understanding single neurons as information processors are essential for putting them together to build realistic, large-scale models.

1.2.2 Beyond the single neuron

Despite our discussion of single neuron function, the focus of this book is definitively *not* on the analysis of single neurons (*c.f.* Koch 1999; Wilson 1999b; Rieke et al. 1997). Rather, it is on understanding how groups or populations of neurons can be characterized as working together to support neural representation and transformation. To this end, we begin our discussion of representation with population-level considerations (in chapter 2) before worrying about the details of information processing in individual neurons (in chapter 4). After developing these two aspects separately, we combine them to give a general description of ‘population-temporal’ representation (in chapter 5). As we show, describing population coding more-or-less independently of the specifics of individual neuron function provides the flexibility to model many levels of representational detail concurrently in a single model.

Because our focus is on population representation, we go to some length to show that having to depend on highly nonlinear processing elements (i.e., the neurons), does not necessarily result in difficult-to-analyze representations. This result is not too surprising if we again consider the analogy between neurons and transistors. Binary representations in computers rely on encoding signals into ‘populations’ of highly nonlinear processing elements (i.e., the transistors). Nevertheless, these signals are decoded using a simple linear decoder (see section 2.1.1). Similarly, we show that considering populations of nonlinear neurons can result in good, simple, linearly decodable representations (see section 4.3.2). Furthermore, as more neurons are included in the representation, it becomes even better; but only if the resulting population is *heterogeneous* (i.e., has a range of neuron properties).

In other words, population activity can be linearly decoded to give an increasingly accurate indication of what was originally (nonlinearly) encoded, with an increasing number of heterogeneous neurons.

There are two important results here. First, it is extremely fortunate that we are able to extract the information that was nonlinearly encoded using a linear decoder because that allows many of the tools of linear signals and systems theory, a very well-developed and understood field in engineering, to be at our disposal. Second, the often underemphasized property of the heterogeneity of neural systems become central. Given this perspective, the heterogeneity of neural populations can be explained from a functional point of view; in fact, heterogeneity becomes indispensable for a good representation (see section 7.5). This has significant consequences for both experimentalists and theoreticians. Specifically, this result shows that it is more appropriate for experimentalists to report the *distributions* of neuron response properties, rather than presenting a few 'typical' (i.e., best) neurons in detail. It is more appropriate because it is the distribution that provides insight into the kind of representation employed by, and the function of, the system under study (see sections 7.4 and 7.3). For theoreticians this result means that assuming that every neuron in a population is identical is going to give significantly misleading results, despite such assumptions making the system mathematically simpler to handle.

The sensitivity of neural representation to population-level properties like heterogeneity and the number of neurons suggests that it is most useful to think of neural representation in terms of populations, rather than in terms of single neurons. Thus, we argue that it the fundamental unit of signal processing in the nervous system is the simplest neural population (a neuron pair), rather than the single neuron (see section 4.3.2).

Adopting this perspective on neural representation has some useful pragmatic results. In particular, focusing on population coding permits us to consider a given model with varying degrees of detail. We can, for instance, build a simulation using only population representations, ignoring the details of individual neurons. Or, we can build the same simulation using neural-level representations and including whatever degree of biological detail is appropriate. Or, we can build that same simulation using *both* kinds of representation concurrently.[9] This flexibility is useful because it addresses an important need in neuroscience: "Ideally, we wish to be able to move smoothly between levels of models and to understand how to reduce systematically more complex models into simpler forms that retain their essential properties" (Marder et al. 1997, p. 143). Our analysis of representation addresses this need by concurrently supporting various levels of representation; from highly abstract population representations to the single cell representations of detailed con-

9 The simulation package that we have released with this book supports such multi-level simulations. It can be found at `http://compneuro.uwaterloo.ca`.

ductance models. When formalizing these representational levels, we discuss how to define a representational hierarchy that spans the levels of biological structure from single neurons through networks and maps to brain areas. To preview, this hierarchy begins with scalar representations, like those found in nuclei prepositus hypoglossi for controlling horizontal eye position (sections 2.3, 5.3, and 8.2). It then incorporates slightly more complex vector representations, like those found in motor cortex for controlling arm motion (section 2.5) and those found in vestibular nucleus for representing angular velocity and linear acceleration of the head (sections 6.5). Lastly, we use the hierarchy to characterize functional representations, like those found in lateral intraparietal cortex (sections 3.4 and 8.3). Despite our not providing examples of the hierarchy past this specific representational level, we show how it is straightforward to generalize these examples to more complex representations (like vector fields). While it is unlikely that there is a precise relation between such a representational hierarchy and biological structure, being able to build a general and flexible hierarchy proves useful for characterizing such structure at many different levels.

1.3 Neural transformation

In part II of this book we show that our characterization of neural representation paves the way for a useful understanding of neural transformation. This is largely because transformation, like representation, can be characterized using linear decoding. However, rather than using the *representational decoder* discussed earlier, we use what we call a *transformational decoder*. The contrast between these two kinds of decoders lies in the fact that, when performing a transformation on encoded information, we are attempting to extract information *other* than what the population is taken to represent. Transformational decoding, then, is not a ‘pure’ decoding of the encoded information. So, for example, if we think that the quantity x is encoded in some neural population, when defining the *representation* we identify the decoders that estimate x (i.e., the information that is taken to be ‘primarily’ encoded by that population). However, when defining a *transformation* we identify decoders that estimate some function, $f(x)$, of the represented quantity, x. In other words, we find decoders that, rather than extracting the signal represented by a population, extract some transformed version of that signal.

Defining transformations in this way allows us to use a slight variation of our representational analysis to determine what transformations a neural population can, in principle, support (see section 7.3). This allows us to determine how well a given neural population can support the transformations defined by a particular class of functions. This can be very important for constraining hypotheses about the functional role of particular neural populations that are observed in a neurobiological system. We show that neurons with certain response properties support particular transformations better than others. This is a good reason to think that populations with those properties are involved in computing certain transformations rather than others. In other words, this approach enables a good, quantitative characterization of the functional potential of sets of neurons.

Furthermore, defining transformations in this way permits us to *analytically* find connection weights in a biologically plausible network. So, rather than having to train a fully connected network using a learning rule (which might prove difficult if not impossible for large-scale simulations), we can define the representations we take to be in the system and the transformations that those representations undergo, and then directly find the weights to implement those transformations. This is beneficial in that it allows us to test explicit hypotheses about the function of a particular neural system, without having to worry about how to train that system to perform some function. As well, there is a significant practical benefit, in terms of computational savings, in not having to simulate large training regimes for complex models. Notably, this does not mean that learning is somehow antithetical to our approach (we discuss the relation of this approach to learning in chapter 9), it merely means that we do not *need* to rely on training to have interesting, biologically plausible models.

However, this is not the whole story about neural transformation. Transformations, as we have discussed them so far, are just like computing some function of x. However, this kind of static computation of a function is not, alone, a good description of the kinds of transformations neurobiological systems typically exhibit. Animals have evolved in a dynamic environment and are themselves dynamic systems. So, it is essential to be able to characterize the *dynamics* of the transformations that neural populations support. Again, engineers have a number of useful quantitative tools for describing dynamic systems. In particular, modern control theory has been successfully used by engineers to describe a huge variety of dynamic systems, both natural and artificial. In order to apply these same techniques to neural systems, we must identify the 'state vector' (i.e., the real-valued vector that tracks the system's internal variables) of the system. In chapter 8 we argue that neural representations can play this role. Given our analyses of neural representation, which include vector representation, this should come as no surprise. However, because neurons have intrinsic dynamics dictated by their particular physical characteristics, we must also *adapt* the standard control theory toolbox for understanding neurobiological systems (see section 8.1). Once this is done, we can directly apply the techniques for analyzing complex dynamic systems that have been developed by, and for, engineers.

Describing dynamics in this manner allows us to address issues that have proven very difficult for other approaches: "A great challenge for the future is to understand how the flexible modulation of motor circuits occurs without the loss of their essential stability" (Marder et al. 1997, p. 147). We attempt to meet this specific challenge in section 8.5.

More generally, because the stability (and various other high-level properties) of control systems is well understood, we believe that this approach to neural dynamics can help quantify our understanding of the dynamics of a wide range of neurobiological systems in an interesting, new way (see section 8.4).

1.4 Three principles of neural engineering

To this point in the chapter we have outlined, in very general terms, the approach that we develop in the remainder of the book. In this section, we consolidate these previous considerations by clearly stating the guiding assumptions of our approach. Much of our argument for these 'principles of neural engineering' is of the proof-is-in-the-pudding variety. That is, throughout the course of the book we provide numerous examples of detailed models of a wide variety of neurobiological systems that were constructed based on these principles. But, our goal is not to simply provide examples, rather it is to demonstrate how to *use* these principles. That is, we are interested in describing a framework for understanding and simulating neurobiological systems in general. Thus we provide not only this set of guiding principles, but also a *methodology* for applying these principles. In order to demonstrate this methodology, we follow it for each of the examples we present. It is important for our purposes that not only are a large number and variety of examples provided, but that they are also built with consistent assumptions and with a consistent methodology. Given our discussion to this point, these principles should be unsurprising:

Three principles of neural engineering

1. Neural representations are defined by the combination of nonlinear encoding (exemplified by neuron tuning curves) and weighted linear decoding (see chapters 2, 3, 4, and 5).
2. Transformations of neural representations are functions of variables that are represented by neural populations. Transformations are determined using an alternately weighted linear decoding (i.e., the transformational decoding as opposed to the representational decoding; see chapters 6 and 7).
3. Neural dynamics are characterized by considering neural representations as control theoretic state variables. Thus, the dynamics of neurobiological systems can be analyzed using control theory (see chapter 8).

We take these three principles to be excellent guiding assumptions for the construction of biologically plausible, large-scale simulations of neurobiological systems. While it is premature to state these principles more quantitatively, later on we will be in a position to do so (see section 8.1.4). In addition to these main principles, there is an important addendum which guides our analyses of neurobiological systems:

Addendum

4. Neural systems are subject to significant amounts of noise. Therefore, any analysis of such systems must account for the effects of noise (see sections 2.2, and 5.2).

We do not consider this addendum to be a principle because, rather than being a claim about how to *explain the functioning* of neurobiological systems, it is a claim about how to *analyze* such systems. Nevertheless, it is essential for articulating the principles in detail. In the next four sections, we briefly discuss the strengths of, and possible concerns with, these principles and the addendum.

1.4.1 Principle 1

Principle 1 emphasizes the importance of identifying both encoding and decoding when defining neural representation. Moreover, this principle highlights the central assumption that, despite a *nonlinear* encoding, *linear* decoding is valid (see Rieke et al. 1997, pp. 76–87). As discussed in detail by Rieke et al., a nonlinear response function like that of typical neurons is, in fact, unrelated to whether or not the resulting signal can be linearly decoded.[10] That is, the nature of the input/output function (i.e., encoding) of a device is independent of the decoder that is needed to estimate its input. This means that a nonlinear encoding could need a linear *or* nonlinear decoding, and vice versa. This is because the decoding depends on the conditional probability of input given the output *and* on the statistics of the noise (hence our addendum). Perhaps surprisingly, linear decoding works quite well in many neural systems. Specifically, the additional information gained with nonlinear decoding is generally less than 5%.

Of course, nonlinear decoding is able to do as well or better than linear decoding at extracting information, but the price paid in biological plausibility is generally thought to be quite high (see, e.g., Salinas and Abbott 1994). Furthermore, even if there initially seems to be a case in which nonlinear decoding is employed by a neural system, that decoding may, in the end, be explained by linear decoding. This is because, as we discuss in section 6.3, nonlinear transformations can be performed using *linear* decoding. Thus, assuming linear decoding at the neuron (or sub-neuron, see section 6.3) level can well be consistent with nonlinear decoding at the network (or neuron) level. So, especially in combination with principle 2, linear decoding is a good candidate for describing neural decoding in general.

It is important to emphasize that analyzing neurons as decoding signals using (optimal) linear or nonlinear filters does not mean that neurons are presumed to *explicitly use* opti-

10 During this same discussion, Rieke et al. mention that there are certain constraints on when linear decoding will work. In particular, they claim that there can be only a few, preferably one, spike(s) per correlation time of the signal. However, we have found that this not the case (see section 4.3.3).

mal filters. In fact, according to our account, there is no directly observable counterpart to these optimal decoders. Rather, the decoders are 'embedded' in the synaptic weights between neighboring neurons. That is, coupling weights of neighboring neurons indirectly reflect a particular population decoder, but they are not identical to the population decoder, nor can the decoder be unequivocally 'read-off' of the weights. This is because connection weights are determined by both the decoding of incoming signals *and* the encoding of the outgoing signals (see, e.g., section 6.2). Practically speaking, this means that changing a connection weight both changes the transformation being performed and the tuning curve of the receiving neuron. As is well known from work in artificial neural networks and computational neuroscience, this is exactly what should happen. In essence, the encoding/decoding distinction is not one that neurobiological systems need to respect in order to perform their functions, but it is extremely useful in trying to *understand* such systems and how they do, in fact, manage to perform those functions.

1.4.2 Principle 2

The preceding comments about representational decoders apply equally to transformational decoders. This should be no surprise given our prior discussion (in section 1.3) in which we noted that defining a transformation is just like defining a representation (although with different decoders). However, we did not previously emphasize the kinds of transformations that can be supported with linear decoding.

It has often been argued that nonlinear transformations are by far the most common kind of transformations found in neurobiological systems (see, e.g., Freeman 1987). This should not be surprising to engineers, as most real-world control problems require complex, nonlinear control analyses; a good contemporary example being the remote manipulator system on the international space station. This should be even less of a surprise to neuroscientists who study the subtle behavior of natural systems. As Pouget and Sejnowski (1997) note, even a relatively simple task, such as determining the head-centered coordinates of a target given retinal coordinates, requires nonlinear computation when considered fully (i.e., including the geometry of rotation in three dimensions). Thus, it is essential that we be able to account for *nonlinear* as well as linear transformations. In section 6.3 we discuss how to characterize nonlinear transformations in general. We provide a neurobiological example of a nonlinear transformation (determining the cross product) that allows us to account for a number of experimental results (see section 6.5). Thus we show that assumptions about the linearity of decoding do not limit the possible functions that can be supported by neurobiological systems.

This result will not be surprising to researchers familiar with current computational neuroscience. It has long been known that linear decoding of nonlinear 'basis functions' can be used to approximate nonlinear functions (see section 7.4). Nevertheless, our analysis sheds new light on standard approaches. Specifically, we: 1) show how observations about neural systems can determine *which* nonlinear functions can be well-approximated by those systems (section 7.3); 2) apply these results to large-scale, fully spiking networks (section 6.5); and 3) integrate these results with a characterization of neural dynamics and representation (section 8.1.3).

1.4.3 Principle 3

As noted in section 1.3, we can adapt standard control theory to be useful for modeling neurobiological systems by accounting for intrinsic neuron dynamics. There are a number of features of control theory that make it extremely useful for modeling neurobiological systems. First, the general form of control systems, captured by the state-space equations, can be used to relate to the dynamics of non-biological systems (with which engineers may be more familiar) to the dynamics of neurobiological systems. Second, the engineering community is very familiar with the state-space approach for describing the dynamic properties of physical systems, and thus has many related analytical tools for characterizing such systems. Third, modern control theory can be used to relate the dynamics of 'external' variables, like actual joint angles, to 'internal' variables, like desired joint angles. This demonstrates how one formalism can be used to span internal and external descriptions of behavior.

Adopting this perspective on neural dynamics allows us to develop a characterization of what we call a 'generic neural subsystem'. This multi-level, quantitative characterization of neural systems serves to unify our discussions of neural representation, transformation, and dynamics (section 8.1.3).

Given our previous discussion regarding the importance of nonlinear computations, a focus on standard control theory, which deals mainly with linear dynamics, may seem unwarranted. However, contemporary nonlinear control theory, which may prove more valuable in the long run, depends critically on our current understanding of linear systems. Thus, showing how linear control theory relates to neurobiological systems has the effect of showing how nonlinear control theory relates to neurobiological systems as well. In fact, many of the examples we provide are of nonlinear dynamic systems (see sections 6.5 and 8.2).

1.4.4 Addendum

There are numerous sources of noise in any physical system, and neurobiological systems are no exception (see section 2.2.1). As a result, and despite recent contentions to the contrary (van Gelder and Port 1995), neural systems can be understood as essentially finite (Eliasmith 2001). This is important, though not surprising, because it means that information theory is applicable to analyzing such systems. This ubiquity of noise also suggests that knowing the limits of neural processing is important for understanding that processing. For instance, we would not expect neurons to transmit information at a rate of 10 or 20 bits per spike if the usual sources of noise limited the signal-to-noise ratio to 10:1, because that would waste valuable resources. Instead, we would expect information transmission rates of about 3 bits per spike given that signal-to-noise ratio as is found in many neurobiological systems (see section 4.4.2).

These kinds of limits prove to be very useful for determining how *good* we can expect a system's performance to be, and for constraining hypotheses about what a particular neural system is *for*. For example, if we choose to model a system with about 100 neurons (such as the horizontal neural integrator in the goldfish), and we know that the variance of the noise in the system is about 10%, we can expect a root-mean-squared (RMS) error of about 2% in that system's representation (see section 2.3). Conversely, we might know the errors typically observed in a system's behavior and the nature of the relevant signals, and use this knowledge to guide hypotheses about which subsystems are involved in which functions. Either way, information regarding implementational constraints, like noise, can help us learn something new about the system in which we are interested.

1.5 Methodology

As noted previously, our central goal is to provide a general *framework* for constructing neurobiological simulations. We take this framework to consist of two parts: the guiding *principles* just outlined; and a *methodology* for applying those principles, which we describe next. So our interests are practical as well as theoretical. To this end, we have written a software package in MatLab® that can be used with this methodology (and applies the principles). Nearly all of the models discussed in the remainder of the book have been implemented with this package. The package, examples, documentation, and some extras are available at `http://compneuro.uwaterloo.ca`.

We present the methodology in three stages: system description; design specification; and implementation.

1.5.1 System description

The main goal of this first step is to describe the neural system of interest in such a way that the principles outlined in section 1.4 are directly applicable to it. In particular, available neuroanatomical data and any current functional understanding should be used to describe the architecture, function, and representations of the neural system. This description should include at least:

1. basic interconnectivity between subsystems (i.e., what is connected to what);
2. neuron response functions (i.e., distributions of neuron parameters evident in the relevant populations);
3. neuron tuning curves (i.e., distributions of encoding functions of the relevant populations);
4. subsystem functional relations; and
5. overall system behavior.

Of course, if this information was easily available, we might have little reason to construct a model. As a result, many of these details will probably be hypotheses. The point is to make explicit the assumptions that inform the model. Then, any differences between model function and the function of the modeled system may be traced to these assumptions.

Importantly, the last two functional descriptions (4. and 5.) need to be expressed in mathematical terms. In particular, the relevant neural representations need to be specified in such a way that they can be used to write explicit transformations that perform the specified functions. The goal here is to translate the functional description provided in neurobiological terms into a description in mathematical terms. So, for example, we might describe a particular neural subsystem as acting as working memory for spatial locations. To provide a mathematical description, we must identify a represented variable (e.g., x), units (e.g., degrees from midline), and a description of the dynamics that captures the notion of memory (e.g., $\dot{x}(t) = 0$, i.e., x stays the same over time; see section 8.3). In this case, there is a natural correspondence between time derivatives equaling zero and memory, which we have exploited. In most cases, the translation is much more involved. If this subsystem is part of a larger neural system that we are interested in simulating, then a similar procedure must be carried out for each subsystem. Control theory-like diagrams can be very useful for performing this decomposition.

Essentially, this step requires a rigorous formulation of hypotheses we may have regarding the function of the system we are interested in. Admittedly, this mathematical description may be highly abstract (e.g., describing a swimming eel as instantiating a kind of sine function; see section 8.5). But that is acceptable so long as the description is complete. It is not necessary to hypothesize about the functioning of every individual neuron or neuronal group, so long as we have a hypothesis about the overall behavior. In fact, our framework is intended to be a means of determining what the likely role of neurons or groups of neurons *is* at these 'lower' levels. Any differences between the resulting simulation and known neurobiological properties of these lower levels (e.g., connectivity, tuning curves, etc.) should help improve the higher-level hypothesis. Given the improved higher-level formulation, predicted neural properties can again be examined, and so on. Thus, this methodology supports a ‘bootstrapping’ approach to understanding neural systems.

In sum, the main purposes of this step are to: 1) identify the relevant neurobiological constraints; 2) identify the represented system variables; and 3) rigorously relate those variables to one another in such a way that the observed behavior results.

1.5.2 Design specification

The purpose of the design specification is to further delineate the real-world limitations that are known, or assumed to be present, for the neural system. As a result of the addendum, we must be explicit about the operating conditions (e.g., noise) that the system is subject to. This second step in the methodology explicitly demands stating the implementational constraints. So, given the representation for each variable as determined in the system description, the dynamic range, precision, and signal-to-noise ratio for each degree of freedom of those variables must be specified. Similarly, the temporal and dynamic characteristics (e.g., bandwidth, power spectrum, stable regimes, etc.) must be described. Ideally, these specifications would be made on the basis of available data. However, they may also be parameters to be manipulated during the implementation stage of the methodology. Adopting this route allows us to ask questions like: How much noise could such a system tolerate? How good does the encoding of individual neurons have to be under various noise conditions? What is the minimum allowable bandwidth for the system to function properly? and so on.

Although the precise specifications may not seem important, they can significantly affect the final model that is generated (see section 2.3.3). This goes to show the significance of implementational constraints for building good neurobiological models. If, for example, the signal-to-noise ratio must be extremely high for a particular variable, there will have to be many neurons dedicated to representing that variable (or fewer highly precise neurons). Conversely, if we have good estimates of the number of neurons in a particular system, we can use that information to determine possible design specifications.

In sum, the main purpose of this step is to precisely specify the implementational constraints on the model for each represented variable identified in the previous step.

1.5.3 Implementation

The third step of the methodology involves generating and running the model itself. Given the system description and design specification, this step combines them to determine the appropriate decoding rules, and hence synaptic weights, needed to implement the desired behavior.

Because the original system description may be framed in terms of high-level neural representations, it is often possible to simulate some parts of the model at the level of those representations (i.e., without simulating every individual neuron's function) while simulating other parts of the model with more realistic spiking neurons. As well, the amount of detail in the model of individual neurons (e.g., a rate model, spiking model, or conductance model) can vary from one part of the model to another. The computational savings of these variations in detail can be significant for large-scale models. In many cases, large-scale models could not be simulated on available hardware without this kind of control over the amount of detail incorporated into various parts of the model.

In general, the purpose of the implementation step is to run numerical experiments on the model. These experiments may take the form of simply changing the input to the network, or they might involve changing system properties defined in the design specification. The implementation stage thus supports performing an in-depth analysis of the model's behavior (e.g., stability analysis, sensitivity to noise, etc.). The results of such experiments and analyses can be used to inform revisions to either of the two previous steps. In the end, the results of such numerical experiments often suggest neurobiological experiments to pursue in the system being modeled (see, e.g., 6.5 and 8.3).

In sum, the main purpose of the final step of the methodology is to apply the principles of neural engineering outlined previously to embed the system description into a plausible neural model, and to analyze and experiment with the resulting simulation.

1.5.4 Discussion

The three main steps of the methodology and their objectives are summarized in table 1.1. These steps provide a 'recipe' for generating simulations of neurobiological systems. However, applying these steps to real systems is seldom a straightforward task. Although we have presented the methodology as consecutive steps, it is often necessary in practice to iterate over these steps (i.e., 'bootstrap' from an initial guess to a final model). Often, the reason for such interplay between steps is the preponderance of gaps in our knowledge about the system we are modeling. Of course, one of the greatest benefits of a good simulation can be determining precisely where those gaps lie.

There are a number of ways in which detailed simulations constructed using this methodology can help fill these gaps. For one, such simulations can both fine-tune and test hypotheses about neural function. More importantly, they can help predict properties of systems based on partial information. For example, if we think a given system performs some function, we can use this methodology to make predictions about what distributions of neurons there should be and what kinds of dynamic properties they should have (see section 6.5). In addition, constructing models using control theory makes it possible to build in top-down constraints (e.g., stability) observed in the real system, giving insight into how those constraints might be met in neurobiological systems (see section 8.5). Notably, the effects of bottom-up constraints (e.g., cellular properties) can be studied at the same

Table 1.1
Summary of the methodology for generating neurobiological models.

Step 1	*System description*
	- Identify the relevant neurobiological properties (e.g., tuning curves, connectivity, etc.).
	- Specify the representations as variables (e.g., scalars, vectors, functions, etc.).
	- Provide a functional description including specification of subsystems and overall system architecture.
	- Provide a mathematical description of system function.
Step 2	*Design specification*
	- Specify the range, precision, and signal-to-noise ratio for each variable.
	- Specify the temporal and dynamic characteristics for each variable.
Step 3	*Implementation*
	- Determine the decoding rules for implementing the specified transformations.
	- Determine which parts of the model are to be simulated to which degrees of detail.
	- Perform numerical experiments using resulting simulation.

time, by varying the parameters of the single neuron model being used. As a result, this approach can serve to unify the often antagonistic top-down and bottom-up perspectives on how to best understand neurobiological systems.

The challenges that arise in applying this methodology can vary significantly from system to system. This will become apparent as we explore the numerous examples in the remainder of the book. The variation stems from many sources: different systems are of varying degrees of complexity; different systems have distinct dynamics and are thus more or less sensitive to different implementational constraints; and, perhaps most importantly, there are unequal amounts of neuroscientific detail about different systems. Nevertheless, the principles we have outlined in section 1.4 and the methodology we have outlined here prove to be useful tools for generating biologically plausible, yet computationally tractable models.

1.6 A POSSIBLE THEORY OF NEUROBIOLOGICAL SYSTEMS

To this point, we have presented what we have called a ‘framework’ that consists of a set of three principles and a corresponding methodology. We have chosen these terms because they are neutral regarding the scientific status of this approach. Nevertheless, in this section we explore the possibility that the three principles can be properly called a theory of neurobiological systems. Note that the practical utility of the framework itself is independent of whether this claim is found convincing.

Recently, a consensus has begun to develop about the state of theories in neuroscience; there aren’t any. Or at least there aren’t any good ones (Churchland and Sejnowski 1992;

Marder et al. 1997; Stevens 1994; Crick and Koch 1998; see also Stevens 2000 or many of the other 'Viewpoints' in the same issue). Neuroscience is, in other words, "data rich, but theory poor" (Churchland and Sejnowski 1992, p. 16). We believe that the three principles identified in section 1.4, to continue the metaphor, can put a few coins into theory's purse. Although the diversity of neurobiological systems is generally thought to be antithetical to building theories, in the case of nervous systems a theory that does not account for the true complexities of the system is likely to be inadequate. Thus, as we have been emphasizing, we are interested in presenting a theory that respects the neurobiological data in all its diversity. In this way, we hope to engage both the theoretically inclined and the empirically inclined at the same time.

But why do we think that the three principles we have identified might constitute a *theory*? Perhaps the easiest way to see why these principles constitute a theory is to compare them to a paradigmatic theory. Consider Newton's laws of motion. Newton's laws constitute a theory of motion for a number of reasons. First, they *unify* what was, until Newton, considered two distinct kinds of motion; celestial and sublunar. Second, Newton's laws are constructed to be consistent with the known data about the motion of objects, gathered largely by Galileo. Hence his laws are *data-driven*. Third, Newton's laws allow for the prediction of planetary orbits, changes in tides, and the flattening of the earth at the poles. The fact that such predictions are a result of the theory make it *useful*. In addition to these main conceptual features, his laws are also *quantitative*. This allows his theory to be applied to a wide variety of phenomena in a rigorous manner.

The three principles outlined earlier share each of these features. First, these principles unify population and temporal representation, showing that both can be understood as non-linear encoding and linear decoding. As well, the principles unify neural representations, transformations, and dynamics in a coherent theoretical structure. Second, the principles are consistent with much of the data gathered from neurobiological systems, as their success at modeling a variety of systems demonstrates. The methodology outlines how such data can directly inform the models. Third, these principles support specific predictions regarding the properties of systems that are modeled. For instance, in section 6.5.4, we discuss predictions regarding receptor distribution on populations of neurons in the vestibular nucleus, as well as the distribution of neuron tuning curves in those populations. As well, more general predictions regarding which kinds of transformations are best supported by neuron populations with particular tuning curves are discussed in section 7.3. Lastly, although we have not presented the quantitative versions of these principles here, they can be found in section 8.1.4.

One interesting aspect of Newton's theory is that it is also, strictly speaking, false. Nevertheless, it is still a *theory* and it is still very useful for understanding and predicting (approximately) a wide variety of phenomena. Given the supreme complexity of neurobiological systems, we would not be surprised if the same was true of this theory (in fact we would be surprised if it was not). However, these principles, which are the result of a synthesis of recent work in neuroscience, is consistent with our current knowledge regarding these systems. And, as such, are likely to be very useful for understanding and predicting a wide variety of neurobiological phenomena. As well, given the clear need for a theory in neuroscience, even a theory that we expect to be disproved in the long run can play a valuable role. It can, for instance, help unify explanations across diverse systems, help to better organize existing data, facilitate experimental design, and help demonstrate where theoretical efforts should be focused.

In conclusion, we take it that the these three principles, like Newton's three laws, constitute a theory because they are *useful*, *unified*, *data-driven*, and *quantitative*. Nevertheless, we do not think that this theory can explain all there will be to explain about neurobiological function. However, we do think that it is *good* theory, as demonstrated by the numerous useful, often state-of-the-art models it has generated; and we also think it is a good *theory*.

This page intentionally left blank

I REPRESENTATION

This page intentionally left blank

2 Representation in populations of neurons

By measuring physical properties like velocity, acceleration, position, and so on, physicists can explain and predict much of our ever-changing universe. The predictive success of modern-day physics is singularly impressive. But, we would be wrong to think that prediction is a competence of physicists alone. Many non-human organisms are also highly successful in predicting changes in their environment. Toads catch flies, foxes catch rabbits, and monkeys catch tree branches—all because they represent where their target *will* be. Of course, in order to figure out where the target will be, it is necessary to represent the *current* physical properties of the target (e.g., where the target currently is, what its velocity currently is, etc.). In fact, the way physicists make predictions in physics, and the way monkeys know how to catch a swaying branch share this fundamental feature. Both, that is, represent the current state of physical properties in order to predict the future state of physical properties. Notably, both monkeys and physicists need much more than just these measurements (both, for example, need 'theories' that transform measurements into predictions; see part II), but both *need* to represent physical facts about the environment in order to make predictions at all.

As theoreticians interested in understanding the complex behaviors of nervous systems, we need to know how these physical properties can be represented by neurons. One very useful characteristic of the way physicists and engineers deal with physical properties is that they break them down into two parts: a scalar magnitude and a unit. Scalar magnitudes are values that determine the position of the property on a scale, which is set by the unit. This distinction is made because there are many mathematically describable relations that hold between physical properties (e.g., Newton's second law, $\mathbf{F} = m\mathbf{a}$). Notably, all of these relations hold regardless of which units are chosen (e.g., imperial units or metric units). From our perspective, the arbitrariness of units provides for the possibility of a general theory of neural representation. In other words, if we characterize how magnitudes are encoded by neural activities, regardless of units, then we have characterized the generally shared aspect of physical properties using the generally shared aspect of neural encodings. Thus, understanding how scalar magnitudes are encoded into neural activities is an important first step towards a general theory of neural representation (see section 2.1).

Presumably, once we understand how a scalar can be represented, it is just a matter of mathematical generalization to combine such scalars into sets of scalars, allowing us to understand the representation of more complex objects like vectors, vector fields, and arbitrary functions of these. Of course, things turn out to be a rather more complicated that 'just' mathematical generalization. Nevertheless, the motivation is a sound one: it turns out that once we know how neurons can be understood as representing scalars, we are well on our way to understanding more general forms of neural representation (see section 2.4).

2.1 Representing scalar magnitudes

2.1.1 Engineered representation

If we suppose that the theory of signals and systems applies to nervous systems, we must also suppose that there are *some* similarities between natural and engineered systems. Indeed, as noted in section 1.2, there are many. Most obviously, both neurobiological and engineered encodings of physical quantities often use the combined resources of many simple encoders. Computers use transistors and biological systems use neurons. Additionally, both computers and neurobiological systems are *real physical systems*, which means that they have a limited capacity and operate in noisy environments (see section 2.2). Finally, both kinds of system encode all physical magnitudes into some common language: for computers it is the binary language of high and low voltages; for biological systems it is (generally) the language of neural spikes. These three similarities suggest that carefully characterizing engineered representation will give us some insight into how to usefully characterize biological representation. Thus, we begin by examining the construction of representations in digital computers, and identify some analogous processes in nervous systems.

Digital computers are perhaps the best-known and most widespread example of systems that use engineered representations of scalar values. As with nervous systems, many of the representations used in computers are generated from continuous physical magnitudes in the external world. One such example is the digitization of sound. In this case, compression waves generated by vibrations in a medium are transduced into a time dependent voltage (e.g., by the diaphragm and magnet of a microphone). This voltage is then passed to an analog-to-digital converter and encoded as a series high ('on') and low ('off') voltages for use by a computer. At any moment in time, certain sets of these high and low voltages represent the magnitude of the original input voltage, and thus the vibrations of the original sound.

Let us consider the representation of the scalar-valued voltage signal in the binary language of computers. Consider, then, a standard type of analog-to-digital converter (ADC) called a 'flash' or 'parallel' converter, which is diagrammed in figure 2.1. This kind of ADC is extremely fast, although in practice it is limited to low resolution applications because of the expense of the large number of components needed. To generate an N-bit representation of an input signal, the flash ADC compares the unknown input voltage to a known reference voltage through a large number ($2^N - 1$) of comparators (C_n in figure 2.1).

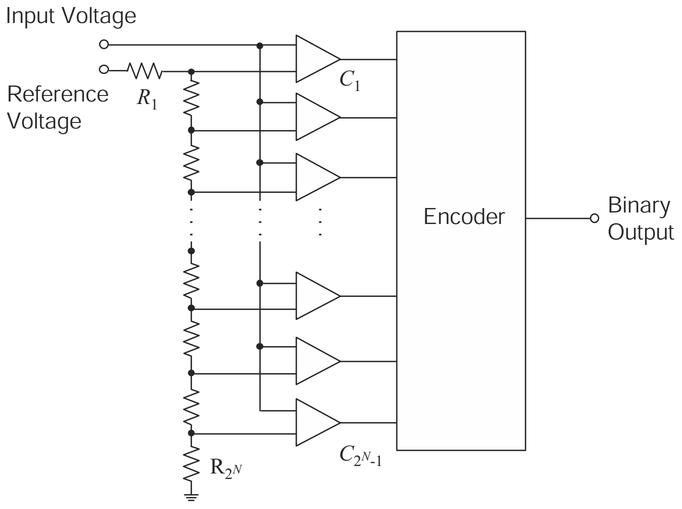


Figure 2.1
Schematic layout of a flash analog-to-digital converter. There are 2^N resistors, R, and $2^N - 1$ comparators, C which generate a binary 'thermometer code' of the incoming continuous voltage.

The reference voltage is systematically decreased with a series of 2^N resistors, R_n, so that the reference voltage at successive comparators is slightly lower. A comparator registers a high voltage ('1') if the reference voltage is higher than the input voltage, and a low voltage ('0') otherwise. In this way, all of the comparators whose reference voltage is less than the input voltage output a '1'. For this reason, the comparators in flash ADCs are often said to use a "thermometer code" because increasing the input voltage causes a comparable rise in the number of comparators that are 'on', just as increasing the temperature causes a rise in mercury in a thermometer. In order to be useful to a digital computer, the thermometer code is converted in to a binary code by an encoder.

In some sense, there are two encoding steps in this example. In the first step, the temperature code is generated by converting one magnitude (i.e., the voltage) into another (i.e., the number of comparators that are 'on'). In the second step, a more useful (i.e., compact) binary representation is generated by the encoder. Analogously, in a peripheral neuron the dendritic tree first converts one magnitude (e.g., pressure, temperature, etc.) into another (i.e., voltage changes at the soma). Then, the more useful *spike* representation is generated by the soma.[1] In the ADC, the encoder generates the final binary representation using a

1 Spikes are 'more useful' because they are extremely important for transmitting information over long distances in biological systems. Note also that in some cases (e.g., mammalian retina) the conversion to spikes may not occur until after the signal has passed through a number of soma.

nonlinear transformation of the comparator results. In the neuron, the soma generates the final spike representation also using a nonlinear transformation of the voltage coming from the dendrites.

In both cases, the representation relation we are interested in is that between the input to the 'device' and its output. We can, at least for the purposes of understanding the representation relation, ignore what happens in between. So, in both cases, we can express the input as a magnitude of some physical property (e.g., 1.3 V, 1.3 °C), and the output as a 'standardized' code. Also, in both cases, there is a particular range of the physical magnitude that the device can actually represent.[2] So, in most cases we cannot address issues of representation independently of characterizing the effective range (and precision) of the representation (see section 2.2).

Note that in order to characterize the range, we have to specify the units of the physical magnitude. However, as mentioned earlier, we can *begin* to understand representation without worrying about the particular units. So, in the case of an ADC, we treat the input signal as a dimensionless integer value between, say, 0 and 15 (represented by the temperature code) that we are trying to encode using a binary code. Of course, these are simplifications that *must* be rescinded (as we do in sections 2.2 and, for example, 2.3).

We can now begin to characterize representation in the ADC. From section 1.2 we know that both the encoding and decoding must be specified. In the case of the relationship between our chosen integer range and a binary code, these procedures are quite simple. In particular, we can identify the following two procedures: an encoder

$$a_i(x) = \begin{cases} 1, & \text{if } x \bmod 2^i > 2^{i-1} \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

and a decoder

$$\hat{x} = \sum_{i=1}^{N} a_i(x)\phi_i, \tag{2.2}$$

where

$$\phi_i = 2^{i-1}.$$

In (2.1) $a_i(x)$ is the result of the encoding, where i is the bit number and x is the number of 'on' units in the temperature code. This equation thus tells us what binary digit should be at which bit number in order to encode the integer value that is equal to the number of 'on' units. In the second equation, (2.2), N is the number of bits in our

2 Engineers often control the range of an input to an ADC (e.g., with an amplifier or limiter) in order to use available parts. This can be seen as analogous to the use of perceptual aids, like microscopes or telescopes.

representation, and ϕ_i is the integer that has the value of an 'on' bit at bit number i. Thus, ϕ_i embodies a *rule* for estimating the original value of x (i.e., $\hat{x}$), given a set of bit values, $a_i(x)$, and their positions, i, in a binary word. For this reason, the ϕ_i are called the *decoding* weights; i.e., the weights that can be used to decode the encoded representation, $a_i(x)$. Notably, $\hat{x}$ can be found by a *linear* combination of the relevant encoded coefficients and decoding weights. This is true *despite the highly nonlinear encoding process* in (2.1). In other words, linear decoding is used to successfully estimate a magnitude that was encoded with a nonlinear process. The same, we will see, is true for neurobiological representation. This fact underlies the first principle of the neural engineering discussed in section 1.4.

For the ADC, equations (2.1) and (2.2) express the relation between an integer magnitude at the input and its representation as distributed across a series of 'on' or 'off' transistors (so it is a *distributed* representation). Despite the fact that each transistor only carries a '1' or a '0', once we know the encoding and decoding relations we can understand the *set* of them as representing an integer value that none of them could represent on its own. Although these equations are not in a general form, they give a first sense of how we can represent scalar magnitudes using a *population* of encoders. The challenge now is to discover how this kind of characterization can be successfully applied to neurobiological systems.

2.1.2 Biological representation

In the case of neurobiological representation, the population of encoders is comprised of neurons rather than transistors. Neuroscientists have clearly demonstrated that individual neurons respond selectively to various stimuli including velocities, colors, motion, sound intensities, pressure, and so on. These neurons can be thought of as encoding the relevant physical property into the alphabet of neural activity. To reiterate our example: just as transistors in an ADC encode the thermometer code into a binary representation, so too neurons in biological systems encode scalar magnitudes into a neural activity representation. Again, what we need to know in order to characterize this latter representational relation are the encoding and decoding procedures. Because we assume that the decoding procedure is linear (as in the ADC case), equation (2.2) again applies. So, we just need to know what the $a_i(x)$ and ϕ_i are in order to define the representation relation.

First, in order to understand the encoding procedure that gives rise to the relevant $a_i(x)$, we need to look to neuroscience. In section 1.2.1, we noted that neuron spike trains encode current changes in the neuron's soma due to dendritic input. Stereotypical neuron response functions (firing rate versus soma current) are shown in figure 2.2. Notably, in this case (and for the rest of this chapter) we have characterized the neurobiological system as encoding physical magnitudes into neural firing *rates* (rather than with individual spikes). This is a simplification that is removed in chapter 4.

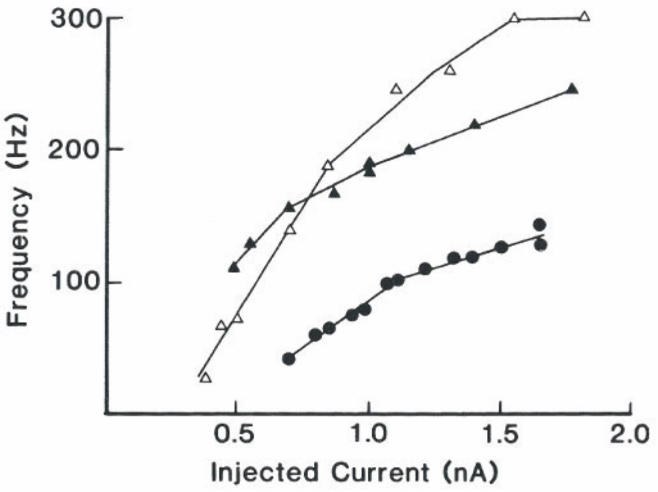


Figure 2.2
Three stereotypical neuron response functions from human cortical regular spiking cells. (From McCormick et al. 1985 © The American Physiological Society, reproduced with permission.)

Functions of the kind shown in figure 2.2 are called neuron *response functions*. They tell us how neural activity relates to soma currents. But, what we *need* to know, of course, is how that activity relates to real-world physical magnitudes, not soma currents. In fact, neurophysiologists are in the business of discovering just that kind of relation; known as a neuron's *tuning curve*. The tuning curve of a neuron is typically found by presenting the system that the neuron is in with a series of systematically varied stimuli, and recording the neuron's response. The neurophysiologist can then construct a graph (like that shown in 2.3) of the relation between the relevant physical magnitude and the neuron's firing rate.

The tuning curve is thus the result of two processes. The first is an extremely complex process that includes all processing of input signals and spikes up to the soma of the neuron of interest. This process generates a particular current at the soma. The second process, which has been well-characterized by neuroscientists, results in the generation of voltage spikes given this soma current. It is this second process that is described by the neuron's response function. *Together*, these two processes determine the neuron's tuning curve.

In order to keep these two processes clearly distinct, we write the first as $J(x)$, and the second as $G\left[J(x)\right]$. Thus, a general expression for the neural encoding process is

$$a(x) = G\left[J(x)\right].$$

The soma current, $J(x)$, actually results from combining two distinguishable currents. One is a 'bias' or 'background' current that is the result of intrinsic processes in the neuron, and/or constant input current from the rest of the nervous system. This current is responsible for the background firing rates of neurons (together, the background firing rates determine the origin of the represented magnitude, x). We call this current J^{bias}. The other current contributing to $J(x)$ we call the 'driving' current, $J^d(x)$. As its name suggests, this second current is that aspect of the soma current that drives the neuron's behavior from its background state. This is the current that is generated by connected neurons whose responses vary as the result of variations in the stimulus.[3]

Let us assume that the driving current is directly related to the relevant physical magnitude, x, as

$$J^d(x) = \alpha x.$$

The parameter α serves two functions. One is to convert the units of the physical variable to current units. The other is to scale the magnitude of x such that the tuning curve observed by neurophysiologists, $a(x)$, is properly reproduced given the response function $G[\cdot]$.

Finally, the response function, $G[\cdot]$, is determined by the intrinsic properties of the neuron. These include the resistances, capacitances, absolute refractory period, etc. that are typically used to model single neuron behavior. Notably, this framework does not depend on the response function being determined by any particular set of intrinsic properties. As we show in chapter 4, various neural models, each with their own characterization of $G[\cdot]$, can be used to model single neuron behavior.

Bringing these considerations together, the encoding process of a given neuron can be written mathematically as

$$a(x) = G[J(x)],$$

where

$$J(x) = J^d(x) + J^{bias}$$

and

$$J^d(x) = \alpha x.$$

3 Throughout the book, we assume that input currents are what drive neural firing. We realize that, biophysically, it is conductances that are varied by neural input. This adds an additional nonlinearity that we do not explicitly include in our models, but only serves to change $G[\cdot]$ in an inessential way. Cases in which this difference might matter (e.g., when inhibitory and excitatory inputs have different nonlinear affects on conductance) can be handled by the framework, as shown by section 6.4.

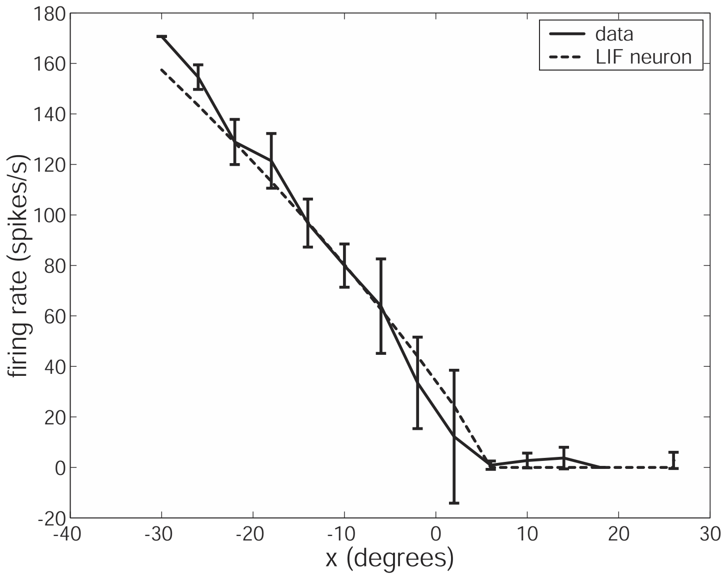


Figure 2.3
A typical neuron tuning curve in the nuclei prepositus hypoglossi that codes for horizontal eye position. The grey line indicates a leaky integrate-and-fire (LIF) neuron approximation to the actual tuning curve (black line). This is a *tuning curve* (as opposed to a response function) because it is a function of some variable, x, not just current entering the soma. (Special thanks to Kathleen E. Cullen for this data).

This, then, completes a preliminary characterization of the encoding half of the neurobiological representation relation. In order to characterize the decoding half, let us look at a simple example. Consider, then, the relation between eye position and neuron activity in the nuclei prepositus hypoglossi (NPH) and the rostral medial vestibular nucleus of the brainstem. These nuclei participate in gaze control by integrating a velocity input to determine a desired eye displacement. More specifically, neurons in these areas are responsible for representing and controlling horizontal eye position (see Robinson 1989 for a review). A typical relation between the firing rate, $a_i(x)$, of neuron i and horizontal eye position, x, in this population of neurons is shown in figure 2.3. Recall that this is a neuron tuning curve.

As shown in this figure, a convenient and often used approximation to this kind of tuning curve can be generated using the leaky integrate-and-fire (LIF) neuron model (see section 4.1 for a derivation and further discussion). In this case, we can write our encoder as

$$\begin{aligned} a_i(x) &= G_i\left[J_i(x)\right] \\ &= \begin{cases} \frac{1}{\tau_i^{ref} - \tau_i^{RC} \ln\left(1 - \frac{J_i^{threshold}}{J_i(x)}\right)}, & \text{if } \alpha_i x + J_i^{bias} > J_i^{threshold} \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{2.3}$$

where

$$J_i(x) = \alpha_i x + J_i^{bias}.$$

The response function, $G_i\left[\cdot\right]$, is determined by the intrinsic properties of the particular neuron, i. These properties include the absolute refractory period, τ^{ref}, the RC time constant, τ^{RC}, and the current threshold, $J^{threshold}$. The precise form of this equation is not important for present purposes. However, because we adopt the LIF neuron throughout our discussion, it is useful to introduce at this point. In any case, (2.3) defines the process that encodes eye positions into neural firing rates just as (2.1) defined the process that encoded decimal values into binary ones.

As noted, the representation generated by the ADC is a distributed representation that requires many transistors to precisely encode scalar values. Similarly, neurobiological systems need many neurons to precisely encode scalar values. In humans, horizontal eye position is encoded by approximately a thousand neurons in the brainstem. In order to accurately encode scalars over the range of possible eye positions, different neurons in this population encode different parts of the full range with different degrees of sensitivity. A limited sample of LIF tuning curves for such a population is shown in figure 2.4 (these have been normalized to the range $[-1, 1]$ for simplicity).

We now have a reasonably complete characterization of the encoding process that this neurobiological system uses to represent horizontal eye positions. However, our purpose in considering this example, is to better understand how to determine the relevant decoders, ϕ_i.

Because we know what we want our population to represent (i.e., a range of eye positions), and because we have assumed that the decoding process is linear, we can *determine* what the optimal decoders are. In order to do this, let us form an expression for the error that can be minimized to determine the values of ϕ_i:

$$\begin{aligned} E &= \frac{1}{2}\int_{-1}^{1}\left[x - \sum_{i=1}^{N} a_i(x)\phi_i\right]^2 dx \\ &= \left\langle \left[x - \sum_{i=1}^{N} a_i(x)\phi_i\right]^2 \right\rangle_x , \end{aligned} \tag{2.4}$$

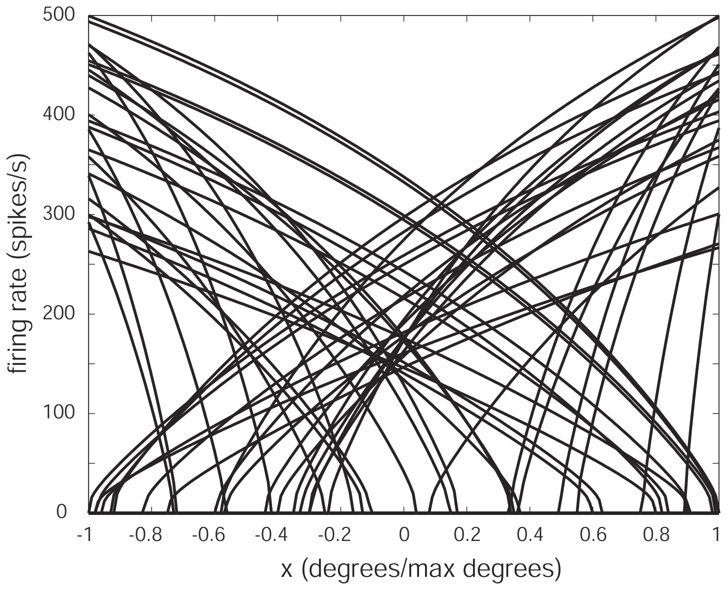


Figure 2.4
Sample tuning curves from the neural integrator population as approximated by LIF neurons.

where $\langle\cdot\rangle_x$ indicates the integral (i.e., average) over x.[4] Equation (2.4) is the average error (i.e., the square of the difference between our representation and the actual value, x) when the scalar, x, takes on values between -1 and 1. Minimizing this 'mean square error' and thus finding the (mean square) optimal ϕ_i ensures that our representation of x encoded with $a_i(x)$ can be well-decoded over the relevant interval (i.e., [-1, 1] in this case). Solving (2.4) for the ϕ_i, where each ϕ_i is an element of the vector $\boldsymbol{\phi}$, gives (see appendix A.1)

$$\boldsymbol{\phi} = \boldsymbol{\Gamma}^{-1}\boldsymbol{\Upsilon}, \tag{2.5}$$

where

$$\Gamma_{ij} = \langle a_i(x)a_j(x)\rangle_x ,$$

4 More generally, $\langle f(x)\rangle_x = \frac{\int f(x)w(x)\,dx}{\int w(x)\,dx}$ where $w(x)$ is a weighting factor. For instance, if we let $w(x) = \frac{1}{x^2+a^2}$, then the precision of the resulting representation varies as the distance from the origin. This reflects the well-known Weber-Fechner law from psychophysics. In all of the examples we present, there is no weighting function unless otherwise specified.

and

$$\Upsilon_j = \langle a_j(x)x \rangle_x .$$

This completes our characterization of the representation of a scalar magnitude in the neural integrator. That is, we have identified the encoding and decoding relations between horizontal eye positions and neural activity of the relevant neurons in the brainstem. Mathematically, we can express this representation with two equations, the encoding,

$$a_i(x) = G_i \left[\alpha_i x + J^{bias} \right]$$

and the decoding,

$$\hat{x} = \sum_i a_i(x)\phi_i.$$

Of course, there is nothing about these equations that depends on this particular example. These equations, then, are an initial characterization of how to represent scalar magnitudes using neural activity in general. Clearly, we have made a number of unrealistic assumptions (e.g., no noise, encoding with rates, etc.) but eventually we show how each of these assumptions can be removed. As simple as this characterization is, it is a significant beginning.

Before relaxing these assumptions, it is worthwhile discussing the similarities and differences between this kind of neural representation and engineered representation. First the differences. For one, the encoding process in digital computers results in either a high or low voltage state, whereas the encoding process in this biological example results in a real-valued analog quantity (the firing rate). For another, the decoding process in digital computers is well-known beforehand, whereas we had to determine a reasonable decoding process on the basis of the encoders and our assumptions about the kind of decoding that was expected. As mentioned earlier, this second point highlights the centrality of principle 1, presented in the last chapter (see section 1.4). Like most assumptions, this one will be justified if adopting it leads to a comprehensive and powerful characterization of neural representation; a claim that needs to be adjudicated in the context of the whole framework. A third difference between the biological and engineered representations is that the biological representation is far *more* redundant than the engineered representation. If a single *transistor* fails to operate correctly, the digital representation could be 'off' by as much as the most significant bit. However, if a single *neuron* fails, the biological representation will only be slightly less precise, because there are numerous other neurons encoding the value in a similar way. The neurobiological representation is, in this sense, 'more' distributed than the human engineered one. The trade-off, of course, is that far more neurons than transistors are needed to encode a value with a particular degree of precision over a given range of values.

There are also a number of notable similarities between these two kinds of representation. First, in both cases, the encoders are all similar (although non-identical). Thus, in both cases, there is something like a standard functional unit that is used multiple times to encode the representation. Second, both representations are, in the standard sense, distributed. That is, both representations are spread across more than one encoder. Third, in both cases, increasing the number of encoders increases the precision of the representation, although clearly to different degrees.[5]

Taken together, these similarities and differences both justify and limit the application of standard engineering approaches to understanding representation in the realm of neurobiology. Simply put, the similarities justify adopting a similar characterization of engineered and neural representations (e.g., linear decoding of nonlinear encoding). However, as we warned in chapter 1, the differences between these two domains should make us wary of blindly applying the engineering characterization. If, for example, biological representations are 'overcomplete' (i.e., highly redundant) whereas engineered representations are 'complete' (i.e., not redundant), then we cannot simply apply mathematical tools that successfully describe the latter to describing the former. That being said, we nevertheless have a fruitful place to begin our explorations of the nature of neurobiological representation. Let us begin, then, by relaxing some of our assumptions.

2.2 Noise and precision

2.2.1 Noisy neurons

In order to properly characterize the representations in neurobiological systems, we have to know how precise neurobiological representations actually are. Typically, engineers and computer scientists interested in artificial neural networks assume that their 'neurons' output double floating point precision real numbers every clock cycle ($\approx 10^{10}$ bits/s). However, Rieke et al. (1997) and others have shown that *real* neurons generally have only about 1–3 bits of precision per spike in their output signal ($\approx 3^{10}$ bits/s; see section 4.4.2 for more detail). When building any physical system, the components we use have some intrinsic limitations on their performance. If the components are very precise, we need only a few to represent a signal precisely. If the components are imprecise, we need many of them to represent that signal with the same precision. Neurons, compared to many engineered devices, are imprecise. In other words, they are noisy components.

If there is any expectation of noise or uncertainty in a signal being transmitted from one neuron to the next, then the information capacity of that 'channel' drops dramatically.

5 As well, the dynamic range of the digital representation scales as 2^N whereas that of the neural representation scales as $\sqrt{N}$ (see section 2.3.3).

Consider the simple analogy of sending a message down a telephone wire using voltage spikes. If the wire introduces even a small amount of jitter to each spike, then the distance between spikes at their destination will be slightly different from the distance between spikes as they were sent. It would therefore be unwise and, as a matter of fact, impossible to rely on the *exact* distance between spikes to transmit messages. In other words, if the transmitted spikes are randomly and slightly moved, and we attempt to decode a message based on the exact distance between spikes, we would be guaranteed to decode a different message than the one that was encoded. In fact, we would be randomly misled about the message that was originally encoded. In the real world, of course, telephone wires have exactly these problems—whether it be because of interfering electrical fields, tiny quantum effects, or imprecisions in manufacturing and materials—noise is ubiquitous.

Nature is subject to the same problems. In fact, a neuron's axon is a lot like a noisy telephone wire. Lass and Abeles (1975), for example, have found that mylenated axons introduce a few microseconds of jitter over a length of about 10 cm. A few microseconds of jitter is not much noise, but it is a start, and there are many other sources of noise in neurons. For one, presynaptic neurons have been found to be rather unreliable in their release of vesicles into the synaptic cleft given the presence of an action potential in the presynaptic axon (Stevens and Wang 1994). For another, the amount of neurotransmitter in each vesicle can vary unsystematically (Henneman and Mendell 1981). Of course, these sources of noise affect information transmission in the nervous system just as electrical noise affects information transmission in telephone wires. This might give rise to the worry that successfully passing messages with neurons won't ever work. Despite these sources of noise, neurons have been shown to respond similarly to similar signals with the right statistics (i.e., natural statistics; Bair and Koch 1995; Mainen and Sejnowksi 1995). So neurons, though noisy, are able to support a reasonable amount of information transmission. In other words, they are somewhat unpredictable, but not entirely unreliable.

So, we now know two important empirical facts about neurons: 1) they are noisy devices; and 2) they have the ability to transmit a signal of interest. This tells us that the code used by neurons is a robust one under natural conditions, as we would expect.[6] Notice that parallel concerns about noise do not appear to arise for engineered representations. However, appearances can be deceiving. In fact, the large differences between 'high' (+5 V) and 'low' (-5 V) voltages are chosen precisely because they make the effects of noise negligible (Kang and Leblebici 1996; Douglas et al. 1995). There is, however, a high cost paid by eliminating noise in this fashion: computational density becomes orders of magnitude lower and power consumption becomes significantly higher (Hammerstrom

6 Rieke et al. (1997) note that the coding efficiency of neurons is within a factor of two of the physical limits set by the statistics of the spike train. Thus, there is no coding scheme that could use such spike trains to encode more than about twice as much information regarding the stimulus signal.

1995; Mead 1990). Because energy consumption is a serious constraint on biological evolution, nature has (evidently) not been willing to pay this price. It has even been suggested that the neurobiological strategy for obtaining precision in the face of noise underlies its "superior performance" in the real world (Douglas et al. 1995, p. 256). In any case, neurons have not been designed to simply avoid the affects of noise the way transistors have been. So, unlike characterizations of engineered representations, characterizations of biological representations *must* explicitly address the effects of noise. This realization is the basis for the addendum to the principles of neural engineering discussed in section 1.4.

2.2.2 Biological representation and noise

The discussion in section 2.2.1 shows how various sources of noise can effect the transmission of a signal from a neuron to its neighbor. Although the sources of noise are diverse, ranging from axon jitter to vesicle release properties, they all have the effect of introducing uncertainty into any signal sent by the 'transmitting' neuron. A simple way to account for this uncertainty is to lump these sources of noise together by introducing a single noise term, η. This term should be understood as an amalgamation of the uncertainty introduced by each of the various processes.

Using this noise term, we can write the 'transmitted' firing rate as $a_i(x) + \eta_i$. That is, the firing rate that a 'receiving' neuron effectively sees will be the actual firing rate, $a_i(x)$, plus some random variation introduced into that neurons activity by the noise sources, η_i. The estimate of x, $\hat{x}$, for the receiving neuron then becomes

$$\hat{x} = \sum_{i=1}^{N} \left(a_i(x) + \eta_i\right) \phi_i. \tag{2.6}$$

To find the decoding weights, ϕ_i, we again construct and minimize the mean square error as we did in (2.4). In this case, we must average over both the range of values of x *and* the expected noise, η:

$$\begin{aligned} E &= \frac{1}{2}\left\langle \left[x - \sum_{i=1}^{N} \left(a_i(x) + \eta_i\right)\phi_i\right]^2 \right\rangle_{x,\eta} \\ &= \frac{1}{2}\left\langle \left[x - \left(\sum_{i=1}^{N} a_i(x)\phi_i - \sum_{i=1}^{N} \eta_i\phi_i\right)\right]^2 \right\rangle_{x,\eta}. \end{aligned} \tag{2.7}$$

In order to proceed further, we make standard assumptions about the kind of noise present in biological systems. In particular, we assume that the noise is Gaussian, independent, has a mean of zero, and has the same variance for each neuron.[7] Solving (2.7) then results in an expression much like (2.4) with the addition of a noise term:

$$E = \frac{1}{2}\left\langle \left[x - \sum_{i=1}^{N} a_i(x)\phi_i\right]^2 \right\rangle_x + \sum_{i,j=1}^{N} \phi_i\phi_j \left\langle \eta_i\eta_j \right\rangle_\eta . \tag{2.8}$$

Because the noise is independent on each neuron, the noise averages out *except* when $i = j$. So, the average of the $\eta_i\eta_j$ noise is equal to the variance, σ^2, of the noise on the neurons. Thus, the error with noise becomes

$$E = \frac{1}{2}\left\langle \left[x - \sum_{i=1}^{N} a_i(x)\phi_i\right]^2 \right\rangle_x + \sigma^2 \sum_{i=1}^{N} \phi_i^2. \tag{2.9}$$

We refer to the first squared term in this expression as the *error due to static distortion* (since it does not depend on the dynamics of x), and the second term as the *error due to noise*.

We can now find the optimal decoding weights, ϕ_i, by minimizing the energy as we did to find (2.5):

$$\boldsymbol{\phi} = \boldsymbol{\Gamma}^{-1}\boldsymbol{\Upsilon}, \tag{2.10}$$

where

$$\Gamma_{ij} = \left\langle a_i(x)a_j(x)\right\rangle_x + \sigma^2\delta_{ij},$$

and

$$\Upsilon_j = \left\langle a_j(x)x\right\rangle_x ,$$

where δ_{ij} is an $N \times N$ matrix with ones on the diagonal (i.e., a Kronecker delta function matrix).[8]

Equation (2.10) is thus an expression for determining the decoding weights that minimize the mean square decoding error under the given noise conditions (characterized by σ^2). This equation incorporates four important factors for describing neural representation:

7 There is some biological support for this assumption. For example, Rieke et al. (1997) note that in the case of frog calls, "the distribution of effective noise amplitudes is nearly Gaussian" (p. 184).

8 One practical consequence of including noise in our characterization of neural representation is that the matrix $\boldsymbol{\Gamma}$ is guaranteed to be invertible despite our use of a highly overcomplete representation. Previously in (2.5), $\boldsymbol{\Gamma}$ would generally be singular because when using large numbers (i.e., >40) of neurons, there are likely to be two with very similar response functions (making two of the rows in $\boldsymbol{\Gamma}$ equal). This means that $\boldsymbol{\Gamma}$ would not be invertible. Although (pseudo-) inversion can be accomplished by singular value decomposition or any similar procedure, direct inversion is computationally more efficient. We provide a detailed analysis of $\boldsymbol{\Gamma}$ and discuss related issues in section 7.4.

1) the amount of noise effecting each neuron; 2) the range over which the neurons represent the scalar (i.e., [-1,1]); 3) the precision with which the neurons represent the scalar (i.e., $1/E$); and 4) the number of neurons that participate in the representation, N. Notably, these factors are not independent since specifying some of them will determine what the others are. For example, if we know the number of neurons, the range of sensitivity, and the precision of the representation, we can determine what the (Gaussian) noise profile of the neurons must look like. Similarly, if we know how noisy the neurons are, the range, and the precision of the representation, we can determine the minimum number of neurons there must be participating in the representation.

Equation (2.10) is thus a result of taking seriously implementational constraints on neurobiological representation. By including the nonlinearities and noise that are intrinsic to the components that implement the representation, we get a theoretically useful yet practical characterization of neurobiological representation. Of course, these kinds of physical constraints directly determine measurable properties of nervous systems. Thus, taking implementation seriously in our theoretical characterization of representation provides a means of connecting this framework directly to empirical work. In the next section, we provide one example of how our framework and current empirical results can together provide a useful picture of the representations used in a simple neural system.

2.3 An example: Horizontal eye position

With this characterization of neurobiological representation in hand, we are in a position to consider a specific neurobiological example—representing eye position. We have already introduced some aspects of the representation in this neural system. In this section we provide more detail and take the opportunity to describe how to apply the methodology we introduced in chapter 1 (section 1.5). However, we are not yet in a position to use all aspects of the methodology because we have yet to introduce a means of characterizing transformations.[9]

2.3.1 System description

As mentioned in section 1.5.1, the purpose of the system description step is both to gather the relevant neuroscientific data and to express our resulting understanding of the system in mathematical terms. We have already done much of this work in section 2.1.2.

First let us reiterate and expand the relevant neurobiological considerations. Recall that two nuclei in the brainstem, the nuclei prepositus hypoglossi (NPH) and the rostral

9 We more fully characterize this same system in sections 5.3, where we introduce spiking neurons, and 8.2, where we include the relevant transformations and input signals.

medial vestibular nucleus (VN), play a central role in controlling horizontal eye position (see Robinson 1989 or Moschovakis 1997 for a review). Together, these populations are often called the *neural integrator* because they convert a velocity signal into a displacement signal. The neurons in these populations project to motor neurons in the abducens and oculomotor nucleus, which then project to the two major muscles that control horizontal eye movement, the lateral and medial recti (Seung 1996). Disturbances to the abducens and oculomotor neurons do not permanently affect eye positions, whereas disturbances to the NPH-VN do. For this reason, the NPH-VN neurons are thought to be the locus of a stable eye displacement signal (Seung 1996).[10]

Like most vestibulo-ocular neurons, the neurons in this population are thought to be linear over a wide range (Moschovakis 1997; Arnold and Robinson 1991). For this reason, the response functions of these neurons are often modeled as rectified lines (i.e., a straight line but whose negative values are set to zero). However, using nonlinear, conductance-based models of single neurons results in more biologically plausible models (Lee et al. 1997). We again use LIF neurons, whose tuning curves closely approximate the response of conductance models, and whose general form is expressed by (2.3).[11]

Single unit recordings have shown that, over the population, the majority of position sensitive cells are active when gaze is centered (Delgado-Garcia et al. 1989), with background rates between about 0–150 spikes/s (Moschovakis 1997). These neurons have also been found to have sensitivities between about 0.1 and 7 spikes/s per degree (Fukushima et al. 1992). A small set of tuning curves based on (2.3) with this kind of distribution of parameters is shown in figure 2.5. The most objectionable assumption we have made about the neural population displayed in this figure is that all of the firing rates are the same at maximal eye deflections. However, this assumption simplifies subsequent analysis and, more importantly, we later show that it does not effect the results (see section 5.3).

Note that there are two distinct subpopulations depicted here; those with rightward increasing responses ('on' neurons) and those with leftward increasing responses ('off' neurons). As well, the majority of neurons have a background firing rate when the eyes are straight ahead (i.e., at $x = 0$). These are both features of the NPH-VN network seen in monkeys (Moschovakis 1997).

To summarize, we take the variable x to range over horizontal eye positions and to be encoded in a population of neurons in the NPH-VN using response functions of the form of (2.3) and with parameters distributed as depicted in figure 2.5.

10 Also see Fukushima et al. (1992) for an in-depth argument based on anatomical and physiological data that NPH is well-suited to act as a neural integrator.

11 Note that we are here concerned only with the displacement related responses of these neurons. Many tend to have velocity related responses as well (Moschovakis 1997).

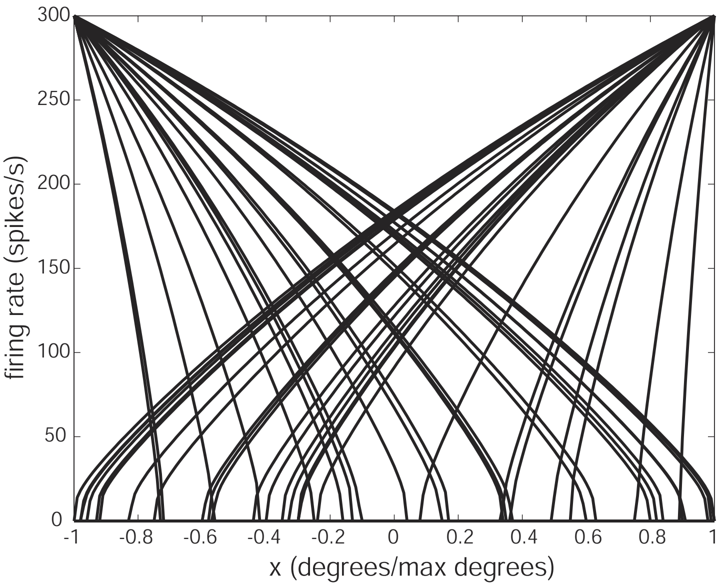


Figure 2.5
A sample of 50 nonlinear LIF tuning curves for a population of neurons used to encode horizontal eye position. The neurons are distributed such that the majority of cells are active when the eyes are centered (i.e., at $x = 0$).

2.3.2 Design specification

As described in section 1.5.2, design specification addresses issues of the precision, noise, and range of the relevant representations. The range in this case is straightforward. Humans, for example, have approximately a ± 50 degree range of horizontal motion (Davson 1990, p. 657). In this example, we normalize this range to the interval between -1 (leftmost position) and 1 (rightmost position).

The amount of noise we expect in the neural representation can be derived from recent results regarding the transmission of signals by individual neurons (Rieke et al. 1997; see section 4.4 for more detail). As mentioned in section 2.2.1, neurons carry about 3 bits of information per spike. This means that the code allows approximately 10% error. Thus, we assume that our neurons can encode with a signal-to-noise ratio of 10:1 (i.e., 10% error; $\sigma^2 = 0.1$ assuming an even distribution of possible signals). As before, we assume independent, Gaussian noise with a mean of zero for each neuron.

As for the precision of the representation, we leave this unspecified and consider it the purpose of our model to determine the relation between the precision of the representation and the number of neurons, given the specifications we have made in this section.

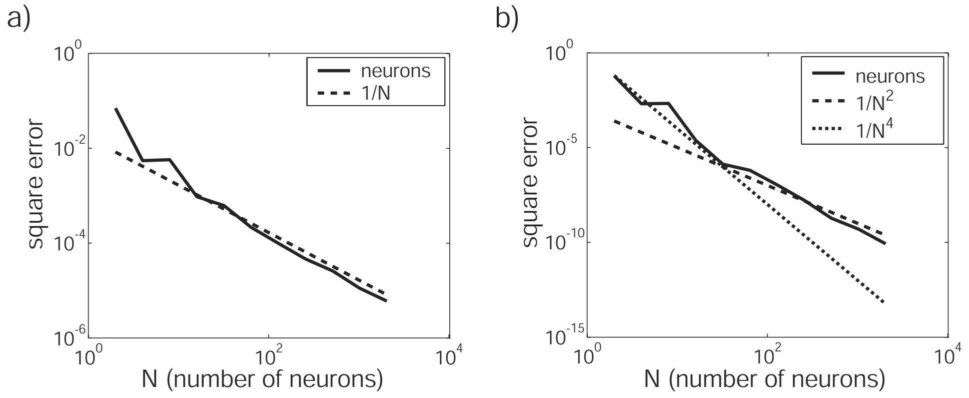


Figure 2.6
Simulation results showing the relation between the precision of the representation and the number of neurons in the representing population. a) Decreasing error due to noise as a function of the number of neurons. We added randomly generated LIF neurons to build up the populations, assuming parameters distributed as for figure 2.5. For comparison, the line $1/N$ is also plotted. b) Decreasing error due to static distortion for the same population. For comparison, the lines $1/N^2$ and $1/N^4$ are also plotted.

2.3.3 Implementation

Much of the implementational work for this example has already been completed in section 2.2.2. To briefly review, the expression for the estimate of eye position, $\hat{x}$, is a linear combination of decoding weights ϕ_i and neural encodings, $a_i(x)$, as shown in equation (2.6). To find the decoding weights under noise, we derived (2.9) which can be minimized, resulting in decodings weights given by (2.10). The system description and design specification we provided in the two preceding sections specify both $a_i(x)$, and σ^2. Therefore, we can directly solve (2.10).

With this characterization of neural representation, we can perform computational experiments to determine the relation between the precision of the representation and the number of neurons in the population, N. As can be seen in figure 2.6a, the mean square error due to noise of the representation decreases as $1/N$.

This result agrees with previous results using maximum likelihood estimation (see, e.g., Snippe and Koenderink 1992 and Paradiso 1988). Because maximum likelihood provides a theoretical bound on optimal representation, these results show that this kind of representation is a good one.[12] This is not surprising given the close relation between our approach and maximum likelihood methods (Salinas and Abbott 1994). In fact, we can directly show that this sort of performance is expected. Note that all of the tuning curves in

12 This has been discussed in greater detail elsewhere (Seung and Sompolinsky 1993).

our model have the same firing rate, r, at $x = 1$ or -1; i.e., $a_i(1) = r$ for all 'on' neurons, i, and zero otherwise. Therefore,

$$\begin{aligned} 1 &= \sum_{i=1}^{N} a_i(1)\phi_i \\ 1 &= r\frac{N}{2}\phi_i \\ \phi_i &= \frac{2}{rN}. \end{aligned}$$

Of course, our method finds optimal ϕ_i over all possible positions, but this argument shows that we should expect $\phi_i \propto 1/N$, since both r and 2 remain constant for any N. Furthermore, from equation (2.9) we know that the noise term is proportional to $\sum_{i=1}^{N} \phi_i^2 \propto N/N^2 = 1/N$. Thus, the square error due to noise (see 2.2.2) decreases as approximately $1/N$.

Figure 2.6b demonstrates numerically that the other source of error, the error due to static distortion, decreases as $1/N^2$. In fact, the error due to static distortion goes as $1/N^4$ for large N if there is no noise in the system (not shown). However, these results are mainly of academic interest since once the number of neurons increases beyond around 10 in number, the error due to noise dominates the error due to static distortion. Given this analysis, it is clear that we can achieve a root-mean-square precision of around 1% with approximately 100 neurons.

2.3.4 Discussion

This example gives a first demonstration of why it is useful to characterize neurobiological systems as representing. In this particular case, such a characterization provides us with specific hypotheses regarding how well a neural population can encode information about some external state. It also lets us know how that encoding changes if neurons are removed or damaged. Furthermore, this representational analysis gives insight into how a population of neurons can be thought of as working together to more effectively process signals. That is, it gives us a theoretical foundation for understanding the relation between the functioning of individual neurons and the characterization of neural systems as processing 'higher-level' variables like eye position. On occasion we call variables like eye position *higher-level representations* to contrast them with neural activity (i.e., firing rates or spikes), which we call *basic representations*. We consider the latter 'basic' because there is a general consensus that neurons are the basic units of neural function. In contrast, determining which neurons belong in a particular population (i.e., participate in a particular higher-level representation) is an open issue in most cases. However, in order to generate useful explanations of neural systems, it is often necessary to talk in terms of higher-level representations, since explanations in terms of individual neurons can quickly become overly complex. As well, being able to relate basic and higher-level representations may serve as a bridge between higher-level behavioral sciences, like psychology, and more basic behavioral sciences, like neuroscience. As discussed in section 3.1, higher-level representations can be similarly combined to produce even higher-level representations. This permits generating theories regarding behavior in terms of very high level representations that may be many steps removed from neural activity (see section 8.5). The relation between each of these levels of representation can be defined by the relevant decoding weights.

In this particular example, the decoding weights, ϕ_i, that we find define what we earlier called the representational decoding. If we think of the signal x as the input to these neurons, and the estimate of that signal, $\hat{x}$, given their firing rates, $a_i(x)$, as the output, then the decoding weights are those weights that make the input/output transfer function as linear as possible over the relevant range under the given noise profile (see figure 2.7). Equivalently, we can think of the decoding weights as being chosen to best approximate a linear function given the nonlinear response functions. That is, the weights are chosen such that when we add up all the response functions in the population, the total is (nearly) equal to the original signal, x, over its entire range. This is a natural way of understanding a *representation* because, if we think of a representation as standing-in for what it represents, then we would expect that manipulating the representation (i.e., $\hat{x}$) would be much like manipulating what it represents (i.e., x). This will only be true, of course, if the representation and what it represents are in some sense the same (or nearly the same). An input/output transfer function that was a perfectly straight line with slope of one and that intersected the origin would indicate that the representation and what it represents were identical. In this case, we have quantifiable deviations from that ideal kind of representation.

2.4 Representing vectors

Vector representations have a long history in neuroscience, and have been found in a wide variety of neural systems, including those responsible for producing saccades (Lee et al. 1988), estimating the orientation of visual input (Gilbert and Wiesel 1990), encoding an organism's spatial orientation (Zhang 1996), detecting wind direction (Miller et al. 1991), coding echo delay (O'Neill and Suga 1982), and controlling arm movement (Georgopoulos et al. 1984). Vector representations have been implicated in diverse anatomical areas including hippocampus, motor cortex, vestibular systems, frontal cortex, peripheral sensory systems, visual cortex, and cerebellum. So it is essential to have a viable characterization of vector representation.

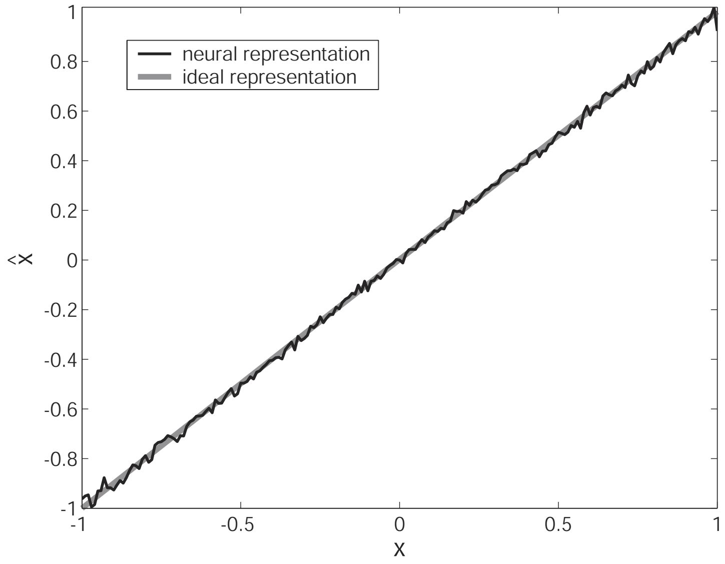


Figure 2.7
The input/output transfer functions for the neural representation in NPH and an ideal representation.

Mathematically, vectors are simply a set of scalars. Thus, it should not be surprising that our characterization of scalar representation generalizes quite naturally to vectors. However, a simple-minded mathematical generalization will not be satisfactory because we need to correctly describe the *particular implementation(s)* of vector representation found in nervous systems. This means that we cannot just construct multiple, independent representations of scalars and then combine them in order to represent vectors. Rather, we must account for the fact that neuron tuning curves themselves often show sensitivity to multiple dimensions. Similarly, a particular neuron in a population representing a vector can be sensitive to almost any direction in the vector space, not just one or a few of the cardinal directions (i.e., axes), which a simple-minded generalization might assume. Of course, using multi-dimensional tuning curves is *equivalent* to using a representation that employs representations of independent scalars. However, given our commitment to understanding neural systems in terms of the properties of neurons, it is essential to characterize neural representation with an eye to actual neural responses.

A multi-dimensional LIF tuning curve has much the same form as a single dimension tuning curve. However, the input current is determined in a different manner. Specifically, we can write

$$\begin{aligned} a_i(\mathbf{x}) &= G_i\left[J_i(\mathbf{x})\right] \\ a_i(\mathbf{x}) &= \frac{1}{\tau_i^{ref} - \tau_i^{RC}\ln\left(1 - \frac{J_i^{threshold}}{J_i(\mathbf{x})}\right)}, \end{aligned} \qquad (2.11)$$

where

$$J_i(\mathbf{x}) = \alpha_i \left\langle \tilde{\boldsymbol{\phi}}_i \mathbf{x} \right\rangle_n + J_i^{bias}.$$

For a neuron sensitive to two different dimensions, the function $J_i(\mathbf{x})$ defines a plane whose maximal slope is along $\tilde{\boldsymbol{\phi}}_i$. This plane is then subjected to the LIF nonlinearity, $G_i[\cdot]$, as before. The resulting 2-D tuning curve is shown in figure 2.8a.[13]

The vector $\tilde{\boldsymbol{\phi}}_i$ is called the *preferred direction vector* of a cell with this kind of response. This is because, for a given vector magnitude, the cell responds most strongly to inputs whose direction aligns with the preferred direction vector (see figure 2.8b). So, the vector is 'preferred' because it causes the highest firing rate for a given magnitude and it is a 'direction' because it has a particular orientation in some (possibly high-dimensional) vector space.

Although $\tilde{\boldsymbol{\phi}}_i$ seems to be a parameter that was not included in the scalar case, equation (2.11) can be thought of as a D-dimensional version of the response in equation (2.3). Previously, when we were discussing scalar representation in section 2.1, the $\tilde{\phi}_i$ parameter was implicitly 1 for 'on' neurons or -1 for 'off' neurons, resulting in a positive or negative slope respectively. In the one-dimensional scalar problem there are only two possible preferred directions (positive and negative). However, in the higher-dimensional problem, there are infinitely many possible preferred directions. This is why it becomes necessary to explicitly include $\tilde{\boldsymbol{\phi}}_i$ in our formulation.[14] Because $\tilde{\boldsymbol{\phi}}_i$ centrally defines the encoding procedure, we also call it the *encoding vector*.

Having determined the encoding, we again turn to determining the relevant decoding. Our expression for the decoding of a vector, $\mathbf{x}$, is much like that for a scalar in equation (2.2):[15]

$$\hat{\mathbf{x}} = \sum_{i=1}^{N} a_i(\mathbf{x})\boldsymbol{\phi}_i. \qquad (2.12)$$

13 For notational convenience, we write the vector dot product between two D-dimensional vectors, $\mathbf{x}$ and $\mathbf{y}$ as $\langle \mathbf{xy} \rangle_n$ which is to indicate the sum over the elements of the vectors, i.e., $\sum_{n=1}^{D} x[n]y[n]$ which defines the dot product. This is consistent with our previous usage of $\langle \cdot \rangle$, as the dot product can be thought of as the discrete version of the integral. Using the brackets in both of these contexts simplifies notation.

14 Note also that the preferred direction vector is always normalized to be a unit vector.

15 Indexing the elements of the vectors explicitly, this is equivalent to $\hat{x}[n] = \sum_i a_i(\mathbf{x})\phi_i[n]$.

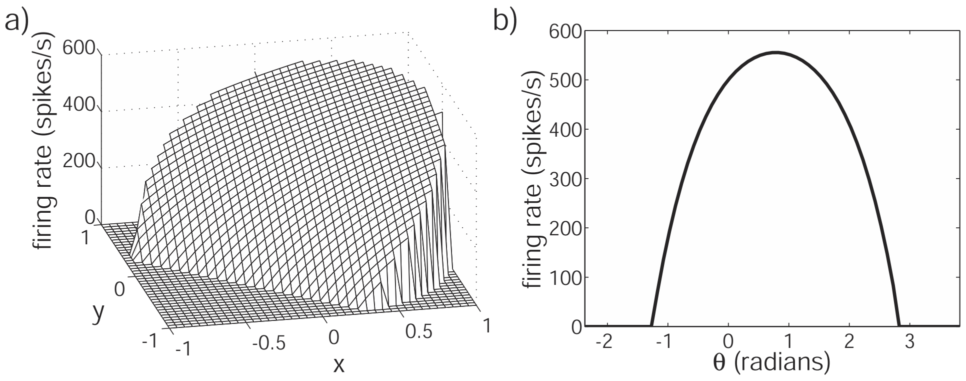


Figure 2.8
Tuning curves for a LIF neuron encoding a 2-dimensional input vector. a) The normalized tuning curve given by equation (2.11). b) The preferred direction is at an angle of $\pi/4$ radians, where the maximal firing rate is achieved. This is the more commonly plotted 'cosine tuning curve' for the same neuron as in a). Note that this second tuning curve only describes sensitivity to direction, not magnitude.

In fact, the expressions for the error and optimal weights are much like (2.4) and (2.5) respectively, and the procedure for including noise in the representation is analogous to that discussed in section 2.2. As a result, we do not need to derive any new methods for finding the decoders for vectors, they can be found in the same way as the decoders for scalars. We have thus defined a general form for vector representation consisting of the encoding,

$$a_i(\mathbf{x}) = G_i \left[\alpha_i \left\langle \tilde{\boldsymbol{\phi}}_i \mathbf{x} \right\rangle_n + J_i^{bias} \right] \tag{2.13}$$

and the decoding,

$$\hat{\mathbf{x}} = \sum_{i=1}^{N} a_i(\mathbf{x}) \boldsymbol{\phi}_i. \tag{2.14}$$

This kind of vector representation has been extensively compared to other kinds of vector representation, as we discuss in the next section.

2.5 An example: Arm movements

One of the best known examples of population coding in higher mammals is the representation of arm movements in motor cortex. Largely due to the efforts of Apostolos Georgopoulos and his colleagues (see Georgopoulos et al. 1984; Georgopoulos et al. 1986;

Georgopoulos et al. 1989; Georgopoulos et al. 1993), it has been shown that populations of broadly tuned cells in monkey motor cortex represent arm movements using a weighted vector code. The particular weighted vector code that Georgopoulos and his collaborators have used to demonstrate this is what they call a 'population vector'. However, to distinguish this kind of population vector decoding from the optimal linear decoding we are considering, we call it a 'preferred direction' decoding. This kind of code is much like (2.12) except that the decoding vector is assumed to be the same as the encoding (i.e., preferred direction) vector; i.e., $\boldsymbol{\phi}_i = \tilde{\boldsymbol{\phi}}_i$, so under this characterization

$$\hat{\mathbf{x}} = \sum_{i=1}^{N} a_i(\mathbf{x})\tilde{\boldsymbol{\phi}}_i. \tag{2.15}$$

Because this approach to understanding population codes has been around for so long, it has be extensively analyzed and compared to other kinds of population decoding, including the optimal linear approach we are using (see, e.g., Salinas and Abbott 1994). In this section, we reproduce some of these past results while using a more realistic neural model.[16]

2.5.1 System description

The system description in this section is taken largely from Georgopoulos et al. (1984). In this paper, Georgopoulos et al. examine two-dimensional arm movements in rhesus monkeys. They report a series of experiments in which the monkeys are cued to move in one of eight directions, either inward or outward (see figure 2.9). They find that a subset of arm-related motor cortical cells is used to produce movements in this particular task. Thus, the cells they analyze are both appendage and task specific. Of these cells, 75% are directionally tuned (i.e., they had a single preferred direction, $\tilde{\boldsymbol{\phi}}_i$). This directional tuning is independent of the destination of the arm movement. In other words, a cell tuned to '$45°$ up' has its maximal firing rate when the movement is in the top right quadrant of the outward task, and also when the movement is in the bottom left quadrant of the inward task (see figure 2.9). Georgopoulos et al. note that the directional tuning of these cells has a fairly general form. Of the tuned cells, 78% are fit with a cosine tuning curve of the form

$$a_i(\theta) = b_0 + b_1 \cdot \cos(\theta_i - \theta_0), \tag{2.16}$$

where θ_0 is angle of the preferred direction vector, θ_i is the angle of the direction of movement, and b_0 and b_1 are regression coefficients. The preferred directions, θ_0, are roughly evenly spread around the $360°$ arc.

16 In particular, the Salinas and Abbott (1994) analysis uses rectified linear neurons, while we use leaky integrate-and-fire neurons.

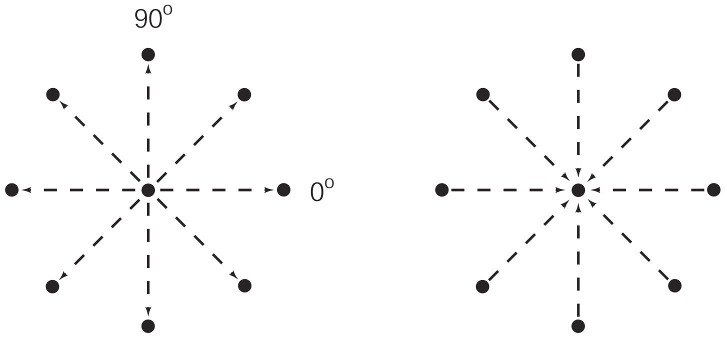


Figure 2.9
Outwards (left) and inwards (right) arm movement tasks. (Adapted from Georgopoulos et al. 1984 © Neurosciences Research Foundation, reproduced with permission.)

Notably, Georgopoulos et al. are concerned only with decoding the *direction* of arm movement, which is why the firing rates are parameterized with respect to θ only. Of course, θ, is a scalar. Thus, this equation describes the encoding of a scalar, not a vector. However, there is evidence that the primary motor area is involved in coding the magnitude, as well as the direction of the movement velocity vector (Todorov 2000; Moran and Schwartz 1999; Schwartz 1994). Thus, a more complete picture is gained by thinking of the motor cortex as encoding a two dimensional vector, $\mathbf{x}$, where

$$a_i(\mathbf{x}) = G_i \left[\alpha_i \left\langle \tilde{\phi}_i \mathbf{x} \right\rangle_n + J_i^{bias}\right]. \tag{2.17}$$

This kind of encoding captures all of the information present in (2.16); i.e., direction information. In fact, we can reconstruct the cosine tuning curves of (2.16) by sampling the possible directions of $\mathbf{x}$ (as shown in figure 2.8b). Substituting a 'sweeping' series of vectors of the same magnitude into (2.17) results in a tuning curve that can be expressed in the form of (2.16); this is a straightforward consequence of using the dot product in (2.17). However, (2.17) also captures the ability of these neurons to encode the *magnitude* of a vector (as is evident from figure 2.8a).

2.5.2 Design specification

We can rely on the same general considerations about the information transmission of neurons as we did in section 2.3.2, to justify the assumption that the variance of the noise is 0.1 of the maximum neuron firing rate. The range of possible vector directions in this case is a 360° circle. Finally, the vector magnitudes are normalized to fall in the range $[0, 1]$. As well, we again take the purpose of this discussion to be a characterization of the precision of the representation given our characterization of neural representation.

2.5.3 Implementation

For both the preferred direction and optimal linear decoding, the encoding is of the same form, described by equation (2.17). As mentioned, the decoding is also of the same form in both cases, i.e.,

$$\hat{\mathbf{x}} = \sum_{i=1}^{N} \left(a_i(\mathbf{x}) + \eta_i\right) \boldsymbol{\phi}_i, \tag{2.18}$$

although in the preferred direction decoding, $\boldsymbol{\phi}_i = \tilde{\boldsymbol{\phi}}_i$. To find the optimal decoders under noise in the vector case, we perform the squared error minimization described in section 2.2, integrating over the vector, $\mathbf{x}$.

As shown in figure 2.10, the optimal decoders perform significantly better than the preferred direction decoders for any population size. Furthermore, the rate of improvement with additional neurons is much greater for the optimal decoders than the preferred direction decoders. This means that more information can be extracted with fewer neurons and that the information from any additional neurons is better used. Note that this comparison only accounts for the ability to decode the *direction* of the encoded vector. Magnitude information, while encoded, has simply been ignored in generating this comparison. This is because the preferred direction decoding cannot extract magnitude information very well at all (Salinas and Abbott 1994).

2.5.4 Discussion

As we can see from this example, extending this approach for characterizing neural representation from scalars to vectors requires little additional theory (most notably making the encoding vectors, $\tilde{\boldsymbol{\phi}}_i$, explicit). As well, this example serves to emphasize a point that has been made by a number of researchers in the past: optimal linear decoders are very good, very simple, and neurologically plausible (Abbott 1994; Pouget et al. 1998; Pouget et al. 1993; Salinas and Abbott 1994). These decoders are very good because they are better than other linear decoders, the preferred direction decoders being just one example. They are very simple precisely because they are linear. And, they are neurologically plausible because neurons can clearly perform the linear computations necessary to support this characterization of representation.[17] As well, although we have only discussed static representation, such representations can also be used for modeling the *functions* of motor cortex. For instance, Nenadic et al. (2000) use this kind of representation in a simple model of two-dimensional arm control.

17 Nevertheless, we think neurons can perform nonlinear computations. We leave the many interesting issues that arise regarding nonlinear computation to chapter 5.

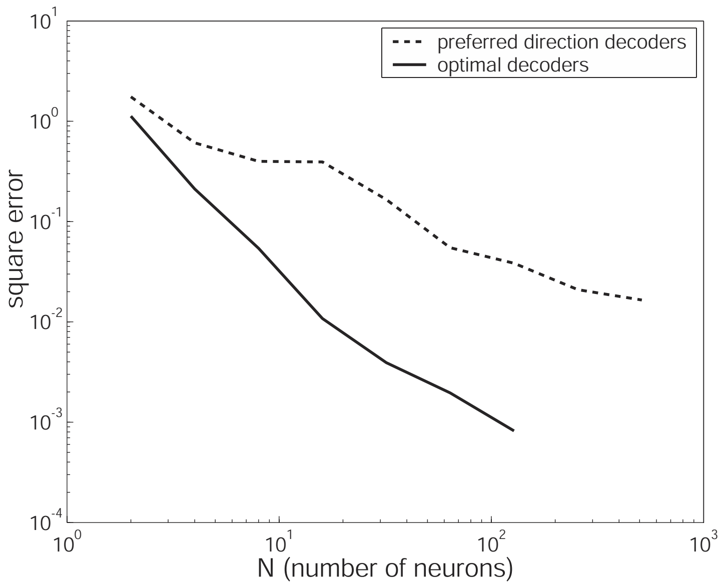


Figure 2.10
Preferred direction decoding compared to the optimal linear decoding of the direction of arm movement. The optimal linear decoding performs significantly better and improves more rapidly with additional neurons.

Finally, it is worth emphasizing that this example demonstrates that the preferred direction characterization of vector representation should be reconsidered. Notably, the population vector characterization is an intuitive and convenient means of understanding the activity of neurons as representational. However, when contrasted with this alternate characterization of neural representation it is clear that intuition can sometimes lead us astray. While the preferred direction approach has been used for many years, and has helped us gain a better understanding of neural representation, it does not have a solid theoretical foundation; choosing preferred direction vectors for decoding was a convenience. For this reason, much of the information carried by the neurons is lost using the preferred direction characterization. Neural systems generally do not seem this suboptimal when it comes to exploiting the information capacity of their components (see the many examples in Rieke et al. 1997). As a result, the optimal linear characterization, which does have a good theoretical foundation and is a natural part of a unified understanding of neural representation should be preferred. Salinas and Abbott (1994) have convincingly shown that representations like (2.12) uniquely combine accuracy and neurological plausibility. We have reproduced this result, and are engaged in showing that such representations can also be a part of a principled approach to understanding neural representation.

2.6 An example: Semicircular canals

As mentioned in section 2.5.1, the preferred direction vectors of the neurons in motor cortex are more-or-less evenly distributed in the relevant vector space. Therefore, in the arm movement representation, the encoding vectors, $\tilde{\phi}_i$, were also evenly distributed over the 360° range of the representation. However, vectors need not always be represented in this manner. Rather, this aspect of the representation depends on the particular system being modeled.

In this section, we briefly describe an alternative 2-dimensional vector representation that is more akin to the kind of representation seen in the semicircular canals of the vestibular system. Our purpose in this section is to show that the theoretical approach we have been presenting must be supplemented by experimental information regarding the particular system being modeled. Although, as we show, such information does not always make a big difference to the quality of the representation, it can make a big difference to the topology of the network, and thus to the kinds of predictions we make about the system being modeled (e.g., what computations are or are not likely to be supported).[18]

2.6.1 System description

There are three semicircular canals on each side of the head, oriented at approximately right angles to each other (see figure 6.11). Each canal is filled with a fluid (endolymph) and is end-stopped by a gelatinous membrane (the cupula). Embedded in this membrane are a large number of tiny hair cells, whose deflections cause neighboring neurons to fire action potentials. The signal carried by the canal afferent neurons is proportional to the angular velocity of the head along the direction of the canal's orientation. This signal is generated as follows: an angular acceleration of the head causes the canal to move relative to the fluid inside it (since the fluid has a moment of inertia and is free to flow). The movement of the fluid deforms the cupula, thus bending the hairs embedded in it, and causing the neurons to fire. The afferent neurons fire in direct proportion to the deformation of the cupula. The reason the neurons carry a signal about angular velocity, and not angular acceleration, is that the fluid's angular velocity very rapidly matches that of the canal after an acceleration, *but* the cupula is not very elastic. This means that the degree of deformation of the cupula by the fluid quickly tracks the angular velocity (since the applied force due to acceleration has a time constant of about 3 ms) and the cupula only slowly regains its original shape after such a deformation (with a time constant of about 10 s; for a more comprehensive explanation see Wilson and Jones 1979, pp. 43–55). So, the canals generally operate as integrating accelerometers (at least over a bandwidth corresponding to frequencies normally incurred during natural movement; Wilson and Jones 1979, p. 54).

18 The relation between topology and function is discussed in detail in chapters 5–7.

Given this description, it is natural to take the vector being represented by the semicircular canals to be angular velocity, Ω. Although this is a three dimensional vector in the vestibular system, we have restricted this example to 2-dimensions in order to compare it directly with the results of the previous section.[19] Thus, we will be implementing the angular velocity around only two of the three possible axes. Because neurons in canals on opposite sides of the head are sensitive to velocities in opposite directions, they comprise 'on' and 'off' subpopulations, like those in the eye position example.

We skip the design specification as we assume it is identical to the previous example, in order to facilitate comparison.

2.6.2 Implementation

Since this representation is equivalent to that in section 2.5, we can write the expression for this representation exactly as in equations (2.17) and (2.18). However, we can also be explicit about the fact that we only use encoders along two orthogonal axes. Thus, for the encoding we can write

$$a_{im}(\mathbf{x}) = G_{im}\left[\alpha_{im}\left\langle \mathbf{x}\tilde{\boldsymbol{\phi}}_{im}\right\rangle_n + J_{im}^{bias}\right]. \tag{2.19}$$

We have introduced an extra index m, to denote the four populations (two 'on' and two 'off') involved in this representation. This allows us to be more explicit about the nature of the encoding vectors. In particular, $\tilde{\boldsymbol{\phi}}_{im} = \mathbf{u}_m$ for $m = 1, 3$ and $\tilde{\boldsymbol{\phi}}_{im} = -\mathbf{u}_m$ for $m = 2, 4$, where the $\mathbf{u}_m$ are the unit vectors along the orthogonal axes, i.e., $\mathbf{u}_1 = [1\,0]$ and $\mathbf{u}_2 = [0\,1]$. Thus, all of the neurons have preferred directions along the principle axes. The decoding then becomes

$$\hat{\mathbf{x}} = \sum_{i,m} a_{im}(\mathbf{x})\boldsymbol{\phi}_{im}. \tag{2.20}$$

Writing the representation in this way shows that (2.20) is identical to simply combining two distinct scalar representations (i.e., the scalar representation for $m = 1, 2$ and the scalar representation for $m = 3, 4$). This is because we could treat these two parts of the sum over m independently given that the relevant $\mathbf{u}_m$ are orthogonal, and thus independent.

Comparing this kind of representation to that of the previous example (where the encoding vectors of the population are evenly distributed around the entire space being represented), we see that the quality of the representation is essentially the same (see figure 2.11). The slight differences in the errors is a result of randomly choosing neural populations of a given size. However, the slope is identical, meaning that the representations are improving at the same rate.

19 For a detailed model employing the full 3-D representation see section 6.5.

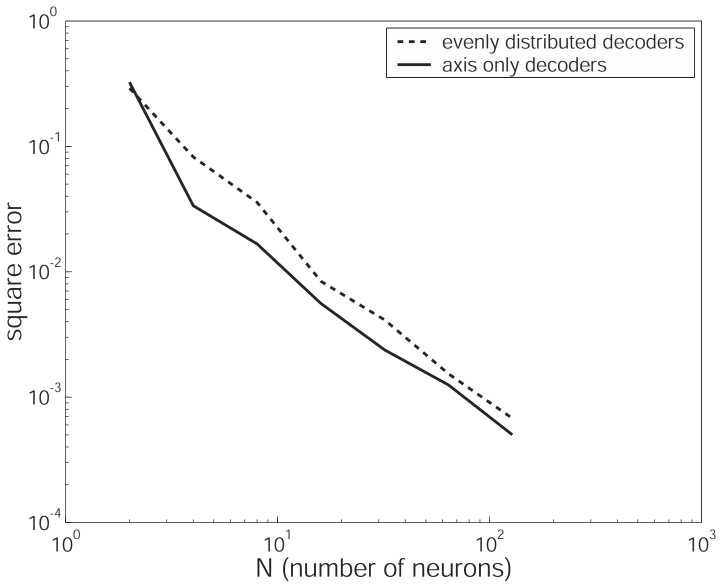


Figure 2.11
Comparison of a vector representation using preferred direction vectors distributed only along the axes (as in the semicircular canals) with a vector representation using preferred direction vectors distributed evenly throughout the vector space (as found in motor cortex). As expected, these representations are equally precise for a given population size. Results averaged over 4 runs for each value of N.

In conclusion, then, choosing the kind of neural representation to employ in a model depends on the particular neural system being modeled. However, the characterization of neural representation we have been discussing can be used to quantify the representation regardless of what particular *kind* it is. This means that the optimal linear approach can help us pick out *general properties* of neural representation (such as the relation between error and population size) that are independent of implementation, while being able to incorporate the *specific properties* of individual neural systems (such as the particular distribution of preferred direction vectors in the system).

2.7 Summary

At the start of the chapter we suggested that neurobiological representation could be understood analogously to engineered codes. We pursued this analogy to show both similarities and differences between neural representation and codes used in digital computers. The most important similarity was that both engineered and neural coding could be defined by nonlinear encoding and linear decoding.

We then noted that, in the neurobiological case, encoding was well understood, but that we needed a means of finding the decoders. As a result, we presented a method for finding the optimal linear decoders.We extended this noise-free analysis to include the effects of noise in order to make this characterization appropriate for real neurobiological systems. This completed the introduction to the main aspects of population representation that we employ throughout the remainder of the book.

We subsequently considered the specific example of the neural integrator, to demonstrate how the theory could be applied to a particular neurobiological system. In that section we showed that the error in the representation due to noise decreased in proportion to the number of neurons in the population. We also showed that the static error due to distortion decreased in proportion to the square of the number of neurons.

Having laid the foundations for understanding neural representation of scalars, we demonstrated that it could be generalized to characterize vector representation as well. We compared this characterization of vector representation to the more popular 'preferred direction' characterization, using representations in motor cortex as our example. We argued, after Salinas and Abbott (1994), that this optimal linear approach should be preferred over other approaches for its unique combination of precision and neural plausibility. We then turned to the example of the semicircular canals to show that the specific properties of the neurobiological system we model are both important and can be accounted for by the characterization we have presented.

3 Extending population representation

As mentioned in section 2.1 of the last chapter, it is reasonable to expect that understanding scalar representation provides an important first step towards understanding more complex kinds of representation. In section 2.4, we showed precisely how we could generalize scalar representation to the the case of vectors. In fact, the resulting characterization for vectors was identical to the scalar case with the exception of having to explicitly identify the encoding vectors. In section 3.1 of this chapter, we argue that this generalization is one that can be repeated for more complex kinds of representation. This leads us to the identification of a representational hierarchy.

The next step on this hierarchy after vector representation is *function* representation. So, in sections 3.2–3.4 we describe how to characterize function representation using this same approach. We also discuss the deep and important relation between function representation and vector representation in section 3.3. We then show how function representation is relevant for understanding certain neurobiological systems by considering such representations in lateral intraparietal cortex (LIP).

3.1 A REPRESENTATIONAL HIERARCHY

In section 2.3 we described how the neurons in the horizontal neural integrator could be understood as supporting a higher-level scalar representation. In sections 2.5 and 2.6, we examined two different kinds of vector representation. Table 3.1 summarizes the kinds of representation we have considered so far.

The theory employed in each of these cases does not essentially depend on the precise nature of the higher-level representation. For instance, the vestibular and motor representations could have been vectors of any dimension. And, furthermore, these two implementa-

Table 3.1
The three kinds of higher-level representation discussed so far.

Neural System	*Dimension*	*Encoder* $(a_i(x))$	*Decoder* $(\hat{x})$
Neural Integrator	1-D scalar	$G_i\left[\alpha_i x + J_i^{bias}\right]$	$\sum_i a_i(x)\phi_i$
Motor Cortex	2-D vector	$G_i\left[\alpha_i\left\langle\tilde{\boldsymbol{\phi}}_i \mathbf{x}\right\rangle_n + J_i^{bias}\right]$	$\sum_i a_i(\mathbf{x})\boldsymbol{\phi}_i$
Semicircular Canals	2-D vector	$G_{im}\left[\alpha_{im}\left\langle\mathbf{x}\tilde{\boldsymbol{\phi}}_{im}\right\rangle_n + J_{im}^{bias}\right]$	$\sum_{i,m} a_{im}(\mathbf{x})\boldsymbol{\phi}_{im}$

tions of vector representation *equivalently* represent any vector. The only reason to choose one kind of representation over the other, in such cases, is that the neurobiological evidence demands one kind of representation rather than another. There is nothing inherently better or worse about either of these two kinds of vector representation. This implies two things. First, as we have been insisting, this is clear evidence that we must turn to the neurobiological system itself to properly constrain our representational characterization. Second, our analysis of such representation does *not* depend on neurobiological detail, which is why it might serve to provide a unified understanding of representation.

Comparing, for the moment, the characterizations of vector and scalar representation, we see that the major difference is merely an increase in the number of dimensions being represented. This can be understood as the beginnings of a kind of representational hierarchy, ordered by the number of dimensions a representation can effectively characterize: scalar representations can only characterize one-dimensional problems; whereas vector representations can characterize a large but finite number of dimensions. It is natural to extrapolate this hierarchy. That is, we can allow the elements of the vectors to become continuously indexed, and hence cover an infinite dimensional space (i.e., represent a function).[1] And, we can combine finite and infinite indices to represent infinite sets of finite vectors (i.e., a vector field). We have summarized these steps in table 3.2.

Note that in each case, the 'basic' representation (i.e., neural activities) are what these representations are defined over. So, the representation *relation* is defined by the encoding and decoding processes which relate higher-level and basic representations. The higher-level representations, themselves, are ordered by increasing dimensionality to give rise to this representational hierarchy. It is important to keep in mind that there is more than one possible implementation at each representational level. So, this hierarchy in no way relieves us from being responsible to the particular neurobiological system we are interested in understanding.

Nevertheless, being able to construct such a hierarchy is useful for a number of reasons. First, the fact that all of the levels of such a hierarchy are of a standard form suggests that this characterization provides a *unified* way of understanding representation in neurobiological systems. This is why the first principle of neural engineering stated in chapter 1 describes the form of the equations shown in table 3.2. Although we do not provide detailed examples of all possible kinds of representation captured by this table, we do examine scalar, vector, and function representations.

Second, if we can construct a hierarchy in such a way that it is applicable to diverse neurobiological systems, then we have a very *general* means of characterizing representa-

1 There are important subtleties in the relation between vector and function representation that technically violate this distinction, as discussed in section 3.3. Nevertheless, categorizing kinds of neurobiological representation in this way is conceptually useful.

Table 3.2
The first levels of the representational hierarchy. Each level of this hierarchy is defined in terms of basic representations, i.e., neural activity.

Kind	*Encoder* $(a_i(x))$	*Decoder* $(\hat{x})$
Scalar (1)	$G_i\left[\alpha_i x + J_i^{bias}\right]$	$\sum_i a_i(x)\phi_i$
Vector (N)	$G_i\left[\alpha_i \left\langle \tilde{\phi}_i[n]x[n]\right\rangle_n + J_i^{bias}\right]$	$\sum_i a_i(x[n])\phi_i[n]$
Function (∞)	$G_i\left[\alpha_i \left\langle \tilde{\phi}_i(\nu)x(\nu)\right\rangle_\nu + J_i^{bias}\right]$	$\sum_i a_i(x(\nu))\phi_i(\nu)$
Vector Field $(\infty \times N)$	$G_i\left[\alpha_i \left\langle \tilde{\phi}_i(\nu,[n])x(\nu,[n])\right\rangle + J_i^{bias}\right]$	$\sum_i a_i(x(\nu,[n])\phi_i(\nu,[n])$

tion. Moreover, the fact that the approach we have outlined does not completely determine which representational characterizations are best for particular neural systems, suggests that this framework is appropriately flexible; i.e., sensitive to empirical data.

Third, being able to work at, and move between, the various levels in such a hierarchy has important practical consequences. For instance, suppose we are interested in modeling a complex neurobiological system that represents a vector field. The hierarchy makes is possible to consider the function representation and vector representation aspects of the vector field representation separately, before combining them. That is, this kind of hierarchy supports the decomposition of complex representations into simpler ones, often making problems far more computationally tractable than they might otherwise be. While it may not be immediately apparent from table 3.2 how this will work, we provide an example of this kind of decomposition in section 8.5.

To demonstrate the generality of this way of thinking about representation, let us now consider a third level of the representational hierarchy.

3.2 Function representation

There are many cases when it is appropriate to describe neural systems as representing functions. Representation of visual images, auditory streams, patterns of movements, tactile sensations, and velocity fields, are all naturally considered representations of functions. This is because, in each of these cases, neural systems must represent the more-or-less continuous variation of one parameter (e.g., light intensity) with respect to the more-or-less continuous variation of another parameter (e.g., spatial location). In fact, we suspect that function representation is one of the most common kinds of neural representation.

For a specific example, consider research on working memory that has shown a sensitivity to shape as well as position in neurons in lateral intraparietal (LIP) cortex (Sereno and Maunsell 1998). If these neurons were sensitive to only the position of a stimulus, we would perhaps want to claim that the population was representing a set of vectors, i.e., the vectors denoted by $\boldsymbol{\nu} = [\nu_1, \nu_2, \nu_3]$, covering all relevant spatial locations. However, since there are other dimensions being represented (in this case, shape) at each spatial location, we know that there will be different patterns of neural activity for different stimuli *at the same position*. Thus, the population encodes a set *functions* that can represent distinguishable shapes at every spatial location, i.e., the functions $x(\boldsymbol{\nu}; \mathbf{A}) = f_{\mathbf{A}}(\boldsymbol{\nu})$, where $\mathbf{A}$ is a set of parameters that pick out a particular function of $\boldsymbol{\nu}$. In the case of shape sensitive neurons, the parameters $\mathbf{A}$, determine the function, $f_{\mathbf{A}}$, that represents a specific shape at location $\boldsymbol{\nu}$. Different values of $\mathbf{A}$ would thus result in different functions that encode the presence of different shapes at the same location (for a more complete discussion of a similar example, see 8.3). This kind of complex representation can support many kinds of transformations that neurobiological systems take advantage of, including texture processing, object recognition, and tactile memory.

At first glance, it may seem that function representation is significantly more complicated than scalar or vector representation because of the parameter $\mathbf{A}$. However, in our previous discussion of scalar and vector representation, notice that we identify variables (e.g., x or $\mathbf{x}$) that range over a *set* of scalars or vectors. So, when discussing neural representation in terms of variables, we are actually talking about the representation of *sets* of scalars or vectors (e.g., all possible eye positions between -50 and 50 degrees, or all possible arm positions in a horizontal plane). Similarly, when discussing functions, we must talk about *sets of functions* that comprise some function space (or part of some function space). The purpose of the $\mathbf{A}$ parameter is to make this explicit, since it is not captured by standard mathematical notation (i.e., we do not have a standard way of writing a variable that ranges over functions instead of scalar magnitudes).

Perhaps the most important step in characterizing function representation, is delimiting the set of functions to be represented (this is precisely what we must do in the design specification). Not only do we need sufficient physiological data to have a sense of what functions could be represented, but we must also try to express that set of functions in a mathematically concise way. This can often be quite challenging (see, e.g., sections 8.3 and 8.5).

Nevertheless, the general expression for identifying an ensemble of functions, or a 'function space', is straightforward. For simplicity, let us consider functions of the scalar variable, ν. We begin by characterizing the valid domain of the functions,

$$x(\nu) \text{ for } \nu \in (\nu_{min}, \nu_{max}), \tag{3.1}$$

and then construct a mathematical representation of the whole ensemble:

$$x(\nu;\mathbf{A}) = \sum_m A_m \Phi_m(\nu) \text{ for } \mathbf{A} \in \rho(\mathbf{A}), \text{ and } m \in \{1,\ldots,M\}. \tag{3.2}$$

Equation (3.1) defines the domain for any given function in the set defined by (3.2).[2]

This mathematical representation resembles our previous characterization of neural representation, but only superficially (see sections 7.1 and 7.4 for further discussion). Unlike neural representation, this function space is characterized by linear combinations of an *orthonormal* basis, $\Phi_m(\nu)$ (see section 7.1 for a discussion of 'basis functions'). This, then, is like using Cartesian coordinates to represent vectors, usually not what neural systems do (a point to which we will return shortly).

A familiar example of defining an ensemble of functions in this way is given by the standard Fourier decomposition,

$$x(\nu;\mathbf{A}) = \frac{A_{1,0}}{2} + \sum_{m=1}^{M} A_{1,m}\cos(\omega_m \nu) + A_{2,m}\sin(\omega_m \nu), \tag{3.3}$$

where $\omega_m = m2\pi/\left|\nu_{max} - \nu_{min}\right|$. Here, each choice of the $2(M+1)$ parameters, $\mathbf{A}$, defines a particular function $x(\nu)$. For instance, if we choose $A_{1,1} = 1$ (and let all the other components be zero), we get a single period *cosine* wave over the range of ν. If, instead, we similarly choose $A_{2,1} = 1$, we get a single period *sine* wave over the range of ν. As is well known, using this decomposition we can choose the $\mathbf{A}$ parameters to get any function we want over this range.

However, in neural systems we seldom, if ever, want to represent *any* function. As a result, we need to define limits on the coefficients $\mathbf{A}$ in this representation. This is why equation (3.2) includes the probability distribution $\rho(\mathbf{A})$.[3] Specifying a probability distribution is a convenient means of determining which functions may *actually* be represented; i.e., of specifying the 'range' of functions. Since the set of coefficients is drawn from some probability distribution, $\rho(\mathbf{A})$, it is that distribution that defines the function space. If, in the Fourier example, we structure $\rho(\mathbf{A})$ so that values of $A_{i,m}$ for small m are likely to be small, and values of $A_{i,m}$ for large m are likely to be large, we will have an ensemble of high bandpass functions (i.e., functions composed of high frequency sines and cosines). If, instead, we structure $\rho(\mathbf{A})$ so that only one particular $\mathbf{A}$ is likely to be chosen, we will have defined a set of signals with one member.

2 It may be more accurate to write $x(\nu;\mathbf{A},\mathbf{\Phi})$ since these functions are only defined *with respect to this particular set of basis functions*. However, since the Φ_m never change once picked, it is more compact to write $x(\nu;\mathbf{A})$.

3 As usual, we assume that implementational constraints will play a role in which coefficients are used. As a result, we always pick $\mathbf{A}$ for some finite M where each A_m coefficient has a limited precision.

Once we have defined the function space in this way, we can begin to construct a neural representation of that space. The neural encoding, as before, is determined by the intrinsic properties of neurons as well as their relation to the rest of the system, as captured by an encoder. Specifically, we write

$$a_i(x(\nu; \mathbf{A})) = a_i(\mathbf{A}) = G_i[J_i(\mathbf{A})], \tag{3.4}$$

where

$$J_i(\mathbf{A}) = \alpha_i \left\langle x(\nu; \mathbf{A})\tilde{\phi}_i(\nu) \right\rangle_\nu + J_i^{bias}. \tag{3.5}$$

This expression should look familiar because of its strong resemblance to vector encoding specified by equation (2.13). There are two main differences. One is that the preferred direction, or encoding, vector has been replaced by the function $\tilde{\phi}_i(\nu)$. However, this 'encoding function' plays much the same role, as discussed below. Second, rather than a dot product, the angle brackets $\langle \cdot \rangle$ indicate integration over the parameter ν. This is because integration functions much like a dot product over a continuous space—both provide a measure of *similarity* when employed in this way. Notice that because of this integration, the ν parameter is integrated out of (3.5). As a result, the soma current J_i is only a function of the coefficients, $\mathbf{A}$. This is to be expected because the firing rates in the population a_i are always representing *some* function over ν. The *particular* function being represented is picked out uniquely by the parameters $\mathbf{A}$, so the firing rates are only a function of those parameters. This is why (3.4) includes $a_i(\mathbf{A})$, which we generally use in place of $a_i(x(\nu; \mathbf{A}))$. As a result of this parameterization, when we think of the tuning curve as a function of ν, and not of $\mathbf{A}$, we may be surprised by the result that the tuning curve looks quite different for different stimuli.

Consider the example of encoding orientation in primary visual cortex. In figure 3.1, the stimulus (i.e., encoded function) defining this tuning curve is an oriented bar. If the stimulus had been two oriented bars at right angles to one another (i.e., a different $x(\nu; \mathbf{A})$), we would expect this tuning curve to be bimodal (i.e., have its highest activity at both 0 *and* 90 degrees).[4] Thus, it would be wrong to think that we can identify the tuning curve of a cell *independently* of how we probe the cell. So it would also be wrong to think that any, arbitrary tuning curve can be used to define neural encoding. Rather, things are more complicated.

To see why, let us return to the question of how the encoding functions, $\tilde{\phi}_i(\nu)$, play a role analogous to encoding vectors. Suppose that the tuning curve of some neuron,

4 This is often what is seen in V1 cells for such cross stimuli (Jay Hegde, personal communication). Although this response is by no means the only one (Shevelev 1998; Shevelev et al. 1998), which suggests there is more work yet to do in characterizing these encoding functions.

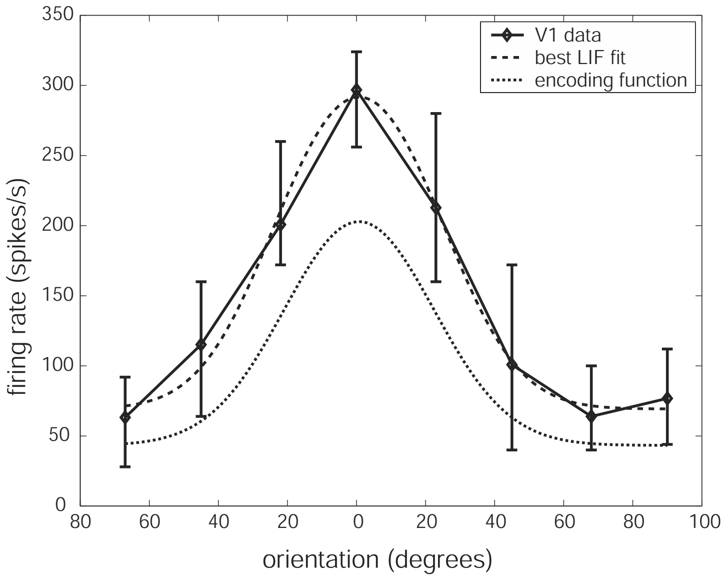


Figure 3.1
Example tuning curve for a cell from a macaque monkey in primary visual cortex (data provided by Dan Marcus). For this cell, ν ranges over orientations of bars in the receptive field. Note that the encoding function includes the gain, α_i, and bias, J_i^{bias}.

i, is approximately Gaussian in shape. This means that when the cell is systematically probed with stimuli that vary along some dimension, ν, it responds as depicted in figure 3.1. As usual, we model this response using a LIF neuron, whose response function is a monotonically increasing function of current. What we must determine, then, is how to map the dimension ν onto current, J, in such a way that the response of our model cell looks like the response in figure 3.1. That is, we must determine an encoding function that relates our currently encoded function to changes in soma current. This is precisely the role of the encoding vector in a vector representation. Furthermore, assuming that the function to be encoded, $x(\nu; \mathbf{A})$, is normalized then, if that input function matches the encoding function, the cell will respond most strongly. In this way, our encoding function becomes the cell's 'preferred' function, just as the encoding vector is the cell's 'preferred' vector.

So, tuning curves are a result of the similarity between encoding functions and an encoded function, modulated by neuron response properties. In neural systems, then, we are in the difficult position of trying to find out how to characterize the tuning curve somewhat independently of the particular encoded function. From mathematics we know that we can represent any function as a series of delta functions, $\delta(\nu)$. So, if we probe the neuron with a series of delta function inputs, $x(\nu; \mathbf{A}) = \delta(\nu - \nu_m)$, at many positions, i, we can determine how it will respond to any function composed of a series of delta functions, i.e., any function. As a result, putting the series of delta function responses that the cell generates *together* will result in a function that looks like the encoding function (modulo the effects of the intrinsic neuron response properties). In fact, probing neurons with delta functions is one way of understanding what neuroscientists are doing when they probe cells with some simple stimuli (like single oriented bars). Thus, these kinds of experiments provide good evidence of what the encoding function is in neurons. So, in the example shown in figure 3.1, the encoding function is taken to look a lot like the final neural tuning curve. But, we must always remember that how much these kinds of tuning curves resemble encoding functions depends on how much the stimuli used in the experiment resembles the appropriate delta function; and determining *that* is seldom a trivial matter.

So far we have centered our discussion on the encoding of functions. To understand representation, of course, we must also characterize the decoding. Fortunately, finding decoders for functions is very similar to finding decoders for vectors and scalars. In particular, we can find the decoding functions by minimizing the mean square error. First, let us explicitly write the linear decoding we are assuming to give our estimate of $x(\nu; \mathbf{A})$ as

$$\hat{x}(\nu; \mathbf{A}) = \sum_i a_i(\mathbf{A})\phi_i(\nu), \tag{3.6}$$

where the $\phi_i(\nu)$ are the optimal linear decoding functions we need to find. We proceed, as before, by constructing and minimizing the mean square error between our estimate and the original signal. Again, we include noise to give

$$E = \left\langle \left[x(\nu; \mathbf{A}) - \sum_i \left(a_i(\mathbf{A}) + \eta_i\right) \phi_i(\nu) \right]^2 \right\rangle_{\mathbf{A},\eta}. \tag{3.7}$$

Minimizing this error gives, as before,

$$\boldsymbol{\phi}(\nu) = \boldsymbol{\Gamma}^{-1}\boldsymbol{\Upsilon}(\nu), \tag{3.8}$$

where

$$\Gamma_{ij} = \langle a_i(\mathbf{A})a_j(\mathbf{A})\rangle_{\mathbf{A}} + \sigma_\eta^2\delta_{ij} \tag{3.9}$$

$$\Upsilon_i(\nu) = \langle x(\nu; \mathbf{A})a_i(\mathbf{A})\rangle_{\mathbf{A}}. \tag{3.10}$$

The resulting decoding functions, $\boldsymbol{\phi}(\nu)$, are thus perfectly analogous to the decoding vectors found earlier. To construct an estimate of the encoded function, we simply sum these functions weighted by the relevant neuron activity. This is somewhat like the Fourier representation discussed earlier in equation (3.3), but the neural decoders are not orthonormal. We have thus defined a general form for function representation consisting of the encoding,

$$a_i(x(\nu;\mathbf{A})) = a_i(\mathbf{A}) = G_i\left[\alpha_i\left\langle x(\nu;\mathbf{A})\tilde{\phi}_i(\nu)\right\rangle_\nu + J_i^{bias}\right] \tag{3.11}$$

and the decoding,

$$\hat{x}(\nu;\mathbf{A}) = \sum_i a_i(\mathbf{A})\phi_i(\nu). \tag{3.12}$$

On a practical note, it is often difficult to define the probability distribution $\rho(\mathbf{A})$, which is needed to find the decoding vectors. Instead, however, if we know approximately what kinds of functions are represented in the neural system of interest, we can use this knowledge to effectively characterize $\rho(\mathbf{A})$. That is, we can take the set of functions that we know to be represented in the system and then determine what vectors, $\mathbf{A}$, can be used to represent these functions. We can then minimize over that set of vectors in (3.7) rather than $\rho(\mathbf{A})$. Effectively, we are sampling some distribution, $\rho(\mathbf{A})$, and constructing a Monte Carlo estimate of it assuming each example function has an equal probability. While less than ideal, this approach is often more appropriate in a practical setting.

3.3 Function spaces and vector spaces

Before proceeding to a concrete example, it is important for us to discuss the intimate and important mathematical relation between function spaces and vector spaces. In section 3.2, we defined the set of functions to be represented by some neural population in terms of an orthonormal basis, $\Phi_m(\nu)$, as (see (3.2))

$$x(\nu;\mathbf{A}) = \sum_m^M A_m\Phi_m(\nu). \tag{3.13}$$

Because this basis is orthonormal, to find the coefficients, A_m, we can project the basis onto the function being represented, $x(\nu;\mathbf{A})$:

$$A_m = \langle x(\nu;\mathbf{A})\Phi_m(\nu)\rangle_\nu . \tag{3.14}$$

As a result, the function $x(\nu;\mathbf{A})$ is completely defined by the vector of coefficients $\mathbf{A} = [A_1, \ldots, A_M]$ and the basis $\mathbf{\Phi}(\nu)$. Thus, as we have discussed, choosing a *set* of $\mathbf{A}$ vectors defines the ensemble of functions that we are interested in. In the context of building models, choosing this set defines the possible higher-level representations that a neural system can support.

It is this same ensemble of functions that is taken to be represented at the basic level, i.e., by neurons. As mentioned in our discussion of (3.5), the soma current resulting from encoding these functions is only sensitive to the coefficients, $\mathbf{A}$. This means that we can write the soma current as $J_i(\mathbf{A})$ and the neural activity as $a_i(\mathbf{A})$. And that we can decode $a_i(\mathbf{A})$ to estimate $\mathbf{A}$ (which completely determines $x(\nu;\mathbf{A})$). In other words, we can think of a population representing the functions $x(\nu;\mathbf{A})$ as representing the vectors $\mathbf{A}$ instead.

However, we must ensure that the range of represented $\mathbf{A}$ match those that we have defined by specifying the function ensemble. That is, we do not want our estimated value of $x(\nu;\mathbf{A})$ to lie outside the ensemble defined in (3.13) and (3.14). One way to ensure this is to constrain the neural decoders, $\phi_i(\nu)$, to lie in this same space. To do this, let us represent these decoders using the *same* orthonormal basis as we used in (3.13) and (3.14), i.e., let us write

$$\phi_i(\nu) = \sum_m^M q_{im}\Phi_m(\nu). \tag{3.15}$$

Similarly, there is no point in allowing our encoding functions, $\tilde{\phi}_i(\nu)$, to encode some function $x(\nu;\mathbf{A})$ in such a way that it cannot be decoded by these constrained decoders, so we can define the encoders in the same manner, namely

$$\tilde{\phi}_i(\nu) \;=\; \sum_m^M \tilde{q}_{im}\Phi_m(\nu). \tag{3.16}$$

We can now use the relations in (3.15) and (3.16) to re-write (3.11) as follows:

$$\begin{aligned}
a_i(\mathbf{A}) &= G_i\left[\alpha_i\left\langle \sum_{n,m} A_m\Phi_m(\nu)\tilde{q}_{in}\Phi_n(\nu)\right\rangle_\nu + J_i^{bias}\right]\\
&= G_i\left[\alpha_i\left(\sum_{n,m} A_m\tilde{q}_{in}\delta_{nm}\right) + J_i^{bias}\right]\\
&= G_i\left[\alpha_i\left(\sum_{m} A_m\tilde{q}_{im}\right) + J_i^{bias}\right]\\
&= G_i\left[\alpha_i\left\langle \mathbf{A}\tilde{\mathbf{q}}_i\right\rangle_m + J_i^{bias}\right].
\end{aligned}$$

We have thus expressed the problem of *function encoding* as one of *vector encoding*, given an orthonormal basis.

Similarly, we can express the problem of *function decoding* as one of *vector decoding*. Recall from (3.8) that

$$\phi(\nu) = \mathbf{\Gamma}^{-1}\mathbf{\Upsilon}(\nu).$$

We can multiply both sides by $\mathbf{\Phi}(\nu)$ to obtain

$$\mathbf{q} = \mathbf{\Gamma}^{-1}\mathbf{\Upsilon}',$$

where

$$\Gamma_{ij} = \langle a_i(\mathbf{A})a_j(\mathbf{A})\rangle_{\mathbf{A}} + \sigma_\eta^2\delta_{ij} \tag{3.17}$$

$$\Upsilon'_{im} = \langle a_i(\mathbf{A})A_m\rangle_{\mathbf{A}}\,. \tag{3.18}$$

We now have an equivalent vector representation (with respect to $\Phi_m(\nu)$) for the function representation defined in (3.12) and (3.11), which can be summarized as an encoding

$$a_i(\mathbf{A}) = G_i\left[\alpha_i\left\langle \mathbf{A}\tilde{\mathbf{q}}_i\right\rangle_m + J_i^{bias}\right],$$

and a decoding

$$\hat{A}_m = \sum_i a_i(\mathbf{A})q_{im},$$

or

$$\hat{\mathbf{A}} = \sum_i a_i(\mathbf{A})\mathbf{q}_i.$$

So, we no longer need to explicitly talk about the variable ν when discussing the encoding and decoding of functions in the space spanned by $\Phi_m(\nu)$. Instead, we can perform all encoding and decoding in terms of the vector of coefficients, $\mathbf{A}$. However, if we want to know what *functions, $x(\nu;\mathbf{A})$,* are being encoded and decoded, we eventually have to relate the vectors, $\mathbf{A}$, back to the original function space. We do this by decoding that vector, $\mathbf{A}$, using the basis, $\Phi_m(\nu)$.

As before, in order to find $\mathbf{q} = [q_1, \ldots, q_m]$ we do not have to explicitly express $\rho(\mathbf{A})$ (i.e., the probability distribution that determines which functions are most likely to be represented). Instead, we can define some set of vectors $\mathbf{A}^*$, where each member, $\mathbf{A}_i$, is found by projecting some represented function $x_i(\nu)$ onto the $\Phi_m(\nu)$. Again, this is like performing a Monte Carlo estimate of the ideal $\rho(\mathbf{A})$. We then use this set, $\mathbf{A}^*$, to find $\mathbf{\Upsilon}'$ in (3.18) above. This effectively minimizes the encoding/decoding error of all vectors in $\mathbf{A}^*$ without regard for other vectors in the same space. This is precisely what we want, since we are most interested in having accurate representations of functions whose projections onto $\Phi_m(\nu)$ give $\mathbf{A}^*$.

We have now shown how any problem that demands function representations can be turned into a vector problem. This raises the question: Why should we bother distinguishing between these two classes of representation? There are two reasons, one practical and the other more theoretical. On the practical side, it is simply far more natural to think of some problems in terms of functional representations (e.g., LIP working memory; see section 3.4).

On the more theoretical side, because each element, A_m, of a vector of coefficients is related to some particular basis, $\Phi_m(\nu)$, we often have reasons to treat different A_m differently, depending on the kinds of dynamics or representation we observe in a given system. For example, if we have a Fourier basis, we may want to damp high frequencies and thus limit the gain of A_m associated with those frequencies (see, e.g., section 8.5). In contrast, cases of standard vector representations treat each component of the vector being represented equally (see, e.g., section 6.5).

In the next section, we show both how function representation can be used, and give an example of why it is often preferable to think of a system in terms of function representation rather than vector representation.

3.4 An example: Working memory

In computational neuroscience there has been a long-standing interest in constructing model neural networks that are able to store simple functions, especially single, uniform-height Gaussian ‘bumps’, ‘hills’, or ‘blobs’. These functions are stored as localized increases in neural activity approximately in the shape of a Gaussian function (technically not *true* Gaussians). Such activity ‘bumps’ have been thought to be present in various neural systems including the head direction system (Zhang 1996; Touretzky and Redish 1996; Skaggs et al. 1995), frontal working memory systems (Laing and Chow 2001), parietal working memory systems (Snyder et al. 1997), and visual feature selecting systems (Hansel and Sompolinsky 1998). Although there is a wide variety of systems that can be modeled as representing Gaussian bumps, we focus on parietal areas that store visual targets. For the time being, we also only focus on the ability of this neural system to represent functions, not store them (see section 8.3 for our discussion of the dynamics of these systems).

Representation in parietal areas is not a simple matter. For example, there is evidence that parietal areas can hold multiple saccadic targets in memory at the same time, suggesting that multi-modal functions can be stored (Platt and Glimcher 1997). As well, it has been shown that the activity in parietal areas is sensitive to non-spatial parameters (e.g., shape (Sereno and Maunsell 1998)), suggesting that a bump of activity at a single location can be different heights under different stimulus conditions.[5] None of the models proposed to date support either multiple bumps or bumps of different amplitudes (see Hansel and Sompolinsky 1998; Kishimoto and Amari 1979; Laing and Chow 2001; Camperi and Wang 1998).

3.4.1 System description

We are interested in modeling the behavior of a subpopulation of neurons in the lateral intraparietal area (LIP) of the neocortex. Although only about 5 millimeters square in macaque monkeys, LIP has been the focus of intensive study (see Andersen et al. 1997; Colby and Goldberg 1999 for a review). LIP receives direct projections from extrastriate visual areas and projects to the superior colliculus and frontal eye fields, both of which have been implicated in the control of saccadic eye movements. As well, LIP is strongly connected to other areas in posterior parietal cortex, including areas 7a, 7b, VIP and MST (Andersen et al. 1997). By far the majority of cells in LIP respond to a visual stimulus placed in their receptive field during a fixation task. Gnadt and Andersen (1988) showed that during a delay period after the presentation of such a stimulus, many neurons in LIP had a sustained firing rate. Subsequently, a debate has developed as to whether these neurons primarily code the intent of the animal to move to a target (Andersen et al. 1997) or whether the firing is a result of drawing the attention of the animal to the location (Colby and Goldberg 1999). However, participants on both sides of the debate agree that a significant number of the cells in LIP code the remembered location of the stimulus (Mazzoni et al. 1996). It is this subpopulation of the cells, and their ability to code remembered locations that we are interested in.

During the delay period, it is clear that the sustained firing rates serve to encode the spatial position of the stimulus. For simplicity, we describe only the encoding of positions along only one spatial dimension, ν. Given the evidence that multiple targets can be represented concurrently in LIP (Platt and Glimcher 1997), and that non-spatial parameters of the stimulus, in particular shape, can affect the response of neurons in LIP (Sereno and Maunsell 1998), LIP representation must be more complex than previously thought. It is undoubtedly the case that the effect of shape on tuning curves in LIP is both subtle and complex. But, for simplicity we take it that, at a minimum, these experiments show that the functions encoded in LIP must be sensitive to parameters of the stimulus other than just location. In this model, we let the amplitude of the encoded function serve to reflect this sensitivity to non-spatial parameters. Thus, a tall bump at a particular location encodes something different (e.g., a wide stimulus) than a short bump at the same location (e.g., a

5 This has also been shown for frontal working memory systems, which are also often modeled as bump networks (Romo et al. 1999).

narrow stimulus). In sum, we take this subpopulation of LIP neurons to represent a set of functions, $x(\nu; \mathbf{A})$, that are composed of one or more Gaussian-like bumps, each of which can be of a different amplitude. Although this is clearly a simplification of the space of functions that are probably represented by these LIP neurons, it is sufficiently complex to capture some of the most salient properties of the area.

We now need to characterize the kinds of tuning curves found in LIP that are used to represent this ensemble of functions. Fortunately, most neurons exhibit Gaussian-like tuning curves, with a median standard deviation of about 5 degrees (Platt and Glimcher 1998). Focusing on just one spatial dimension, we can write the tuning curves in the standard form, i.e.,

$$a_i(\mathbf{A}) = G_i \left[\alpha_i \left\langle x(\nu; \mathbf{A}) \tilde{\phi}_i(\nu) \right\rangle_\nu + J_i^{bias} \right],$$

where the encoding functions, $\tilde{\phi}_i(\nu)$, are uniform height Gaussians that are evenly distributed over the visual field. The neuron response properties, defined by G_i, α_i, and J_i^{bias}, are assumed to be such that they result in an even distribution of thresholds over the range of possible amplitudes of the function $x(\nu; \mathbf{A})$. Thus, there are some neurons that respond strongly to low amplitude stimuli and others that do not respond until the stimuli is of quite high amplitude. Notably, in LIP, the width of the tuning curve varies approximately linearly with eccentricity (Platt and Glimcher 1998). Thus, there are narrow ($\sigma \approx 1$ degree) tuning curves near the fovea and wider ($\sigma \approx 30$ degrees) tuning curves in the periphery. For simplicity, we do not include this variation of tuning curves, but assume $\sigma \approx 10$ degrees.[6] A sample of the population of tuning curves is shown in figure 3.2.

3.4.2 Design specification

Because we are interested in the representation of functions, the design specification step is broken into two parts. First, we must identify the function space that is to be represented (i.e., the ‘higher-level’ representation). Then, we must specify the more implementation-oriented constraints on the neural representation (i.e., the ‘basic’ representation). The first step is concerned with defining the range and precision of the idealized representation (in an orthonormal space). The second step is concerned with specifying how we implement this idealized representation in a neural population.

To specify the range and precision of the functions to be represented in the orthonormal space, we must define the structure of the distribution of parameters, $\mathbf{A}$, that determine the ensemble of functions to be represented. As mentioned in section 3.2, it is often difficult to explicitly write down an expression for this distribution, $\rho(\mathbf{A})$, so we must resort to other means of specifying the relevant function space.

6 If this is disconcertingly unrealistic, note that we can consider our model and results to be an inversely, linearly-scaled model of the actual system. This kind of distortion will not affect our results in any significant way.

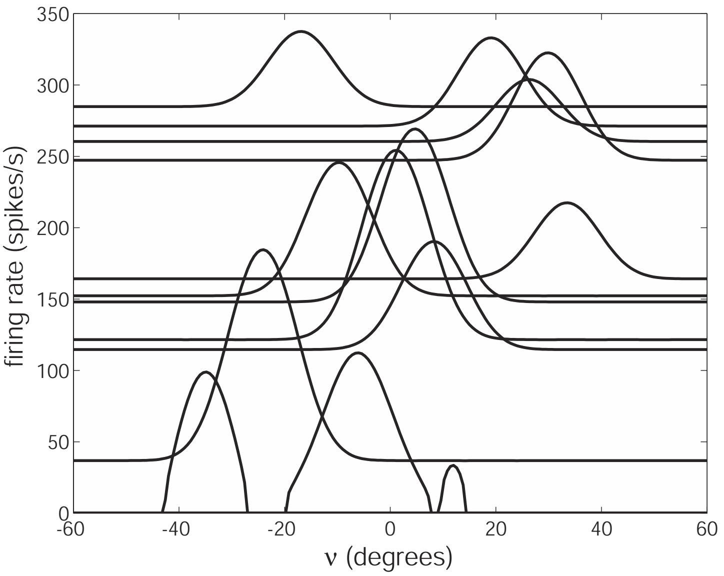


Figure 3.2
A sample of tuning curves for a simulated population of LIP neurons. Note that the neurons have diverse background firing rates, widths, and heights as seen in LIP.

Nevertheless, to begin we can re-write our standard expression for the ensemble of functions from (3.2):

$$x(\nu;\mathbf{A}) = \sum_m A_m \Phi_m(\nu) \text{ for } \mathbf{A} \in \rho(\mathbf{A}), \text{ and } m \in \{1, \ldots, M\}. \tag{3.19}$$

We have thus defined our function ensemble as a linear combination of some orthonormal basis, $\mathbf{\Phi}(\nu)$. It is straightforward to specify the range of domain of the functions, $x(\nu;\mathbf{A})$. In this case, we note that the visual field spans approximately 120 degrees of arc, so we write

$$x(\nu) \text{ for } \nu \in (\nu_{min}, \nu_{max}), \tag{3.20}$$

where $\nu_{min} = -60^\circ$ and $\nu_{max} = 60^\circ$, with 0° being straight ahead. For convenience, we often normalize this range so that $\nu \in (-1, 1)$. As well, since the basis is orthonormal, we can specify the range of allowed coefficients of the functions to be represented as

$$A_m \in (A_{min}, A_{max}),$$

where we choose $A_{min} = 0$ and $A_{max} = 1.0$ for this example. Thus we assume that the non-spatial parameter represented in LIP is normalized to the interval $(0,1)$, in the appropriate units.

Before we can place more constraints on the coefficients, we must have some idea of what the orthonormal basis functions are. Suppose, for the time being, that we will use sine and cosine functions, as if we were doing a Fourier expansion. One of the main constraints we have to impose in defining our ensemble of functions, is the number of bases that we use, denoted by M in (3.19) above. Since, in this case, we have limited the set of functions to be represented in our model to combinations of Gaussians, we can choose a maximum frequency that allows us to represent Gaussians of some particular width, σ. We can find this frequency by taking the Fourier transform of a Gaussian of width σ. Since we are assuming that our representation is a linear combination of encoding functions, $\tilde{\phi}_i(\nu)$, we know that we will not be able to successfully represent anything narrower than these encoding functions. Thus, a reasonable choice for σ is about two or three times larger than the encoding function width, since such functions can be well represented. Taking the Fourier transform of a Gaussian with $\sigma = 10°$ gives a maximum frequency of about 0.4 cycles/degree. Then by the Nyquist sampling theorem, we know that we need $M \approx 20$ frequency channels to represent any function over the visual field well.

We have now precisely defined the space that our ensemble of smooth (i.e., no frequencies above 0.4 cycles/degree) functions occupy. However, we do not want to represent *all* functions in this space, since we are interested only in a subset of the functions that are sums of Gaussians. Ideally, we would now construct the probability density function, $\rho(\mathbf{A})$, that picks out only those functions we want. Because this is unduly difficult, we instead 'sample' the ideal density function by explicitly finding the amplitudes for functions of the kind we want to represent, and then averaging our error over just *those* functions, as discussed earlier (see section 3.2).

This completes the design specification in the orthonormal space. We now need to state our assumptions about the representation at the level of the individual neurons. As before, we take the noise to have a variance of 0.1. In order to encode the range of amplitudes between A_{min} and A_{max} in an unbiased way, we assume an even distribution of neuron intercepts. Essentially, we can think of each of the parameters, A_m, as being a scalar that we want to encode with a certain precision. This problem is perfectly analogous to the problem of representing x to a certain precision in the eye position example (see section 2.3). Thus, we can use what we learned from that example so, assuming that we want to encode our A_m with a signal-to-noise of about 25:1, we need about 50 neurons per A_m parameter. Thus, our network will consist of $M \times 50 = 1000$ neurons. We have now specified the range and precision of our encoding for both the higher-level and basic representations.

3.4.3 Implementation

As usual, we find the population decoders by minimizing the error expressed in (3.7). We can use these population decoders in combination with LIF neuron responses to determine our estimate of the encoded function. Later, we use a spiking version of this representation to support a model of the dynamics of LIP (see section 8.3). For the time being, however, let us look at the resulting function representation in more detail.

In section 3.4.2 we assumed that the orthonormal basis would be somewhat Fourier-like. However, we know that they will not be a Fourier basis because, given what we know about LIP, a set of infinite, cyclic functions will not be a viable basis. Specifically, because the functions we are interested in representing are defined over a *finite* space (i.e., from -60 to 60 degrees), we need to find an orthonormal basis over *that* space.

To find the desired orthonormal basis, we must define the space of possible functions to be encoded. This can be done by taking all of the encoding functions, $\tilde{\phi}_i(\nu)$ in the population together and determining what the space is spanned by those encoders (which are not orthonormal and thus overcomplete). We can then use a technique known as Singular Value Decomposition (SVD) to find an orthonormal basis that spans the same space as the encoders (see figure 3.3 for the result). In section 7.2 we describe these concepts and techniques (i.e., SVD, spanning, overcomplete representations, etc.) in more detail. For now, let us just assume this result.

This orthonormal basis is very much like a Fourier basis restricted to a finite interval. Successive functions have one additional half-cycle over the range of ν. However, unlike the Fourier basis, all of these functions go to zero at the extremities of the variable being encoded. This is very important because it ensures that there will not be any undesirable 'end-effects' that might disrupt a good representation.

Now that we have defined the relevant orthogonal basis, $\Phi_m(\nu)$, for defining the functions that we take to be represented in LIP, we are in a position to switch our characterization of this system from a function representation to a vector representation if we so choose. However, because there is an intuitively close relationship between Gaussian bumps at multiple positions and bumps of neural activity at those positions, and because it is far less intuitive how the vectors of coefficients that represent these bumps with respect to some basis are related, it is simply *easier* to talk about the Gaussian bumps than the equivalent vectors of coefficients. But we should note that while there is no reason to perform this switch when considering representation in isolation, it provides a significant practical advantage when considering the dynamics of this system as demonstrated in section 8.3.

Of course, regardless of whether we characterize this problem as a function representation or vector representation problem, we can find the optimal linear decoders for the function space we have defined. Doing so results in a very good representation of the kinds

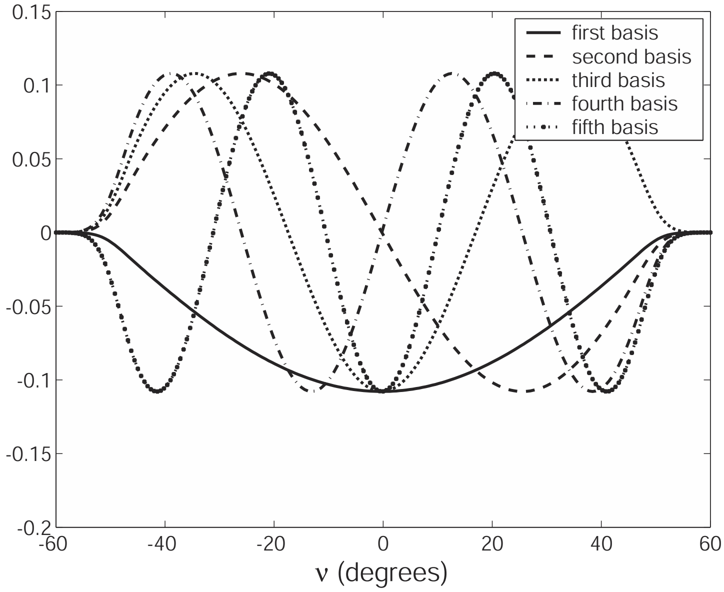


Figure 3.3
The first five orthonormal basis functions found using SVD on the population encoders.

of functions observed in LIP. Figure 3.4 shows the neural representation found using those decoders. In fact, the representation is so good that the error is unnoticeable in this graph. However, once dynamics are introduced into the model, the effects of the error become quite obvious (see section 8.3).

3.4.4 Discussion

Considering neural representation in LIP helps demonstrate that characterizing function representation is only slightly more challenging than characterizing scalar and vector representation, despite the fact that function representations often seem significantly more complex. For instance, the challenge of constructing a system that can represent multiple Gaussian bumps of different heights may seem significantly more difficult than constructing a system that can represent a single scalar. However, the methodology and theory are identical: define the neural encoding and find the optimal linear decoders. Using this approach, we have built the ability to support fairly complex representations directly into our model.

Certainly this representation could be improved upon in a number of ways: we could more carefully model the distribution of tuning curves in LIP; we could more carefully

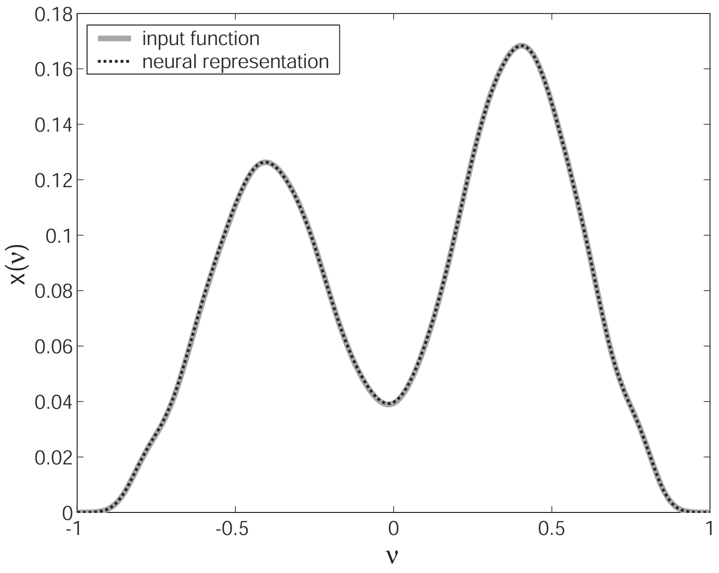


Figure 3.4
Neural representation of a 'double bump' function. Notice that each bump is a different height.

characterize the set of functions that can be represented by LIP; we could use two-dimensional tuning curves (as we have done elsewhere Eliasmith and Anderson 2001); and, as always, we could use more realistic spiking neurons. However, each of these improvements is not outside the scope of the general approach we have taken, and some we address in more detail later. More importantly, the model outlined here is a good, simple first step to building these more complex models.

3.5 Summary

We began this chapter by arguing that our previous analysis naturally leads to the construction of a representational hierarchy for neurobiological representation. We suggested that being able to construct such a hierarchy demonstrates that this approach is unified and general, yet flexible. Table 3.3 summarizes the parts of the hierarchy we have explicitly defined, and that we use in the remainder of the book.

We then showed that, in fact, vector and function representation are intimately related. In particular, function representations can be implemented as vector representations with respect to some orthonormal basis. Nevertheless, it is often more intuitive to characterize neural systems as representing functions. We presented one such example in the case of modeling the representations in area LIP.

Table 3.3
A summary of population representation for scalars, vectors, and functions.

Representation	**Definition**
Scalar	$x \in (x_{min}, x_{max})$ $a_i(x) = G_i[J_i(x)]$ $\hat{x} = \sum_i a_i(x)\phi_i$ $J_i(x) = \alpha_i x + J_i^{bias}$ $E = \left\langle \left[x - \sum_i (a_i(x) + \eta_i)\phi_i\right]^2 \right\rangle_{x,\eta}$ $\boldsymbol{\phi} = \boldsymbol{\Gamma}^{-1}\boldsymbol{\Upsilon}$ $\Gamma_{ij} = \gamma_{ij} + \sigma_\eta^2 \delta_{ij}$ $\gamma_{ij} = \langle a_i(x) a_j(x) \rangle_x$ $\Upsilon_i = \langle x \cdot a_i(x) \rangle_x$
Vector	$\mathbf{x} = \{x_1, \ldots, x_D\}$ for $\mathbf{x} \in Vol_D$ $a_i(\mathbf{x}) = G_i[J_i(\mathbf{x})]$ $\hat{\mathbf{x}} = \sum_i a_i(\mathbf{x})\boldsymbol{\phi}_i$ $J_i(\mathbf{x}) = \alpha_i \left\langle \mathbf{x}\tilde{\boldsymbol{\phi}}_i \right\rangle_n + J_i^{bias}$ $E = \left\langle \left[\mathbf{x} - \sum_i (a_i(\mathbf{x}) + \eta_i)\boldsymbol{\phi}_i\right]^2 \right\rangle_{\mathbf{x},\eta}$ $\boldsymbol{\phi} = \boldsymbol{\Gamma}^{-1}\boldsymbol{\Upsilon}$ $\Gamma_{ij} = \gamma_{ij} + \sigma_\eta^2 \delta_{ij}$ $\gamma_{ij} = \langle a_i(\mathbf{x}) a_j(\mathbf{x}) \rangle_{\mathbf{x}}$ $\Upsilon_i = \langle \mathbf{x} a_i(\mathbf{x}) \rangle_{\mathbf{x}}$
Function	$x(\nu)$ for $\nu \in (\nu_{min}, \nu_{max})$ $x(\nu; \mathbf{A}) = \sum_m^M A_m \Phi_m(\nu)$ for $\mathbf{A} \in \rho(\mathbf{A})$ $a_i(x(\nu; \mathbf{A})) = a_i(\mathbf{A}) = G_i[J_i(\mathbf{A})]$ $\hat{x}(\nu; \mathbf{A}) = \sum_i a_i(\mathbf{A})\phi_i(\nu)$ $J_i(\mathbf{A}) = \alpha_i \left\langle x(\nu; \mathbf{A})\tilde{\phi}_i(\nu) \right\rangle_\nu + J_i^{bias}$ $E = \left\langle \left[x(\nu; \mathbf{A}) - \sum_i (a_i(\mathbf{A}) + \eta_i)\phi_i(\nu)\right]^2 \right\rangle_{\mathbf{A},\eta,\nu}$ $\boldsymbol{\phi}(\nu) = \boldsymbol{\Gamma}^{-1}\boldsymbol{\Upsilon}(\nu)$ $\Gamma_{ij} = \gamma_{ij} + \sigma_\eta^2 \delta_{ij}$ $\gamma_{ij} = \langle a_i(\mathbf{A}) a_j(\mathbf{A}) \rangle_{\mathbf{A}}$ $\Upsilon_i(\nu) = \langle x(\nu; \mathbf{A}) \cdot a_i(\mathbf{A}) \rangle_{\mathbf{A}}$

4 Temporal representation in spiking neurons

Few neurobiologists will heed any characterization of neural representation if it depends on the assumption that neurons output real-valued firing rates. This is because it might come as no surprise that real-valued output from neurons can be used to represent signals changing constantly in real time. But neurons in real nervous systems traffic in neural spikes. As a result, what we need to know is how these highly discontinuous, nonlinear outputs can be used to successfully represent continuous temporal signals. In other words, neural spikes are fundamental features of neurobiological systems that present an unavoidable challenge for understanding neural representation in its 'full-fledged' form; i.e., as the encoding and decoding of time-varying signals in populations of spiking neurons.

In this chapter, our goal is precisely to understand the representation of time-varying signals by spiking neurons. In order to reach this goal, we begin by introducing a spiking version of the leaky integrate-and-fire (LIF) model. We use this model extensively in our subsequent simulations and thus begin by discussing its strengths and weaknesses. Eventually, we demonstrate that our framework does not, in any way, depend on this particular neural model. Neverthelesss, it is a useful place to begin our consideration of temporal encoding in neural systems.

Again, because we are interested in characterizing representation, we are confronted with the challenge of developing a method for decoding the results of this kind of encoding as well. In many ways, the approach we develop is analogous to that in the previous chapters; we again rely on the mean square error to find optimal linear decoders. There are some important differences as well, but the similarities hint at the fact that the first principle outlined in chapter 1 applies to both population and temporal coding.

4.1 The leaky integrate-and-fire (LIF) neuron

4.1.1 Introduction

The properties of the leaky integrate-and-fire (LIF) model neuron have been investigated since 1907, even before much was known about actual spike generation in neural systems (Arbib 1995; Koch 1999, p. 335). Since that time, the LIF model has become a standard approximation to the complex behavior exhibited by real, spiking neurons. It is widely used for a number of reasons. First, it is a very simple model that affords a good approximation to the behavior of many kinds of real neurons under a wide range of conditions. Second, it has been shown to be a limiting case of more complex conductance models such as the well-known Hodgkin-Huxley model (Partridge 1966). Third, though simple, the LIF

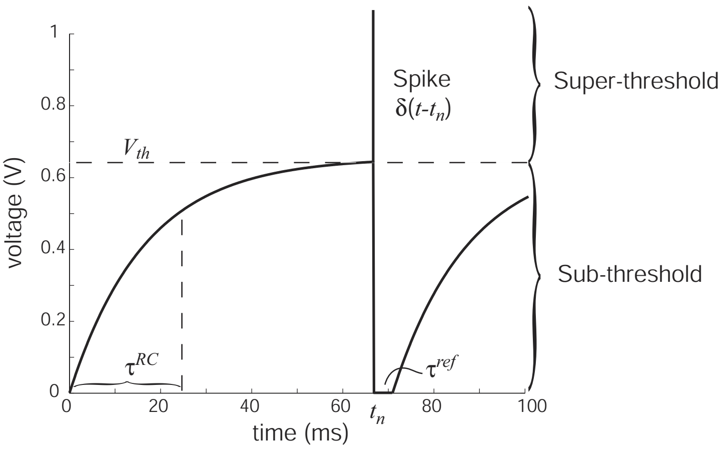


Figure 4.1
Leaky integrate-and-fire (LIF) neuron with constant current input. When the sub-threshold soma voltage reaches the threshold voltage, V_{th}, the model emits a delta function spike, $\delta(t - t_n)$. The model is then reset for a time, τ^{ref}, before again integrating its input. As discussed in section 4.1.2, the sub-threshold regime is well-characterized by an RC circuit, parameterized by τ^{RC} .

neuron is far more realistic than rate neurons because it introduces the most prominent nonlinearity in neural systems—the neural action potential, or spike.

It is this third reason that is most important for those concerned with understanding neural representation. As we show, being able to understand the neural code as modeled by the spiking behavior of the LIF neuron goes a long way to understanding representation in far more detailed neural models (see section 4.5). Thus, we could have based our discussion of representation on one of the more complex models found in section 4.5, but, as that section shows, little theoretical insight would be gained. The LIF neuron, then, is a convenient and fruitful mixture of realism and simplicity (Koch 1999, see especially chp. 14).

In its standard form, the LIF neuron has two behavioral regimes: sub-threshold and super-threshold. As shown in figure 4.1, the sub-threshold behavior during a constant soma input consists of an ever-slowing approach towards the input voltage. Once the threshold is reached, super-threshold behavior begins and an infinitely thin and infinitely tall delta-function voltage spike ($\delta(t - t_n)$) with unit area (i.e., $\int_{-\infty}^{\infty} \delta(t - t_n)\, dt = 1$) is produced. The system is then reset to zero for some time period, τ^{ref}, before it is allowed to again begin its approach to threshold.

There are a number of physiologically motivated features of this model. First, the spike which is produced is both narrow and stereotypical as is seen in many real neurons. Of course, representing the spike as a delta function is a mathematical convenience. However, as long as the width of the neural spike in the neuron being modeled is small compared to the typical interspike interval, using a delta function approximation is warranted. Typically, neural spikes are about 1–2 ms in width compared to interspike intervals between 50 and 200 ms (i.e., 50–200 Hz), so the delta-function approximation is a good one. Second, the absolute refractory period, τ^{ref}, that forces the LIF model voltage to zero for a short duration after a spike is emitted is an excellent approximation to a similar phenomena found in real neurons (Koch 1999). Third, the sub-threshold leaky integration of the model is produced by a simple passive resistance-capacitance circuit whose elements have physiological correlates. We introduce and discuss this circuit in more detail in the next section.

As we note in sections 4.1.3 and 4.5, there are a number of shortcomings of the LIF model as well. But the realism, or lack thereof, of LIF neurons is itself of little importance to us because we are only interested in the model to the extent that it allows us to construct a useful formalism for understanding neural representation. Showing that the formalism we construct using LIF neurons is generally applicable is the burden of sections 4.3–4.5.

4.1.2 Characterizing the LIF neuron

The LIF neuron is best understood as a passive RC circuit coupled to an active spike (i.e., delta function) generator (see figure 4.2). This circuit provides a simple means of describing the time course of the membrane potential, V. The membrane potential is the result of the interaction between charges, Q, in the fluids inside and outside the cell. These charges are separated by the cell membrane, which acts as a dialectric layer. The capacitor, C, in the LIF circuit accounts for the charge build-up on either side of the bilipid layer that comprises the membrane. Notably, there is never any movement of charge across the lipid membrane itself (Jack et al. 1975, p. 13). Rather, the capacitive current J_C results from a change in the amount of charge separated by the membrane. Given that the voltage across the capacitor is $V = Q/C$, we can find J_C by differentiating:

$$J_C = C\frac{dV}{dt}. \tag{4.1}$$

In addition to the capacitance of the bilipid layer, the LIF model includes a leak resistance, R, that models the effects of (some of) the proteins embedded in the lipid membrane. These particular proteins act as ion channels, allowing sodium, potassium, and chloride ions to pass through the otherwise impervious membrane. The concentration gradients across the neuron membrane of the various ions results in the movement of ions

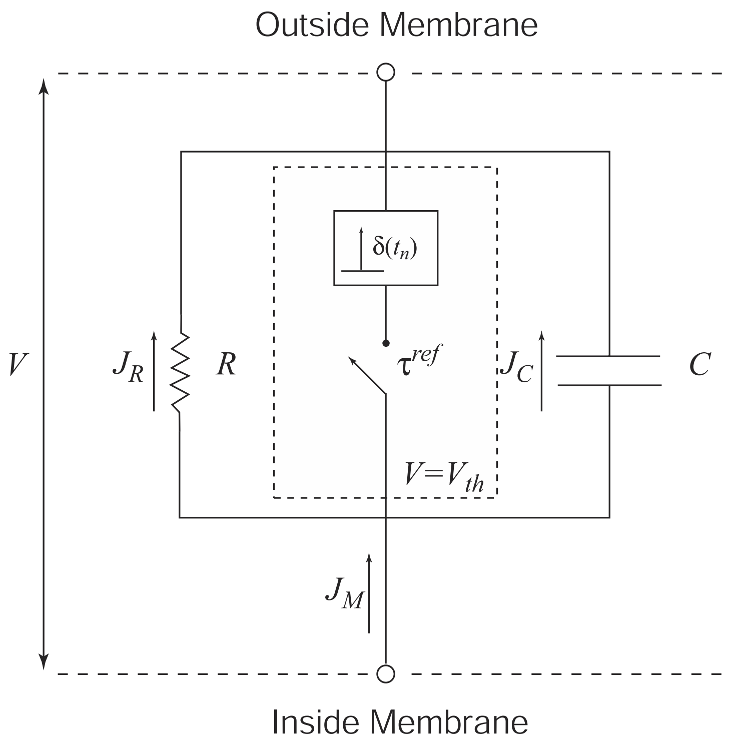


Figure 4.2
An RC circuit that implements the LIF neuron. The standard RC circuit, describing the sub-threshold behavior of the neuron is that part outside of the dashed box. It consists of the membrane potential, V, the leak resistance, R, and the capacitance due to the bilipid layer, C. The active, super-threshold behavior is described by the additional components inside the dashed box. When the membrane potential is equal to the voltage threshold, V_{th}, at a time, t_n, the short-circuit switch is closed, resulting in a 'spike', $\delta(t_n)$. The switch remains closed, resetting the circuit and holding $V = 0$, until it is opened after the refractory period, τ^{ref}.

into and out of the cell. The ionic current, J_R, accounts for this passive 'leak' of charge across the membrane.[1] Ohm's law tells us that the ionic current is

$$J_R = \frac{V}{R}. \tag{4.2}$$

The final current of importance to the LIF model is the membrane current, J_M. In effect, this current represents the input to the model. It can be thought of as the somatic

1 Because we are assuming that the circuit is passive, R is constant and we can assume that linear cable theory is applicable (see Jack et al. 1975, chapters 8, 9, and 10 for a discussion of nonlinear cable theory applied to cellular models).

current resulting from all of the postsynaptic currents (PSCs) generated at the dendrites. As mentioned previously, we consider this current to be comprised of two distinct components, the bias or background current, J^{bias}, and the drive current, J^{d}. The bias current is an ever-present, steady state current, whereas the drive current accounts for the rapid current fluctuations due to dendritic inputs.

Since the flow of charge must be conserved between the inside and outside of the membrane (i.e., Kirchoff's law applies), we know that

$$J_M = J_C + J_R. \tag{4.3}$$

Substituting equations (4.1) and (4.2) into (4.3) and rearranging gives

$$\begin{aligned} J_M &= C\frac{dV}{dt} + \frac{V}{R} \\ \frac{dV}{dt} &= -\frac{1}{\tau^{RC}}\left(V - J_M R\right), \end{aligned} \tag{4.4}$$

where $\tau^{RC} = RC$.

Recall that this ordinary, first-order, partial differential equation only describes the passive behavior of the LIF model. In figure 4.2, the distinction between the active and passive components of the circuit is denoted by a dashed box. Once the membrane potential, V, crosses the LIF neurons threshold, V_{th}, the components in this box control the model's behavior. In particular, the gate denoted by τ^{ref} closes and a delta function, $\delta(t_n)$, spike is generated. By short-circuiting the capacitor and resistor, the gate sets the potential across the membrane to zero (i.e., the assumed resting potential) since there is no way for a difference in charge to build up. This gate stays closed for a length of time, τ^{ref}, equal to the absolute refractory period of the neuron being modeled.

Solving equation (4.4) for V is examined in some detail in appendix B.2. The result, which can be verified by differentiating, is

$$V(t) = J_M R\left(1 - e^{-t/\tau^{RC}}\right). \tag{4.5}$$

From this equation, we can determine the effects of past input on the model's behavior. In effect, we can begin to understand when the model's 'memory' is important, and when it is not. To do this, let us consider changes to the membrane time constant, τ^{RC} (i.e., changes to either the resistance or capacitance). First, we can see that under a constant input current, J_M, and after a length of time equal to τ^{RC}, $V(t)$ will be equal to approximately two-thirds (i.e., $1 - e^{-1}$) of the steady state input, $J_M R$. So, a larger time constant means both that it takes longer to get up to threshold for above-threshold inputs, and that a slower 'forgetting' is occurring when there is no input. Notably, τ^{RC} becomes much less important if the input current is extremely high. In this case, $V(t)$ becomes nearly linear between the resting voltage and the threshold voltage for most values of τ^{RC}. Given the fact that the circuit is reset after the threshold is passed, high input current results in very little effect of past states on current behavior. Thus, there is an interplay between τ^{RC} and the magnitude of the input current that determines the relevance of past input to the LIF neuron behavior.

In real neural systems, of course, the input current, J_M, is not static, but time varying. As we discuss in appendix B.2, the model's behavior under these conditions can be understood as a convolution between the input signal and the exponential leak. Under that characterization, the voltage right now (at t) depends on all past current input, $J_M(t)$, where the input is weighted by a function that exponentially decays as that input gets further away (in the past) from the present time. In other words, the present voltage depends most strongly on the most recent inputs, and exponentially 'forgets' about past inputs as they get farther away. It is, in fact, this latter description that most accurately captures the nonlinearity inherent in the LIF model (and which we use in our simulations). However, equation (4.5) is easier to handle analytically because J_M is assumed to be relatively static.

In particular, we can use (4.5) to derive an expression for the neuron response rate curve that we used extensively in chapter 2. We know that the steady-state firing rate of an LIF neuron is inversely proportional to the length of time it takes the RC circuit to pass V_{th}. Note that because we are here concerned with steady-state firing (i.e., constant input current), this derivation is based on the assumption that the input current is changing slowly compared to the interspike interval as assumed when finding (4.5). Under this assumption, the time to threshold is equal to the length of time, t_{th}, it takes the circuit described by equation (4.5) to pass V_{th} plus the absolute refractory period, τ^{ref}:

$$a(t_{th}) = \frac{1}{t_{th} + \tau^{ref}}. \tag{4.6}$$

From equation (4.5) we can find t_{th} as follows:

$$\begin{aligned} V_{th} &= J_M R \left(1 - e^{-t_{th}/\tau^{RC}}\right) \\ t_{th} &= -\tau^{RC} \ln\left(1 - \frac{V_{th}}{J_M R}\right). \end{aligned} \tag{4.7}$$

Substituting (4.7) into (4.6), we find that the firing rate as a function of the input current J_M is

$$a(J_M) = \frac{1}{\tau^{ref} - \tau^{RC} \ln\left(1 - \frac{V_{th}}{J_M R}\right)}. \tag{4.8}$$

Note that $V_{th} = J_{th}R$, so we can cancel the resistance terms. As well, if we know what the input current is as a function of some 'external' variable, x, we can re-write equation

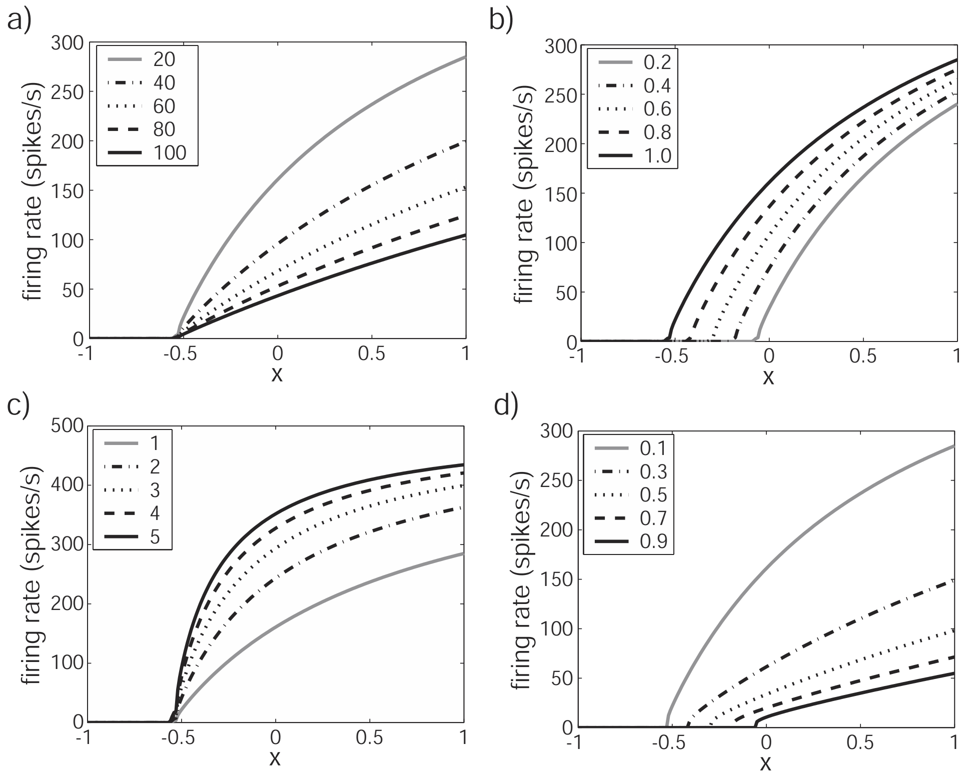


Figure 4.3
The effects of changing various parameters on the LIF response function. a) Varying τ^{RC} between 20 and 100 ms. b) Varying J^{bias} between 0.2 and 1.0 nA. c) Varying τ^{ref} between 1 and 5 ms. d) Varying J_{th} between 0.1 and 0.9 nA. Note that $\alpha = 17$, and unless explicitly changed, $J^{bias} = 10$, $J_{th} = 1$, $\tau^{RC} = 20$, $\tau^{ref} = 1$. Changing these parameters is partially redundant as demonstrated by comparing a), b), and d). Specifically, varying J_{th} is like varying both τ^{ref} and J^{bias}, thus in subsequent models we do not vary J_{th}.

(4.8) in the form we first encountered it in section 2.1.2, i.e.,

$$a(x) = \frac{1}{\tau^{ref} - \tau^{RC} \ln\left(1 - \frac{J_{th}}{J_M(x)}\right)}, \tag{4.9}$$

where $J_M(x) = \alpha x + J^{bias}$. Recall from section 2.1.2 that α is both the gain and a unit conversion factor, and J^{bias} accounts for the steady background input to the cell. Figure 4.3 demonstrates the effects of changing various of the parameters in this equation on the shape of the neuron response function.

4.1.3 Strengths and weaknesses of the LIF neuron model

We have now progressed from basic considerations of neuron physiology and simple circuits through a derivation of the spiking LIF model to arrive at a derivation of a rate LIF model. In order to motivate our focus on the LIF model, we have already discussed many of its strengths. To summarize, the LIF neuron model:

1. naturally incorporates a number of physiological parameters, including membrane capacitance, membrane (passive leak) resistance, and absolute refractory period;
2. is a good approximation over the normal operating range (i.e., low firing rates) of most neurons; and
3. introduces the 'important' nonlinearity of real neurons, i.e., the neural spike.

Essentially, we agree with Koch (1999) that "such single cell models represent the most reasonable trade off between simplicity and faithfulness to key neuronal attributes" (p. 335). In addition, it is important for the characterizations of representation we have introduced previously that there is an analytic expression for the rate curve of such a neuron. As can be seen from section 2.1.2, this approach depends on our being able to define $a_i(x)$ in order to minimize the error expressed by equation (2.4). Being able to write $a_i(x)$ explicitly as in equation (4.9) makes it easier to solve this error for large populations of neurons. In particular, it makes it possible to generate large populations of model neurons whose tuning curves can be easily manipulated.[2]

The weaknesses of the LIF model have been extensively discussed in the neuroscientific literature (Koch 1999; Softky and Koch 1995; Jack et al. 1975; Shamma 1989). Many of the concerns regarding the use of LIF neurons can be summarized by noting that LIF neurons are physiologically unrealistic to some degree. For example, LIF neurons have no spatial extent; i.e., they are 'point' neurons. A real neuron is extended in space, so the membrane potential at one location may be very different from the potential at another location. Furthermore, being a point neuron means that the electrochemical properties of the dendrites are completely neglected in the LIF neuron. And, as a final example, the complex time courses of membrane ion conductances are not accounted for. Contrary to what is assumed by equation (4.2), the membrane ionic currents tend to be nonlinear functions of both the membrane potential and time (Jack et al. 1975). All of these sorts of physiological facts are simply ignored by the LIF model.

Why do we think ignoring all of these important details will not hamper our attempts at understanding neural coding? The answer is simple: our approach does not depend on

2 It is also possible to generalize the approach we take to neurons that have dynamic tuning curves (e.g., adapting neurons). We do not discuss this generalization in any detail, although see section 4.5 for an approach to understanding such neurons that does not depend on this generalization.

how spikes are generated. It only depends on the statistics of spike generation. Of course, including more physiological detail will make our model statistics more closely match the actual statistics of spike generation. For this reason, detailed models are eventually important (see Deco and Schurmann 1998); we discuss a number of them in section 4.5). However, our method for analyzing *some statistics or other* remains the same.[3]

It has been suggested by Softky and Koch (1993) that the problems with models like the LIF go much deeper. They claim that such models do not even have the right kinds of statistics. In particular, they show (using LIF neurons as an example) that models which assume linear summation of dendritic inputs are inconsistent with the highly irregular firing statistics found in most cortical neurons. Given the standardly assumed (i.e., Poisson) statistics of spikes impinging on a LIF neuron with linear dendrites, the output of that neuron will be *less* variable than its input. This is simply a consequence of the 'law of large numbers' (or 'Bernoulli's theorem') familiar to probability theorists. This law states that the variance of a random variable linearly comprised of other random variables, each with a variance σ^2, is equal to $\frac{\sigma^2}{n}$, where n is the number of random variables. Clearly, as n becomes large the variance of the resulting variable becomes small. This means that standard LIF models that sum dendritic PSCs will have a lower variability in their firing rate than their randomly firing input neurons. However, this is *not* a problem with the LIF neuron itself, but a problem with how the input current, J_M, is determined. Suffice it to say that there are a number of ways that J_M can be determined such that the output from a LIF neuron is sufficiently variable. So, there are ways to make LIF neurons have the relevant statistics, which means that the statistical analyses we perform can generalize properly.

4.2 Temporal codes in neurons

When discussing temporal representation (a.k.a. temporal coding) in neurons, it is difficult to avoid the vigorous debate between those who take the code to be a rate code (Shadlen and Newsome 1994; Shadlen and Newsome 1995; Buracas et al. 1998), and those who take it to be a timing code (de ruyter van Steveninck et al. 1997; Softky 1995; Rieke et al. 1997). In this section we show why this debate is, from our perspective, irrelevant to a good understanding of temporal representation in neurons.

Both rate codes and timing codes are clearly time dependent codes. A rate code, as proposed by Adrian (1928) is one that takes the information about a stimulus to reside in the mean firing rate of a spike train over a relatively long time window (about 100 ms). From this perspective, which still finds favor with some neuroscientists, the observed

3 Our approach is certainly not unique and the independence of this kind of approach from particular models has been noted before (de ruyter van Steveninck and Bialek 1988).

variability about this mean firing rate is considered to be noise (see, e.g., Shadlen and Newsome 1994; Shadlen and Newsome 1995). So defined, there are a wide variety of problems with adopting such a rate code as being a general feature of neural systems. First, most animals are embedded in highly dynamic environments and encounter signals that change very rapidly. If these animals needed to integrate information over an extended period (of even 100 ms), they would have little chance of survival. Indeed, there is plenty of evidence that many behavioral decisions are made on the basis of one or two neural spikes, which can arrive only a few milliseconds apart (see Rieke et al. 1997, pp. 55–63 for a good review). Second, there are limitations of rate coding that clearly do not affect some neural systems. For example, the frequency/amplitude ambiguity[4] can be resolved by a timing code, but not by a rate code (Bialek and Rieke 1992). Real neural systems seem to have no problem with this ambiguity. Third, there is experimental evidence that different input spike trains with the same mean rate, but different temporal structure produce significantly different output from the same cell (Segundo et al. 1963). Fourth, Zador (1998) has shown that the existence of a mean rate code contradicts the observed redundancy of synaptic connections. In particular, he has shown that the information transmission of such a code *falls* with redundant synaptic connections. However, such connections are common in neural systems. Fifth, and lastly, rate codes cannot support the information transmission rates observed in real neurons (Rieke et al. 1997), although it has long been known that a timing code can (MacKay and McCulloch 1952). In conclusion, there are many reasons to think that the neural code is not a mean rate code.

So, is the neural code obviously a timing code? Unfortunately not. For instance, there is evidence that the precise timing of spikes is not mandatory for the successful transmission of neural signals (Bialek et al. 1991).[5] More generally, whether or not the neural code is a timing code depends on what we mean by ‘timing code’. The standard timing code is one that takes spike train *variability* to encode the stimulus signal (MacKay and McCulloch 1952; de ruyter van Steveninck and Bialek 1988; Softky 1995; Rieke et al. 1997). One way of expressing this kind of code is to take the inverse of the interspike intervals (i.e., 1/ISI) as a measure of the variations in a single trial. Because the same stimulus commonly elicits different spike trains, this kind of measure is often averaged over a number of trials. The averaged measure is sometimes called the ‘instantaneous’ rate code for the neuron (Buracas et al. 1998; Rieke et al. 1997). The term ‘rate code’ should not be too surprising here, as the measure is equivalent to a rate code where the window size approaches a limit

4 This ambiguity arises in auditory neurons because sounds with high amplitude that are not at the preferred frequency of a neuron can result in the same spike rate as sounds with low amplitude at the neuron’s preferred frequency.

5 Actually, for those who like timing codes, this is evidence of the ‘robustness’ of a timing code. For those who like rate codes, this is evidence of the lack of importance of precise timing.

of zero. However, the fact that the window size is so small has prompted many to consider this code a timing code.

There are other timing codes, as well. For example, Optican and Richmond (1987) suggest that information about spatial patterns can be found in an 'onset' timing code (see also Richmond and Optican 1990). They argue that the placement of spikes *relative to stimulus onset* carries information about the stimulus, and that this information can be about non-temporal features, such as shape (Optican and Richmond 1987). Although more recent results have contradicted this interpretation (Tovee et al. 1993), the ubiquity of adaptation in excitatory cortical neurons (i.e., the slowing of firing rates given sustained, super-threshold input) suggests that it may be the case that spike times relative to stimulus onset are important. The main difference between this kind of code and the instantaneous rate code is that non-temporal features are thought to be multiplexed into the time course of the neurons. So, traditional rate codes suffer a number of limitations, instantaneous rate codes do not seem to be rate codes, and onset timing codes are not empirically well supported.

This brief review gives some sense of the kinds of approaches to neural coding available, and why they are at odds. But, why did we say that choosing amongst these different codes is irrelevant? There are three reasons. The first is semantic. That is, there is little agreement as to whether instantaneous rate codes are rate codes (Buracas et al. 1998; Stevens and Zador 1995) or timing codes (Rieke et al. 1997; de ruyter van Steveninck and Bialek 1988). So, it would be unclear what we meant if we simply claimed that we were exploring a rate code or a timing code. The second reason is that it seems likely that the brain uses different codes for different problems (Zador 1998; Rieke et al. 1997); perhaps rapid sensory processing is more likely to use a timing code and static sensory processing is more likely to use a rate code (Buracas et al. 1998). This leads us to our third, and far more important reason for not 'choosing' a code: we simply don't have to. Given the statistical methodology that we adopt (section 4.3), the 'appropriate' code is determined by the signals that are represented and the properties of the neurons which represent those signals. In cases where the signals are rapidly varying relative to the interspike interval, something more like a timing code is appropriate. In cases where the dynamic range of the signals is large, but the correlation time is fairly long relative to the interspike interval, something more like a rate code is appropriate (see section 4.4). The method we discuss is effective for both of these cases. As well, in cases where the neurons adapt, thus changing the placement of spikes relative to stimulus onset, an appropriate code can be found again using the same method (see sections 4.5.1 and 4.5.3). In other words, we do not need to commit ourselves to one particular kind of coding as being central to neural representation. The approach we use here is general enough to characterize the kind of coding *appropriate to the problem at hand*. This understanding of neural coding thus transcends concerns about whether neurobiological systems use rate or timing codes.

4.3 Decoding neural spikes

4.3.1 Introduction

Recently, there has been a large amount of attention given to the information theoretic properties of neural spike trains (Bialek et al. 1991; Miller et al. 1991; Koch 1999; Stevens and Zador 1996; Richmond and Optican 1990; Roddey and Jacobs 1996; Bialek and Rieke 1992). The resulting attempts to decode neural spike trains have been largely successful, and provided many insights into neural coding. Given this success, there is no need to develop a completely new means of understanding neural representation. As a result, the methods we discuss in this section are a variation on past themes, to which we introduce only modest improvements. Thus, the importance of this section largely lies not in its novelty, but in the fact that it is an integrated part of a general, unified framework for modeling large-scale neural systems.

To begin, then, recall that a LIF neuron provides a good characterization of the temporal encoding process that is found in nervous systems (see figure 4.4). That is, a LIF neuron produces a series of stereotypical output spikes, $\delta(t - t_n)$, given some real-valued input signal, $x(t)$. To put this more precisely, and in the same form as encountered in chapter 2, we write

$$\begin{aligned} a(x(t)) &= G[J(x(t))] && (4.10)\\ &= \sum_n \delta(t - t_n), && (4.11)\end{aligned}$$

where

$$J(x(t)) = \alpha\tilde{\phi}x(t) + J^{bias}.$$

Here, the encoding function $G[\cdot]$ is defined by the parameters of the LIF neuron. As a result of these parameters, the model produces spikes (i.e., $\delta(t - t_n)$) at times t_n. In order to understand this output as a representation, we must find the relevant decoder.

In real nervous systems, it makes the most sense to think of peripheral neurons (e.g., retinal ganglion cells) as encoding some external, time-varying signal. This is because the encoded signals are relatively easy to pick out: they can be light intensities, velocities, pressure changes, or any other time-dependent physical magnitude. Such signals are fairly directly encoded into a series of neural spikes, $\delta(t - t_n)$, in nervous systems. As neurons become farther removed from the periphery, it is often less clear what the signal is that is being encoded. Nevertheless, a 'central' neuron is clearly part of *some* encoding process. The fact that the neuron is only one of many elements that give rise to the encoding

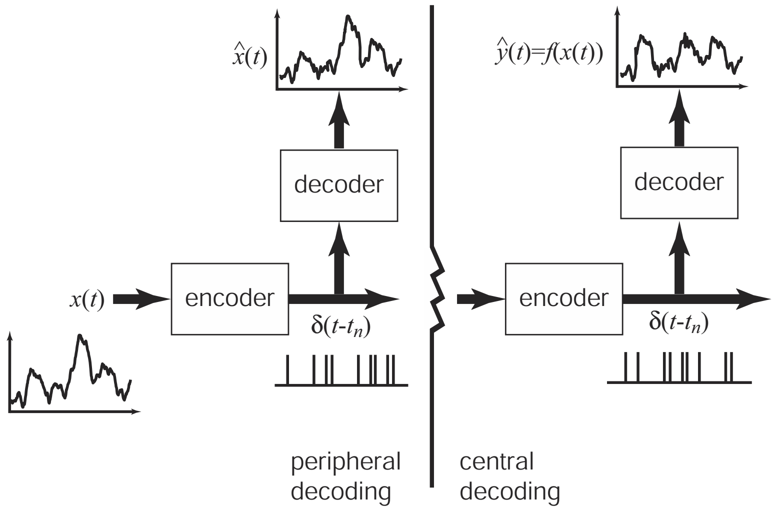


Figure 4.4
The temporal encoding and decoding process. Some physical signal, $x(t)$, such as light intensity is encoded by peripheral neurons into a series of spikes, $\delta(t-t_n)$. These can be passed to a (non-neural) decoder to give an estimate, $\hat{x}(t)$, of the original signal and thus help quantify the signal processing characteristics of the neural system. In more central neurons, the characterization is the same, but the signal being estimated, $y(t)$, will be some complex function of the input, i.e., $y(t)=f(x(t))$. Thus the decoder is a means of characterizing all of the encoding steps that precede that decoding. A long-standing problem in computational neuroscience is determining the relevant decoder (FitzHugh 1961).

process simply means that when we attempt to decode that neuron's spike train, we should not associate the decoder with that particular neuron regardless of its relations to other neurons. Rather, such decoders must be associated with the entire encoding process that they complement (see figure 4.4). But, importantly, the problem of characterizing neural representation remains the same.

Despite the fact that a central neuron is only one part of some highly complex encoding process, we can still uniquely identify each neuron with one encoding process. As a result, we can assume that the decoder associated with a neuron is the decoder for the encoding process for which that neuron is the terminus. With this in mind, we will not be lead too far astray if we think of decoders as being associated with a particular neuron. However, we must remember that neurons do what they do (i.e., encode what they encode) because of their relations to other neurons, and not solely in virtue of their intrinsic properties.

4.3.2 Neuron pairs

In order to successfully find the decoders in this picture of neural representation, it helps to make things a bit more complicated. That is, it helps to consider two neurons at a time instead of just one. This may seem strange, as the fundamental unit of signal processing in the nervous system is often thought to be the single neuron. However, given the considerations of the previous chapters, it is clear that we must understand representation in *populations* of neurons in order to understand the behavior of neural *systems*. The simplest possible population is a pair of neurons. So, while being simple, understanding temporal representation in pairs of neurons constitutes a significant first step towards understanding temporal representation in populations. And, importantly, some of the more troublesome properties of single neurons (e.g., their highly nonlinear responses) become far less troublesome when we consider neuron pairs.

This might come as no surprise to communications engineers. It has been known since the 1920s that superior amplifiers can be built by using complementary pairs of tubes or transistors. This kind of amplifier is commonly known as a push-pull amplifier (see figure 4.5). These amplifiers are significantly more efficient than single element amplifiers that achieve the same linearity (Sawdai and Pavlidis 1999). In general, the push-pull topology provides higher efficiency, higher power output, greater symmetry in the output signal, and a wider range of linear responses compared to a single element circuit.

For our purposes, the increase in linearity is of the most interest. The reason that the push-pull arrangement provides this benefit is that the nonlinearities in the elements are used to offset one another near the midpoint of the range of operation (usually 0). In effect, each element is used to code half of the input signal. However, the halves overlap such that their sum in the overlapping region serves to linearize the total response, despite each element having significant nonlinearities in the same region (see figure 4.5). Thus, the total range over which the response of the system is reasonably linear is greater than could be achieved with two independent (i.e., averaged) elements.

So what does this have to do with neural coding? There is evidence that nervous systems often use a similar strategy. It is not unusual to find ‘on’ and ‘off’ cells in a population that codes a single physical quantity. Most obviously, ‘on’ and ‘off’ retinal ganglion and lateral geniculate nucleus cells are used to code light intensities (Hubel and Wiesel 1962). In the neural integrator circuit, the neurons can be divided into two groups; those whose response increases during leftward movements and those whose response increases during rightward movements (Fukushima et al. 1992). The same is true of velocity sensitive neurons in the head-direction system of mammals (Redish 1999) and angular acceleration sensitive cells in the vestibular system (Wilson and Jones 1979). In motor cortex, cells are broadly tuned and have a wide range of preferred directions

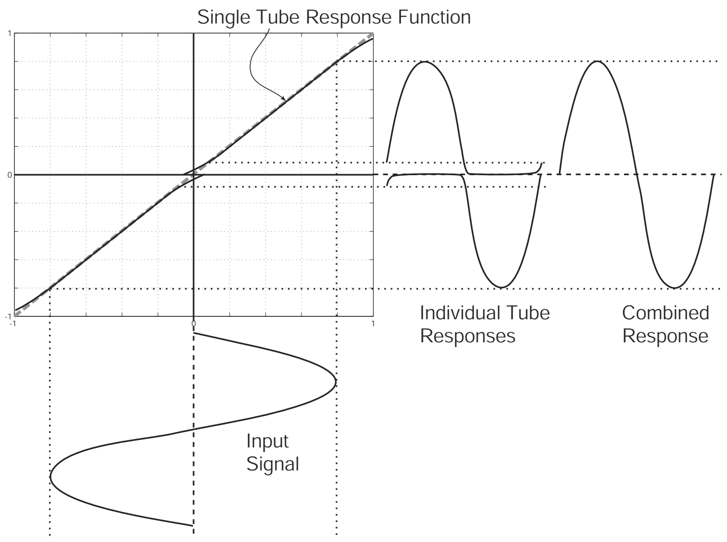


Figure 4.5
A push-pull amplifier. The input signal has nonlinear distortions introduced by each of the elements, but when the responses of the elements are summed those distortions cancel. The result is a highly linear reproduction of the original input signal over a wide range, despite nonlinear elements.

(Georgopoulos et al. 1993). This is simply a generalization to higher dimensions of the same kind of strategy. That is, for most one-dimensional slices that go through zero in this two-dimensional space, we observe the same kinds of opponency cells we see in these other systems; that is, some increase firing for movement in one direction and some increase firing for movement in the opposite direction.[6] In many cases, then, cells have an opponency relation analogous to that of push-pull amplifiers. And we think the reason might be generally the same—this is a more effective means of constructing a system with a broader range of linear responses.[7] In appendix B.1 we explain why opponency provides more linear responses.

There is another reason that opponency is important. This second reason is unique to neurons. It is crucial for neurons to have this kind of arrangement because they do not encode their input with a *continuous* output signal. Amplifiers have continuous output. Neurons, in contrast, have an 'all-or-none' output in the form of a voltage spike. In order to

6 This is the result of the large number of neurons randomly distributed over the hypersphere in the relevant space. We do not mean to insinuate that neurons are actually *paired* (i.e., for each neuron there will always be a neuron tuned to the opposite direction).

7 Neural systems are often quite linear over a broad range. For instance, in some vestibular systems there is only a 10% deviation from linearity over a 16 fold change in stimulus intensity (Wilson and Jones 1979, p. 99).

understand the consequences of this difference, consider a simple code in a perfectly linear neuron: the more intense the stimulus, the higher the firing frequency, with zero intensity equal to zero firing frequency. Significant problems arise with this kind of code for low stimulus intensities. At low intensities, the neuron fires at low frequencies so it will take a *very long time* to determine what the frequency is. A mechanism decoding the output would have to wait until it sees enough spikes to guess what the firing frequency is. In other words, the lower the stimulus intensity, the longer it takes for the neuron to transmit information about stimulus intensity. This is not acceptable for a system attempting to operate in a rapidly changing environment.

However, this problem is solved with an opponency arrangement. By adding a second neuron that increases firing with decreasing stimulus intensity, we have guaranteed that the average firing rate (so the average time to transmit information about stimulus intensity) is constant. If we are not too concerned about having a constant average firing rate, so long as it is above a certain value, we can increase the slope of our encoding neurons. This gives us a better signal-to-noise ratio over the same range since a smaller change in stimulus intensity leads to a large change in the output firing rate. A similar argument holds in the case of nonlinear neurons like the LIF neuron.

In sum, this on-off opponency arrangement provides a temporally efficient and more linear encoding than is evident from examining single neurons. Since we are interested in constructing linear decoders for neural encodings, it is likely that the more linear response from a pair of neurons will serve our purposes better.[8] As well, since we are ultimately interested in understanding temporal codes in neural populations, it makes sense to start with one of the simplest populations—just two neurons. Given these considerations, we take it that neuron *pairs* are at least as fundamental for understanding the properties of neural representation as individual neurons.

4.3.3 Representing time dependent signals with spikes

Taking opponent neurons as the fundamental unit of signal processing in neural systems, figure 4.6 shows what we take to be the basic encoding and decoding steps that are needed to characterize neural representation. As mentioned earlier, the encoding process is determined by the neural model we have chosen (LIF in our case) and so we are largely concerned with the task of determining the appropriate decoder. This is much like the problem we originally faced in chapter 2, except that we are interested in understanding the representation of a continuously changing function of time, rather than a static scalar or vector.

8 Notably, Rieke et al. (1997) also find optimal decoders by examining two neurons, although they do not make much of the point.

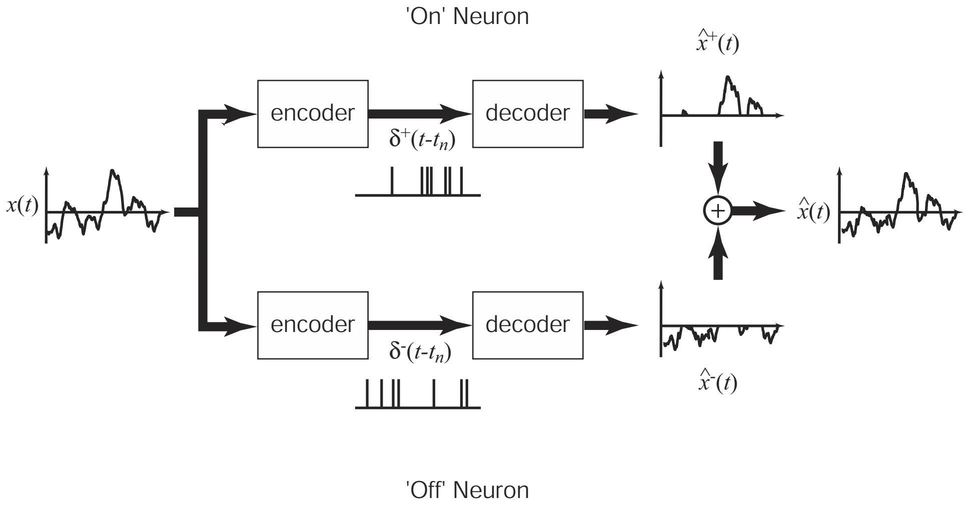


Figure 4.6
Encoding and decoding with an on-off neuron pair. Just like the push-pull amplifier, each neuron encodes part of the signal, and their decoded sum provides a good representation of the whole signal.

As a result, we begin by making the same well-supported assumption about the linearity of the decoding as we did for the population code (see Rieke et al. 1997 for an excellent review, especially pp. 76–98 and 168–172; see also Bialek et al. 1991; Bialek and Zee 1990). In particular, we presume that we can decode the neural spike train by taking a (continuous, time-shifted) *sum* of some decoding function, $h(t-t')$, weighted by the encoded signal (see figure 4.7 for an intuitive picture of linear decoding). We know, given the LIF model, that the signal, $x(t)$, is encoded as a series of delta functions, which we can write as $\sum_n \delta(t-t_n)$. Thus, our decoding of the encoded function, $x(t)$, can be expressed as

$$\hat{x}(t) = \int_0^T h(t-t') \sum_n \delta(t'-t_n) dt'. \tag{4.12}$$

This kind of expression is known as a convolution integral. This equation says that our best guess as to what $x(t)$ is, can be found by adding up the occurrences of a time-shifted decoding function (analogous to the decoding weights, ϕ_i, in the population code). Another way of describing this decoding process is to say that we are assuming that there is some linear system that can decode the spike train, $\delta(t-t_n)$, and successfully reconstruct the original input signal that was presented to the neuron. Our job, then, is to find this linear system.

Because a linear system can be completely characterized by its impulse response, $h(t)$, finding the linear system is the same as finding, $h(t)$. In particular, the output from any linear system, $y(t)$, can be written as a convolution integral of the input, $f(t)$, and the system's impulse response, $h(t)$:

$$y(t) = \int h(t - t')f(t')dt'.$$

This is clearly analogous to equation (4.12) above. So what we mean when we say that we assume a linear decoding, is that we can find an impulse response, $h(t)$, that reconstructs the original signal. Because the function $f(t)$ in equation (4.12) is a series of delta functions, we can simplify the integral by evaluating it analytically to give

$$\hat{x}(t) = \sum_n h(t - t_n). \tag{4.13}$$

Note that calling $h(t)$ a linear decoder (or linear filter) does not mean that we assume that we can find $h(t)$ by a linear analysis of the neuron's response. In fact, we have shown explicitly elsewhere that a linear analysis will not allow us to find this decoding filter (Anderson et al. 2000). In some ways, this should not be surprising given how nonlinear neurons really are.

Now that we have a characterization of the general nature of temporal representation, we need to consider if there are any special properties of the signals themselves. This is because any constraints on what is actually represented by nervous systems can help us find the relevant decoders. To this end, it is important to emphasize that we do not want a linear decoder that will decode one particular signal. Rather, we want a linear decoder that is useful for decoding all signals of interest; i.e., we want to find a linear decoder that is good for an *ensemble* of signals, just as we earlier wanted decoders for an ensemble of functions. There is good evidence that real neurons are optimized for passing information about particular ensembles—namely, *natural* signals (de ruyter van Steveninck et al. 1997). It would be very surprising if biological relevance *had not* played a role in constraining the representational characteristics of neural systems, so this is just what we might expect.[9]

One general property of naturally occurring signals that we assume is that they are not unique in time. This property variously goes by the name of ‘stationarity’, ‘ergodicity’, or ‘time independence’. By definition, if a signal (or process or system) is stationary, then the statistics of any sample of the signal does not depend on the temporal placement of that sample. So, for example, signals consisting of natural images are stationary because they

9 This makes it rather unfortunate that many experiments in neurophysiology use very unnatural signals (e.g., step functions) to characterize neuron responses. Using signals drawn from a more natural ensemble often shows the neurons to be more adept at passing signals that was previously thought (Mainen and Sejnowksi 1995).

have the same kinds of statistics *now* as they did *yesterday*, and as they will *tomorrow*. In general, sensory signals do not have clearly defined beginnings and endings, so they can be usefully treated as stationary.

Being able to ascribe the property of stationarity to natural signals is important because it allows us to define the relevant ensembles of signals using statistically independent coefficients. In particular, we can write the ensemble that we take to be represented as a Fourier series:[10]

$$x(t;\mathbf{A}) = \sum_{n=-(N-1)/2}^{(N-1)/2} A(\omega_n)e^{i\omega_n t}, \tag{4.14}$$

where $\mathbf{A}$ is the vector of frequency amplitudes and ω_n is $n\Delta_\omega$ where Δ_ω is the frequency step size. The series of frequency components, ω_n, consists of N complex frequency components where $A(\omega_n) = A^*(-\omega_n)$, and N is odd. It is important to note that we always assume a finite N. The reason we do this is because we are always interested in characterizing systems that operate in the real world, under tight implementational constraints. Under such conditions, there will always be some finite time, T, that is the maximum length of a signal in our ensemble. Given the duality between the length of time of a signal and its resolvability in the frequency domain, we know that $\Delta_\omega = 2\pi/T$. This guarantees that we can faithfully represent our signal with a finite number of frequency components.

As before, choosing a particular set of amplitude coefficients (i.e., a specific $\mathbf{A}$ vector) picks out one signal in this ensemble (see section 3.2). So, in order to properly define an *ensemble* of signals, we must specify what values the vector $\mathbf{A}$ can assume. Again, we can do this by specifying the probability distribution of $\mathbf{A}$. Assuming that each component of $\mathbf{A}$ is an independent Gaussian random variable whose variance is given by the power spectrum $\mathcal{P}(\omega_n)$, we can write a Gaussian distribution[11]

$$\rho(A(\omega_n)) = \frac{1}{\sqrt{2\pi\mathcal{P}(\omega_n)}}e^{-A(\omega_n)^2/2\mathcal{P}(\omega_n)}. \tag{4.15}$$

Given stationarity, we know that these distributions define the distribution of all of the amplitudes, $\mathbf{A}$, namely,

$$\rho(\mathbf{A}) = \prod_{n=0}^{(N-1)/2} \rho(A(\omega_n)), \tag{4.16}$$

and $A(\omega_n) = A^*(-\omega_n)$.

10 This is equivalent to our definition in equation (3.3), but is in a more convenient form for the subsequent analysis.

11 Notably, the experiments that we later compare our results to draw signals from signal ensembles with the same structure (Bialek et al. 1991).

Now that we have a means of describing the relevant signal ensembles, we can return to the problem of finding an optimal linear filter. Recall that we are interested in analyzing pairs of neurons, and take the system of interest to be comprised of two complimentary neurons, whose response profiles mirror one another (see figure 4.6). Appropriately extending equation (4.11), we can write the encoding of the *pair* of neurons of some signal (picked out by $\mathbf{A}$) into M spikes as

$$\begin{aligned} R(t;\mathbf{A}) &= \sum_{k}^{M_{on}} \delta(t - t_k^+(\mathbf{A})) - \sum_{l}^{M_{off}} \delta(t - t_l^-(\mathbf{A})) \\ &= \sum_{i=1}^{2}\sum_{k=1}^{M} \phi_i \delta(t - t_{ik}(\mathbf{A})), \end{aligned}$$

where $\phi_i = 1$ for the 'on' spikes and -1 for the 'off' spikes as before. We use $R(t;\mathbf{A})$ to indicate the *response* of the pair of neurons; that is, all of the spikes produced by both neurons given the input (this is *not* the same as the estimate of the signal in (4.13)). In order to find the optimal linear filter for decoding this signal, we want to minimize the mean square error, as we did in (2.4):

$$\begin{aligned} E &= \left\langle [x(t;\mathbf{A}) - h(t) * R(t;\mathbf{A})]^2 \right\rangle_{t,\mathbf{A}} \qquad (4.17) \\ &= \left\langle \left[x(t;\mathbf{A}) - \sum_{i,k} h(t - t_{ik}(\mathbf{A})) \right]^2 \right\rangle_{t,\mathbf{A}}, \end{aligned}$$

where $*$ indicates convolution. Rather than attempting to minimize this error in the time domain, we can use the Fourier transform to express the error in the frequency domain. This has the immediate advantage of turning the convolution in (4.17) into a multiplication. As a result, we can write the error for each frequency channel as (see appendix B.3)

$$E(\omega_n) = \left\langle \frac{1}{2\pi} \left| A(\omega_n) - h(\omega_n) R(\omega_n;\mathbf{A}) \right|^2 \right\rangle_{\mathbf{A}}. \qquad (4.18)$$

Unfortunately, this is a very high-dimensional integral. Specifically, it will have as many dimensions as there are frequency components needed to define the ensemble of signals. It is very difficult to directly evaluate this kind of integral, so instead we perform a Monte Carlo estimate of its solution. In other words, we generate a large number of actual signals, $x(t;\mathbf{A}^\alpha)$ (where α indexes a particular set of values for our amplitudes, $\mathbf{A}$), find the spike train responses from the pair of neurons, $R(t;\mathbf{A}^\alpha)$, and then determine the linear filter, $h(t)$, that minimizes the average error between the linear estimate of the signal and the original signal (see appendix B.3 for details).

This analysis, though more intuitive in the time domain, is more efficient in the frequency domain. This is because the number of degrees of freedom (i.e., the number of free parameters used to estimate our linear filter) is much smaller in the frequency domain, even though the results are equivalent. To see why this is the case, consider the following example of finding the optimal filter for the retinal ganglion parvo cells. The peak frequency response of these cells is at about 30 Hz (Van Essen and Anderson 1995). Assuming that the membrane leakage time constant is approximately 20 ms (and thus that effects of inputs about 100 ms in the past can be ignored (i.e., $5\tau_{RC}$)), we can determine the number of frequency components needed to characterize the filter. As mentioned earlier,

$$\begin{aligned}\Delta_\omega &= \frac{2\pi}{T} \\ &= \frac{2\pi}{100\,\text{ms}} \\ &= 10\,\text{Hz}.\end{aligned}$$

So the number of components needed is

$$N = \frac{30}{10} = 3.$$

Each of these components is complex, so the problem of describing the linear filter $h(\omega)$ has approximately 6 degrees of freedom. In contrast, if we were to try and estimate the filter in the time domain, where sample times necessary for good approximations are on the order of .1 ms or less, we would have approximately 1000 points in $h(t)$ to estimate (i.e., almost three orders of magnitude more degrees of freedom). Of course we could put various constraints on how these parameters were estimated in order to reduce the complexity of the problem. However, in the frequency domain, we have a built-in, unbiased means of reasonably limiting the number of parameters that need to be estimated to find the linear filter.

Theoretically speaking, in order to estimate this filter, we need only run many short trials (of size $T = 5\tau_{rc}$ or so). However, large numbers of short trials is expensive both experimentally and numerically, and startup transients are likely to influence the results of such experiments. So, as also done by Rieke et al. (1997), we have developed a means of 'windowing' signals of several seconds so as to analyze these longer experiments as if they were many shorter experiments. As discussed in more detail in appendix B.3, there are several advantages to our method over existing ones. Most importantly, our windowing method makes more efficient use of the data from short trials, decreasing the likelihood of error from transients. This has important practical consequences since many experimental preparations cannot be maintained for the long run times needed to employ previous methods.[12]

As shown in appendix B.3, minimizing the windowed version of (4.18) gives

$$h(\omega_n) = \frac{\langle A(\omega_n) R^*(\omega_n; \mathbf{A})\rangle_{\mathbf{A}}}{\left\langle |R(\omega_n; \mathbf{A})|^2 \right\rangle_{\mathbf{A}}}, \tag{4.19}$$

where the convolution with the window is included in the averaging, $\langle \cdot \rangle_{\mathbf{A}}$, to emphasize that this is a means of approximating the average over the stochastic variables, $\mathbf{A}$. From the numerator of (4.19) we can see that the filter will have a low amplitude if there is not much correlated power between the input amplitudes and the spike train amplitudes. The denominator tells us that this is also true if the average power of the spike trains is large at a particular frequency. This can occur if the neurons have a high mean background firing rate that generates power that is uncorrelated with the input signal.

Those familiar with past analyses of this kind may be wary of certain limitations of this kind of filter. In particular, Rieke et al. (1997) note that linear filtering is far more likely to work if the correlation time of the signals being encoded are significantly shorter than the mean interspike interval (ISI) (p. 86). This is because having a short correlation time relative to mean ISI ensures that individual spikes are providing independent information about the signal. If this was not the case, then there may be significant interactions between neighboring spikes, i.e., both spikes may be providing information about the same parts of the signal. However, there is nothing in the preceding analysis that makes assumptions about the relation between the mean ISI and the correlation time of the input signal. So, it is merely a *possibility* that linear decoding will not work in the case where correlation times are long compared to mean ISI. In fact, as we show in section 4.4.1, linear decoding performs very well even for long correlation times.

Fortunately, we are now in a position to quantify precisely what it means for our decoder to work well. To determine how accurately our optimal linear filtering approximates the original signal, we can substitute (4.19) back into (4.18) to obtain the residual error, E_r:

$$E_r(\omega_n) = \left\langle |A(\omega_n)|^2 \right\rangle_{\mathbf{A}} - \frac{|\langle A(\omega_n) R^*(\omega_n; \mathbf{A})\rangle_{\mathbf{A}}|^2}{\left\langle |R(\omega_n; \mathbf{A})|^2 \right\rangle_{\mathbf{A}}}.$$

This error will only be zero when there are no interharmonic distortions that generate

12 The method used by (Rieke et al. 1997) was employed in the context of experimental run times as long as 2 days.

spurious power at frequency ω_n. Nonlinear processes, even those as simple as the LIF neuron, generally do suffer from these kinds of distortions (Anderson et al. 2000). The residual error is a good measure of how well our linear decoding works, and is analogous to what we called 'static error' in chapter 2.

We have now shown how to define the representation of time dependent signals in pairs of spiking neurons. Specifically, we have defined the encoding

$$R(t; \mathbf{A}) = \sum_{i,k}^{M} \phi_i \delta(t - t_{ik}(\mathbf{A})), \tag{4.20}$$

and decoding

$$\hat{x}(t) = h(t) * R(t; \mathbf{A}), \tag{4.21}$$

where

$$h(\omega_n) = \frac{\langle A(\omega_n) R^*(\omega_n; \mathbf{A}) \rangle_{\mathbf{A}}}{\left\langle |R(\omega_n; \mathbf{A})|^2 \right\rangle_{\mathbf{A}}}. \tag{4.22}$$

4.3.4 Discussion

Now that we know how to find the optimal temporal filter, and have a means of determining how good it is, let us consider the linear decoder, $h(t)$, in more detail. As just noted, our estimate of the signal that is encoded by the neurons into the spike train is given by (4.21). Substituting (4.20) into (4.21), and recalling the result in (4.13), we can write

$$\hat{x}(t) = \sum_{i,k}^{M} \phi_i \delta(t - t_{ik}) * h(t) \tag{4.23}$$

$$= \sum_{i,k}^{M} \phi_i h(t - t_{ik}). \tag{4.24}$$

In essence, this equation says that our estimate is found by putting a waveform in the shape of the linear filter at each spike time and summing the results (see figure 4.7).

This estimate is closely related to the population estimates we used in the previous chapters. In particular, we could re-write (4.24) as

$$\hat{x}(t; \mathbf{A}) = \sum_{i}^{M} a_i(x(t; \mathbf{A})) \phi_i(t), \tag{4.25}$$

where M is the number of time-steps we have divided the signal into, the a_i are 1, 0, or -1, depending whether or not a neuron emitted a spike, and the $\phi_i(t)$ are all time-shifted

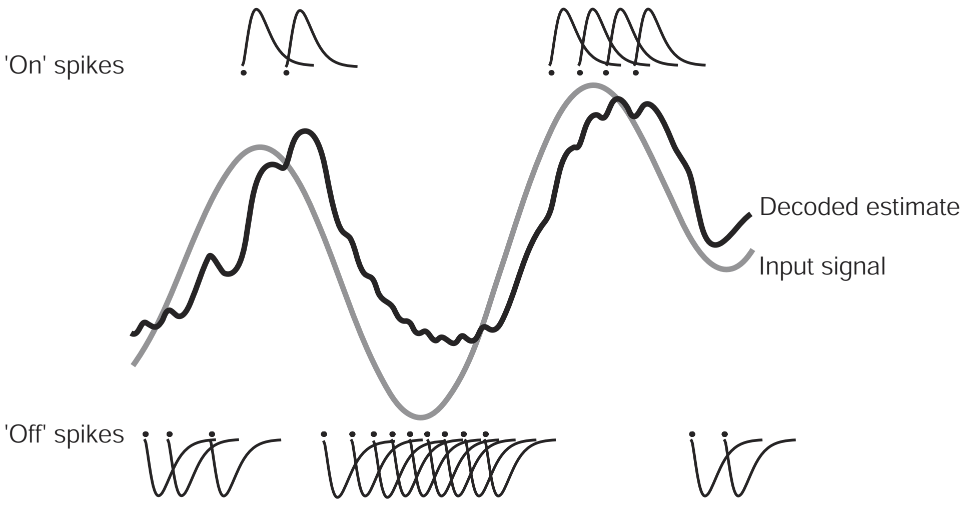


Figure 4.7
An example of linear filtering. The input signal, $x(t)$, is fed into the somas of a pair of on-off neurons which encode the signal into 'on' and 'off' spikes. To get an estimate, $\hat{x}(t)$, of that signal, we can linearly filter those spike trains by effectively placing the filter at the time of occurrence of each spike and summing the result. When the on and off neurons are symmetrical, their respective filters will be 'mirror images', as shown in the figure.

versions of $h(t)$ (i.e., $h(t - t_i)$). This notation is unusual because the neuron activity in the population code is mapped onto activity (the presence of a spike) at some time t_i, so, as in the case of the population code, for each 'active element' we need one decoder. This results in a large number, M, of temporal decoders, all of which are essentially the same. While awkward, this notation shows exactly how the temporal decoders and the population decoders perform the same function—both serve to translate an activity back into the original domain that was encoded (i.e., either x for the population code or $x(t)$ for the temporal code). As an aside, it is interesting to note that most of the coefficients, a_i, in (4.25) will be zero for any particular signal. In this sense, the representation is a 'sparse' representation. It makes sense that the neural code is sparse, as this results in more efficient coding (Olshausen 2000) and a better use of available energy (Baddeley 1996).

Because $x(t)$ is a *function*, its representation is much like the representation of $x(\nu)$ that we discussed in section 3.2. Looking again at equation (3.6) we see that it is indeed very similar to (4.25). However, there is also an important difference between population and temporal encoding that becomes evident from this comparison. Namely, there is no temporal encoding function in the sense of $\tilde{\phi}_i(\nu)$. This is because the temporal encoding is defined completely by the intrinsic properties of the neuron, which are captured by $G_i\left[\cdot\right]$. This difference means that it is much more difficult to derive the same kinds of analytical results for understanding temporal coding as we do for population coding (see section 7.3).

Nevertheless, it proves to be useful that both temporal and population codes in neurons can be characterized using linear coding, since it allows us to unify these two kinds of coding (as we discuss in section 5.1). Before doing so, however, let us consider a number of examples of how to use this characterization of temporal coding to measure the efficiency of information transmission in neural models. We begin with the simple LIF neuron and progress to more complex models. Perhaps the most important lesson to be learned from these examples is that the basic LIF model has just about the same information transmission characteristics as its more complex counterparts. And, both kinds of models perform comparably to real neurons.

4.4 Information transmission in LIF neurons

While we now have a method for characterizing temporal representation, there are two related issues that we have so far ignored or only partially addressed. First, we suggested that this analysis applied to spike trains with both short and long-correlation times, unifying rate and timing codes—we must show that this is the case. And second, we did not discuss how these optimal decoders (or 'filters') relate to their biophysical counterparts, the postsynaptic currents (PSCs). Although these subsequent discussions rely solely on applying our method to the LIF model, in section 4.4.2 we relate these results to the information characteristics of real neurons. As well, in section 4.5 we compare LIF results with the same measures in more complex neuron models. These comparisons show that it is reasonable to use simple LIF neurons in simulations of neural information processing.

4.4.1 Finding optimal decoders in LIF neurons

It is important that nothing in our discussion of optimal linear decoders depends on the nature of the encoder. Of course, choosing a specific encoder greatly affects the particular optimal decoder that we find, but the methods and analyses we discuss are independent of the encoder. In this section we present results using LIF neurons as our encoder. For these simulations, and all similar ones in this chapter, we assume that our signal ensemble is band-limited Gaussian white noise (an assumption shared with the experiments on real neurons that we compare our results to). As well, the encoding in each case is done by a symmetrical pair of 'on' and 'off' neurons. This ensures that the optimal filter is the same for both neurons (as we have assumed to this point, see (4.24)). The LIF neurons we use for our analyses have a background firing rate of 40 Hz, a refractory period of 2 ms, and an RC time constant of 20 ms.

In figure 4.8, the optimal linear decoder is shown in both the frequency and time domains. The decoder displayed here was found using the sliding window analysis described

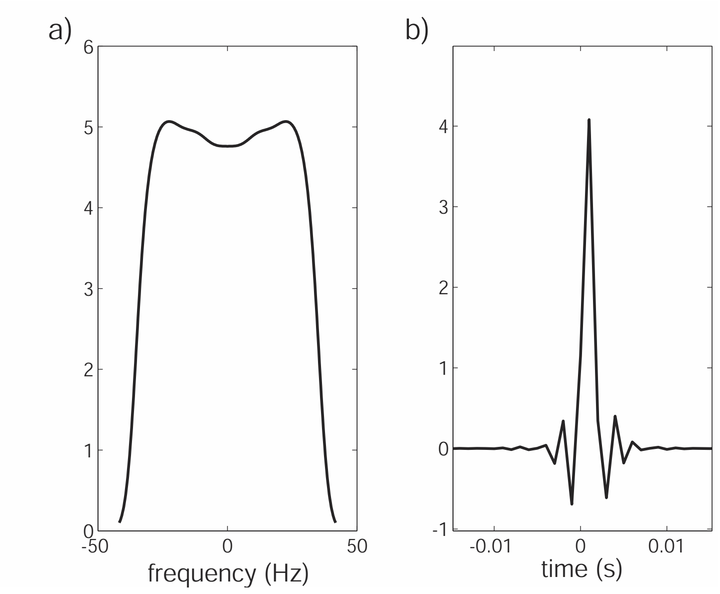


Figure 4.8
An optimal linear decoder for a pair of LIF neurons in a) the frequency domain and b) the time domain.

in section 4.3.3 and appendix B.3 on the signal shown in figure 4.9a. Using four seconds of data, we are able to find a filter that very successfully decodes the original input signal, as well as other signals that were randomly drawn from the same ensemble (as shown in figures 4.9b and 4.9c).

We can see from figure 4.10 that the encoding of the original signal via the LIF neuron into a spike train introduces spurious power, especially at high frequencies. This is not surprising given the ‘spiky’ nature of the encoding. The power at frequencies that are actually in the signal, however, are also well-preserved. Thus, one of the main functions of the decoding filter is to remove these spurious high frequency components while not otherwise altering the spectrum; i.e., it acts as a low pass filter. As can be seen in figure 4.8, the filter does indeed resemble a low pass filter. Thus it is localized in time (figure 4.8b) and nearly flat for frequencies below some cutoff (50 Hz in this case) in the frequency domain (figure 4.8a). The precise width of the optimal filter in the time domain depends on the kinds of signals that were used to find it. For higher frequency signals, it will be thinner, increasing the cutoff frequency of the low-pass filter. Conversely, for a lower frequency set of signals, it will be wider in the time domain and have a lower cutoff frequency. It is also worth noting at this point that the filter somewhat resembles a dendritic PSC. Specifically,

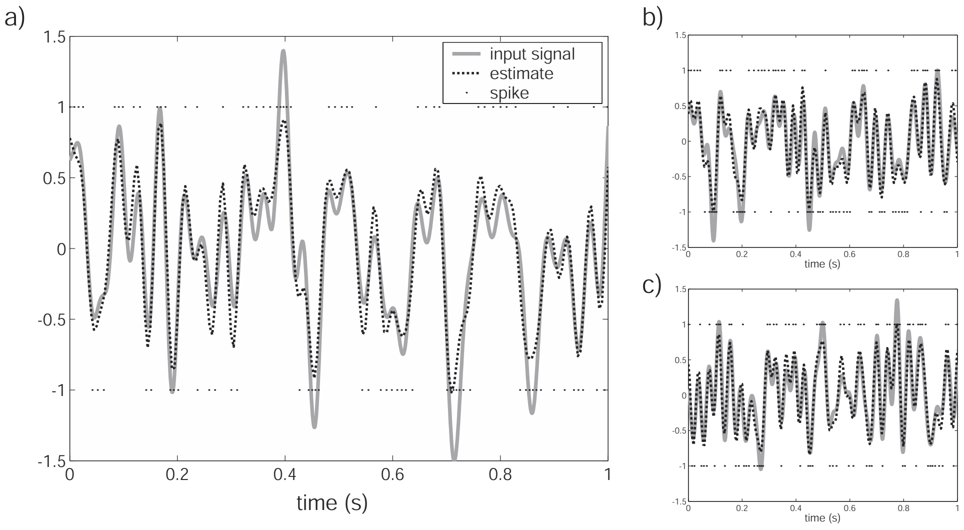


Figure 4.9
a) Reconstruction of the randomly generated signal used to find the optimal filter in figure 4.8 (RMSE = 0.153). Figures b) and c) show reconstructions of randomly generated signals from the same distribution as the signal in a). These were reconstructed using the filter found using the signal in a). RMSE = 0.149 for b) and RMSE = 0.142 for c).

it is localized in time and has a large, rapidly decaying initial response. However, it is clearly non-causal as it is defined over both negative and positive parts of the time axis.

As mentioned earlier, there is some concern about being able to use such decoders for signals with long correlation times (see Rieke et al. 1997, p. 86). However, in figure 4.11, it is evident that the optimal filter found for the signal with a short correlation time (figure 4.9) works equally well for a signal with a long correlation time. This is true in general. A thousand trials of both slow (i.e., frequency channels between 0 and 4 Hz) signals with long correlation times and fast signals (i.e., frequency channels between 0 and 30 Hz) with short correlation times results in a mean RMSE for the signals with long correlation times of 0.123 and mean RMSE for those with short correlation times of 0.147. This is true *despite* finding and using the optimal filter for the *fast* signals only (i.e., short correlation times). Thus, the same filter not only works for signals with long correlation times, it actually works better than for signals with low correlation times. This is to be expected, since RC circuits (like the LIF neuron) function as low pass filters, and thus are better able to preserve lower frequencies. However, as shown in figure 4.12, finding the optimal filter using a long correlation time ensemble results in an even better estimate of such signals, as expected.

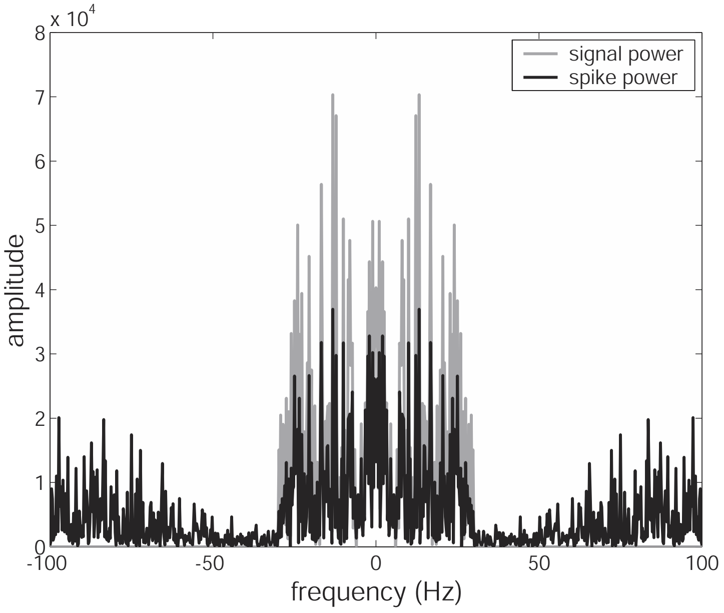


Figure 4.10
Comparisons of the power of the original signal (depicted in figure 4.9a) and the spike train which encodes the signal (also depicted in figure 4.9a). Note that all power in the spikes signal above 30 Hz is not in the original signal.

By comparing figures 4.9 and 4.11, we can see how the same neuron is using something more like a ‘timing’ code in the first case, and something more like a ‘rate’ code in the second case (look especially at the spike times, indicated by small dots). By comparing figures 4.11 and 4.12, we can see that both seem to use a kind of rate code. However, in figure 4.12, the optimal filter is much wider in the time domain because it was constructed from a set of signals with more similar statistics. If the optimal filter gets wider (while remaining equally smooth), then the precise timing of spikes becomes less important for decoding the signal, thus acting more like a rate code. We have verified this by introducing random jitter in the spike times that encode either the fast (short correlation time) or slow (long correlation time) signal. As expected, the RMS error increases significantly more when decoding the jittered spike trains for fast signals (using the ‘fast’ optimal filter) than for slow ones (using the ‘slow’ optimal filter), despite the fact that the spike trains have approximately the same total number of spikes. As well, the RMS error increases significantly more for the slow signals decoded with the ‘fast’ optimal filter. Thus, the average firing rate is more informative about the slow signals, as long as we know it is drawn from a distribution with long correlation times.

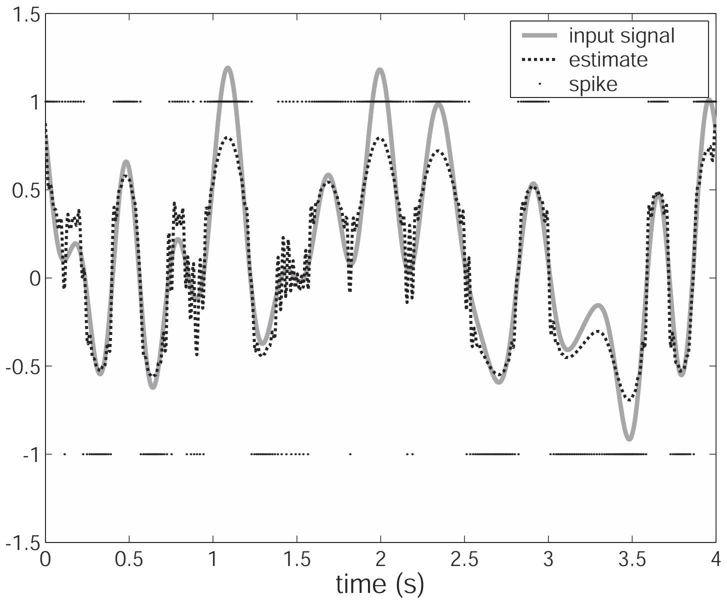


Figure 4.11
Reconstruction of a randomly generated signal from a distribution guaranteeing a low frequency signal, using the optimal filter from figure 4.8. This filter was found using a signal distribution with high frequencies as well. Nevertheless, this reconstruction works very well (RMSE = 0.143). Note that the time scale has been lengthened.

So, in conclusion, we have found a means for determining an optimal decoder that works well under a variety of conditions. Not surprisingly, the optimal filter generally works better for a signal drawn from an ensemble with characteristics similar to those used to find the optimal filter. Nevertheless, filters found from ensembles that generally result in short correlation times work very well for signals with longer correlation times. So, this method of finding optimal filters bridges the gap between timing and rate codes because it works effectively in either realm.

4.4.2 Information transmission

In order to compare the behavior of LIF information processing to that of natural neural systems, let us briefly review the information characteristics observed for real neurons. Chief amongst the measures of information processing is information transmission rates, measured in bits per spike or bits per second. These measures are surprisingly consistent across many different neural systems. In the cricket cercal system, which measures wind velocity, information rates of between about 150 (Roddey and Jacobs 1996) and 300

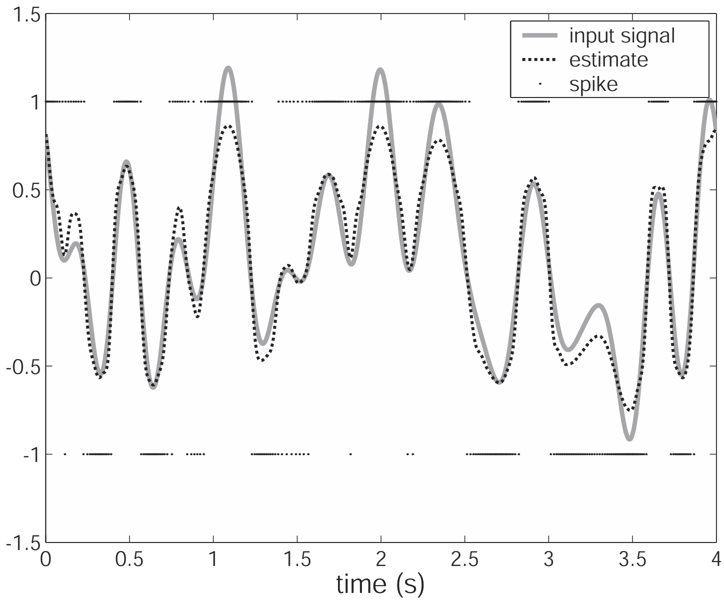


Figure 4.12
Reconstruction of a the same signal from figure 4.11 using an optimal filter based on a long correlation time (i.e., low-frequency) ensemble (RMSE = 0.122). The error is slightly improved, and the high-frequency aspects of the reconstruction in figure 4.11 are removed.

(Bialek and Rieke 1992) bits per second have been reported.[13] These rates are equivalent to between 1.1 and 3 bits per spike. In the bullfrog sacculus, which senses ground-borne vibrations, Bialek et al. (1991) report transmission rates of about 3 bits per spike. As well, Bialek et al. (1991) show that motion-selective H1 neurons in the blowfly visual system carry about 3 bits per spike. In salamander retina, recent results suggest that information is transmitted at a rate of about 3.4 bits/spike (Berry II and Meister in press). In primate visual area V5, information transmission rates of 1–2 bits per spike have been observed (Buracas et al. 1998). The highest transmission rates we have seen reported are for the bullfrog auditory neurons, which reach rates as high as 7.8 bits per spike (Rieke et al. 1997, p. 185). Notably, these rates were only achieved for stimuli with frequency spectra of naturally occurring bullfrog calls. Broadband stimuli had transmission rates of about 1.4 bits per spike. In sum, natural sensory systems are generally able to encode stimuli with between about 1 and 3 bits of information per spike (see also Rieke et al. (1997) for a review).

13 Although Miller et al. (1991) calculate the rate to be about 40 bits per second, they used a 100 ms binned rate code to calculate information.

These are impressively high transmission rates that approach the optimal possible coding efficiencies (de ruyter van Steveninck and Bialek 1988; Rieke et al. 1997). In the frog sacculus, the cricket cercal system, the bullfrog auditory system, and the electric fish electrosensory system, the codes are between 20 and 60% efficient (Wessel et al. 1996; Rieke et al. 1997, p. 180). And, efficiency significantly increases when stimuli are restricted to be more like naturally occurring stimuli of interest to the organism (Rieke et al. 1997, p. 185).[14] All of these measures of information transmission performance in natural systems are found by closely related methods. The researchers use opponent neurons, assume stationarity, and find optimal linear filters, just as we have done. What is of interest to us, as modelers, is to see how these measures in neurobiological systems compare to those for model neurons. So, in the remainder of this section, we perform a similar analysis for the LIF neuron.

To begin, we must realize that model neurons are entirely deterministic. Thus, unlike real neurons, they can be run under conditions of no noise. Technically, then, information transmission rates can be unlimited. However, we can still find an analogous information measure because we linearly decode a nonlinear system. In particular, we can compare the total variance to the variance that is unexplained by our linear decoding procedure. Usually noise is the source of unexplained variance, but in this case, it is the result of our linear filtering.

We can derive and express the information transmission per frequency channel as (see appendix B.4 for the derivation)

$$\mathrm{Info}(\omega_n) = \frac{1}{2}\log_2\left[\frac{\left\langle |A(\omega_n)|^2\right\rangle_{\mathbf{A}}}{\left\langle |A(\omega_n)|^2\right\rangle_{\mathbf{A}} - \frac{\left|\langle A(\omega_n)R^*(\omega_n;\mathbf{A})\rangle_{\mathbf{A}}\right|^2}{\left\langle |R(\omega_n;\mathbf{A})|^2\right\rangle_{\mathbf{A}}}}\right] \quad (4.26)$$

$$= \frac{1}{2}\log_2\left[\frac{\left\langle |R(\omega_n;\mathbf{A})|^2\right\rangle_{\mathbf{A}}}{\left\langle |R(\omega_n;\mathbf{A})|^2\right\rangle_{\mathbf{A}} - \frac{\left|\langle A(\omega_n)R^*(\omega_n;\mathbf{A})\rangle_{\mathbf{A}}\right|^2}{\left\langle |A(\omega_n)|^2\right\rangle_{\mathbf{A}}}}\right]. \quad (4.27)$$

In (4.27), the numerator expresses the total variance at the output, while the denominator expresses the variance that is not explained by our assumption that the output is linearly related to the input. Equation (4.26) has a similar interpretation, though in terms of the input signal. Because we assume linear decoding, these equations express only a lower bound on the amount of information that could be transfered through this system (figure

14 Also note, that the estimation of information transmission rates using this method places a *lower* bound on the amount of information transmitted by the code. There are reasons to think this bound generally *under*estimates the actual amount of information transmitted (Stevens and Zador 1996). Thus, efficiencies are likely even higher.

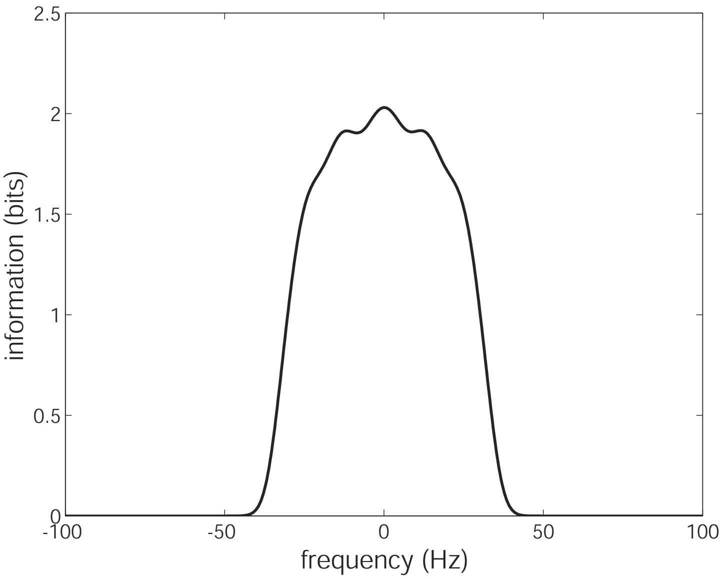


Figure 4.13
Information transmission for the example in section 4.4.1 (see figures 4.9, 4.10, and 4.11). This shows the amount of information decodable from the original signal at each frequency band using the optimal filter.

4.13 depicts the application of these equations to the example in the section 4.4). The resulting information transmission rates for this example are 1.24 bits per spike or 114 bits per second. Both of these measures are comparable to those reported earlier for real neurons.

If we had a more sophisticated, nonlinear decoding technique, we could get a better estimate of the input signal given the output, and thus increase the information transmission. However, Rieke et al. (1997) argue that nonlinear decoding only provides about a 5% improvement in information transmission in real neurons. And, more importantly, it is unclear how such nonlinear techniques relate to biophysical mechanisms.

Linear filtering does not suffer this same lack of biological realism because it can be related to the production of postsynaptic currents (PSCs) in the postsynaptic cell. In fact, we can use (4.27) to determine the information transmission using PSCs instead of the optimal filter. Assuming a simple PSC model, $h_{psc}(t) = e^{-t/\tau_{syn}}$, where τ_{syn} is the synaptic time constant, we can perform an analogous analysis on the resulting decoding.[15]

15 The analysis is not strictly identical because, in order to determine the PSC information transmission, we replace the spike train with the PSC filtered signal. That is, R in (4.26) and (4.27) is the frequency domain representation of the PSC filtered signal for this analysis.

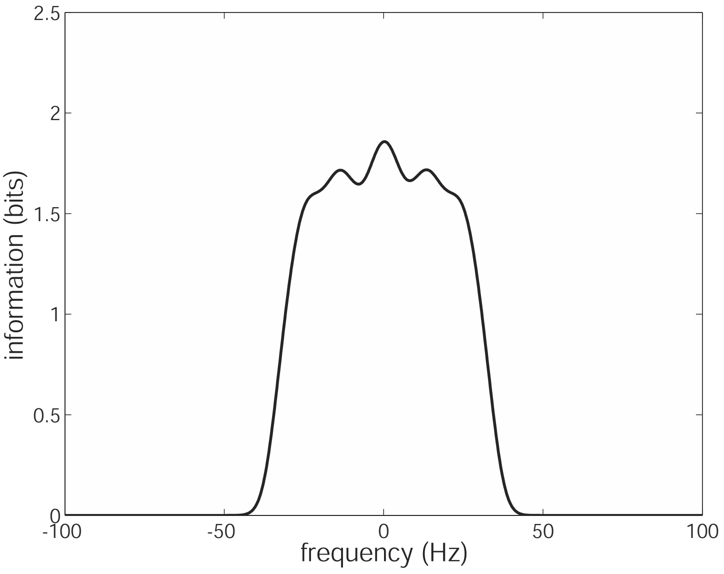


Figure 4.14
Information decodable using the postsynaptic current (PSC) as a temporal filter instead of the optimal filter as in figure 4.13.

These results are shown in figure 4.14. As can be seen from this figure, information transmission using PSCs compares very favorably to that using the optimal filter. Over the entire range, there is a 6% decrease in the information transmitted using the PSCs compared to the optimal filter. This is a very small loss considering the vast increase in neurological plausibility when using PSC decoding.

Figure 4.15 provides an example of decoding a signal with PSCs, which can be compared to the optimal decoding in the previous section (see figure 4.9a). Here, we can see that the decoding is good, though not as good as for the optimal case. However, information transmission rates are similar at 1.17 bits per spike or 108 bits per second; again similar to what is observed in real neural systems. As well, despite the fact that the RMS error is about twice that of the optimal decoders, it is still reasonably low. This is not too much of a concern since we are only using two neurons in this example. Given what we know about the effects of increasing the number of neurons in the population, we can be reasonably confident that this error can be reduced by adding neurons (in chapter 5, we explicitly show this to be the case). Given these promising results, and the vastly increased physiological plausibility of models that rely on PSCs as their temporal filters, all of the simulations we present in subsequent chapters use PSCs as the temporal decoder.

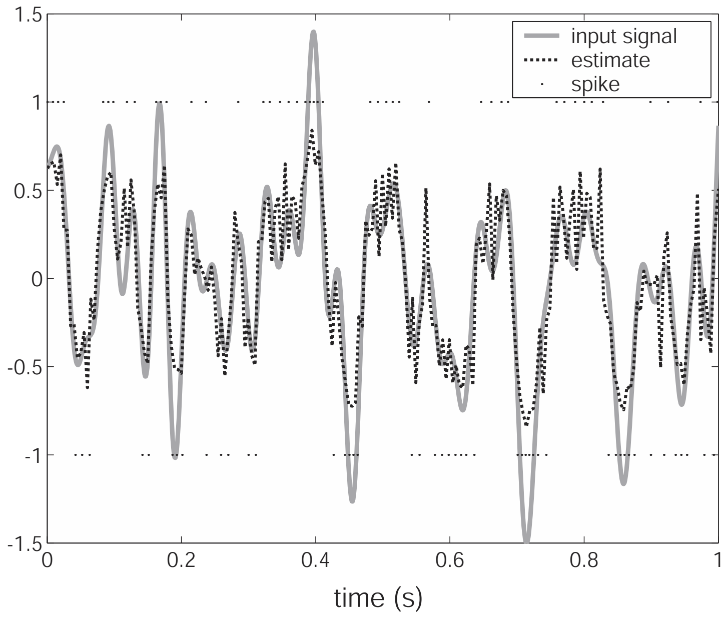


Figure 4.15
Signal reconstruction using PSC (RMSE = 0.287). This compares favorably to the reconstruction using the optimal filter in figure 4.9a.

It is worth noting that there are other methods for finding information transmission rates that often result in higher transmission rates (Stevens and Zador 1996; Buracas et al. 1998). However, by far the majority of researchers who have found information rates in real neural systems have used reconstruction methods like the one we have adopted. Thus, taking this approach facilitates comparisons with the analyses done on real neurobiological systems. Nevertheless, Stevens and Zador (1996) have made it clear that such reconstruction methods do not provide the highest possible lower bound on information transmission.

4.4.3 Discussion

Our central focus to this point in the chapter was on finding optimal decoders for characterizing temporal representation in neurobiological systems. However, we noted along the way that we do not expect there to actually be such optimal decoders in real neural systems. So why bother finding optimal decoders? There are a number of reasons. First, optimal decoders allow us to calculate quantitative measures of the system's performance. This means we can objectively compare the information processing characteristics of neural models to actual neurons. Second, developing such measures provides bounds on the performance of systems that might use non-optimal decoders, like PSCs. Third, if the optimal linear decoder does a good job of decoding, and the non-optimal decoder is substantially like the optimal decoder, then it is reasonable to assume that neural systems can be well-characterized as doing linear filtering of spike trains. Fourth, once we have used the optimal decoder to show that linear decoding is useful, we can more easily combine our characterization of the temporal code with that of the population code (as we do in section 5.1). Finally, getting a general handle on how to understand filtering in neural systems (optimal or not) should ultimately allow us to predict what kinds of filters (i.e., the characteristics of the PSCs) we expect to see given the task of certain neural systems. This is not an avenue we have explored in detail, although finding and quantifying optimal filters is a first step along this path.

4.5 More complex single neuron models

It is important to compare the LIF neuron to more complex models because there are serious concerns regarding the neurobiological plausibility of the LIF model. For example, the LIF model is very limited with respect to its modeling of the active spiking behavior of neurons. In particular, the so-called ‘spike’ in LIF models is simply ‘stuck on’ to the output when the membrane voltage crosses threshold. This is done essentially to make the output *look* like that produced by real neurons. More complex models include detailed descriptions of the active mechanisms that actually produce the voltage changes that occur during the spiking process. As well, LIF models include only a single ion current and thus do not have adapting firing rates. In mammalian cortical neurons, there are at least 12 different ion currents (Gutnick and Crill 1995; Johnston and Wu 1995). Furthermore, in a large subset of these neurons (e.g., the ‘regular-spiking’ cells), some of these currents result in spike adaptation.

Our strategy in this section is to consider successively more complex models. Thus we begin by introducing an extra ion current needed to explain adaptation into an LIF model (section 4.5.1). However, this model still does not explain spiking. Next, we consider a model that has been developed as a canonical model of a large class of neurons (section 4.5.2). This model is called the θ-neuron, and includes the spike generation process as part of its intrinsic dynamics (Ermentrout 1996; Gutkin and Ermentrout 1998a). However, this model does not include adaptation and makes mathematical reductions unfamiliar to most neurophysiologists. For these reasons, we conclude by considering a conductance-based model, which we call the Wilson model, that includes a number of currents that account directly for adaption and the relevant dynamics of the spiking process (section 4.5.3).

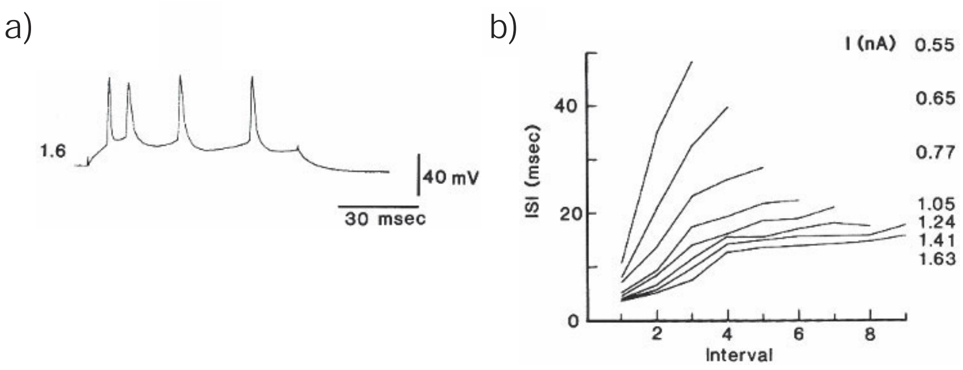


Figure 4.16
a) An adapting spike train recorded from a human regular spiking cortical neuron with an input current of 1.6 nA. b) Adaptation at different input strengths. ISI is the interspike interval, which increases with interval number for every input current. (From McCormick et al. 1985 © The American Physiological Society, reproduced with permission.)

During our consideration of each of these models, we analyze their information processing behavior using the methods developed previously. As a result, we conclude by comparing all four models to show that the various increases in complexity do not significantly affect information transmission. And, just as important when constructing large simulations, we show that the LIF neuron is a far less computationally demanding model to run.

4.5.1 Adapting LIF neuron

Adaptation, or slowing of the spike rate, is seen prominently in what are called 'regular spiking' cortical neurons (Connors and Gutnick 1990). When these neurons are injected with a depolarizing step function, they spike rapidly at first, but quickly slow their firing rate to a much lower steady-state firing rate (see figure 4.16). In order to capture this behavior in a leaky integrate-and-fire (LIF) model, we can incorporate the voltage dependent resistance as shown in figure 4.17 (Wehmeier et al. 1989; Koch 1999). This mimics the effects of the slow hyperpolarizing potassium current (J_{adapt} in figure 4.17) found in regular-spiking cells.

We can write the equations governing this circuit as follows:

$$\begin{aligned} \frac{dV}{dt} &= -\frac{1}{\tau^{RC}}\left(V(1+\frac{R}{R_{adapt}}) - J_M R\right) \\ \frac{dR_{adapt}}{dt} &= \frac{R_{adapt}}{\tau_{adapt}}. \end{aligned} \tag{4.28}$$

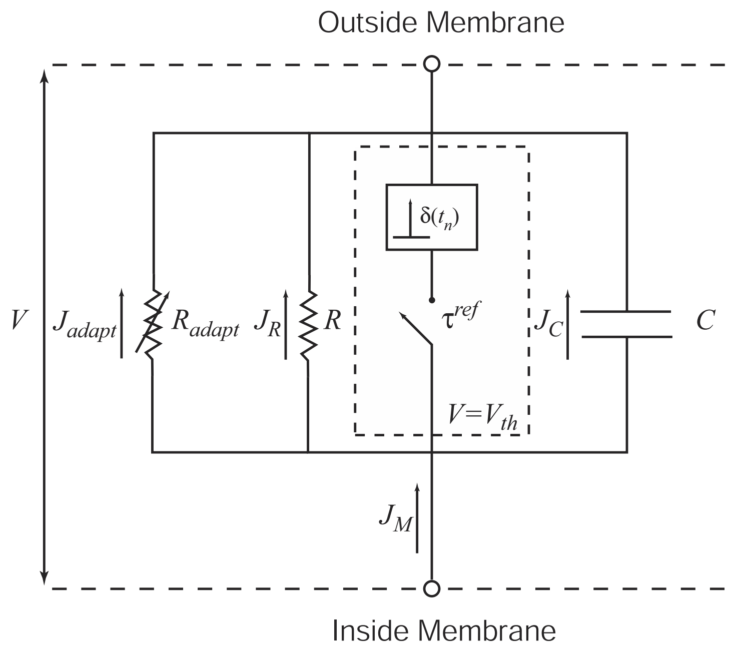


Figure 4.17
RC circuit for LIF with adaptation. The dynamics of variable resistor, R_{adapt}, are controlled by an adaptation time constant, τ_{adapt}. (See figure 4.2 for the circuit with no adaptation).

Equation (4.28) is identical to (4.4) with the addition of the time varying resistance, R_{adapt}. Notably, the time constant that controls the speed of adaptation, τ_{adapt}, will be large compared to the RC time constant of the circuit, τ^{RC}. Assuming the cell is initially at rest (i.e., $V = 0$ and $R_{adapt} = \infty$), once V_{th} is passed and an action potential is produced, the voltage dependent resistance, R_{adapt}, begins to decrease (i.e., the conductance increases) by some value, G_{inc}.[16] This introduces an extra current, J_{adapt}, which acts to lessen the effects of the input voltage, J_M. Thus, it takes longer for the next action potential to be generated because there is a larger difference between $V_{threshold}$ and V_{reset} than there was at rest. When there is no input, the resistance drifts exponentially towards its resting state.

Perhaps surprisingly, this simple addition to the LIF model makes it behave quite similarly to detailed conductance-based models, as shown in figure 4.18. As can be seen in this figure, the strong nonlinearity of the LIF model near threshold is also removed

16 For mathematical convenience, it is easier to model the decreasing resistance as an increasing conductance. Note that the conductance is 0 when the resistance is ∞.

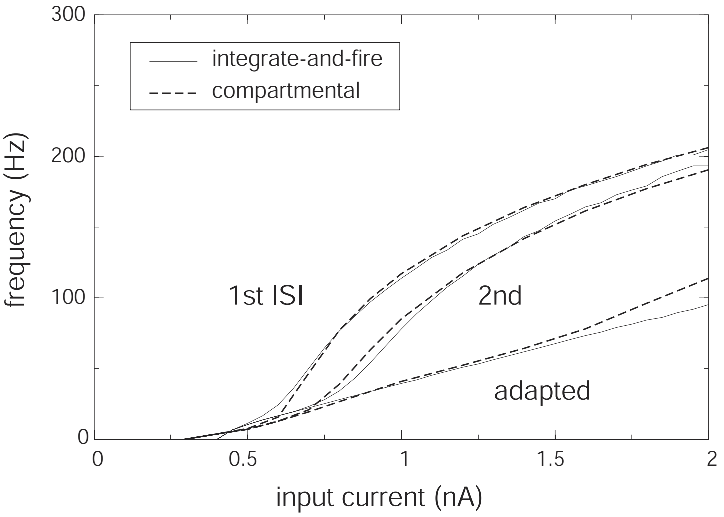


Figure 4.18
Comparison of an adapting LIF neuron with a biophysically detailed, conductance-based model of a layer 5 pyramidal neuron. (From Koch 1999, p. 337 © Oxford University Press, reproduced with permission.)

by the inclusion of adaptation. In fact, Ermentrout (1998a) has shown that, in general, adaptation in highly nonlinear neurons serves to linearize the response function. Notably, this linearization of the response function makes it more likely that we can find a good linear decoder.

Comparing figures 4.19 and 4.9a is instructive regarding the effects of including adaptation in our model neuron. As can be seen, the resultant decoding is very similar to the original LIF. In fact, both the RMSE and the information transfer rate are the same. However, the efficiency of the adapting LIF is significantly higher (2.23 bits/spike) than the standard LIF (1.24 bits/spike). This suggests that the ubiquity of adaptation in cortical neurons might serve to improve the efficiency of information transfer.

So, the adapting LIF model is just as good, if not better than the standard LIF model. However, there is also a 25% increase in the length of time it takes to simulate the adapting model, so the computational costs are significantly higher.

4.5.2 θ-neuron

Recently, there have been a number of attempts to generate simple, nonlinear, spiking models that effectively capture the behaviors of an entire *class* of neurons (Gutkin and Ermentrout 1998a; Ermentrout 1996; Hoppensteadt and Izhikevich 1997; Hoppensteadt and Izhikevich in press). Most of these are focused on understanding the dynamics of what

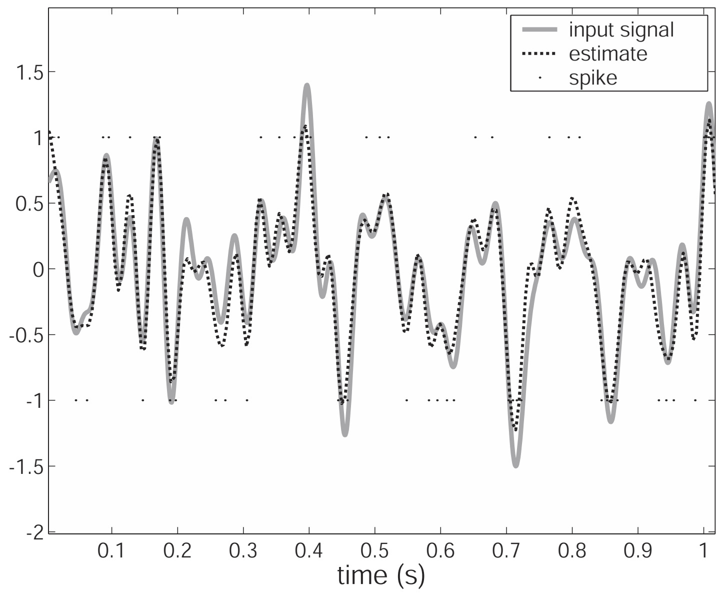


Figure 4.19
Signal reconstruction using the optimal filter as the decoder and the adapting LIF model as the encoder ($\tau_{adapt} = 55$ ms, $G_{inc} = 20$ nS, RMSE = 0.153). This can be compared to figure 4.9a, where the same signal and decoder are used but the encoder is the non-adapting LIF model.

are called 'class I' neurons. Hodgkin (1948) proposed a distinction between two classes of neurons, where class I neurons are those that can spike at arbitrarily low frequencies and steadily increase their frequency as a function of input current.[17] Notably, mammalian neurons are almost all class I neurons (Wilson 1999b, p. 149).

Class I neurons are very similar to Hodgkin-Huxley neurons, except that they incorporate an extra, very fast, voltage dependent potassium current, called the A-current (J_A in figure 4.20). Notice that the circuit describing class I behavior (figure 4.20) no longer incorporates a delta function generator as in the case of the LIF neuron. This is because the time courses of the voltage dependent potassium (R_K) and sodium (R_{Na}) channels are responsible for the generation of the neural spike (for a discussion of the dynamics of these currents see Nelson and Rinzel 1995).

17 In fact, Hodgkin suggested that there are three classes, but the third is a class of neurons that do not fire repetitively at all; presumably this is a methodological artifact. Class II neurons are those, like the famous Hodgkin-Huxley neuron (Hodgkin and Huxley 1952), that have a non-zero minimum spiking frequency, can have graded spike size, and whose firing rates are less sensitive to changes in input current strength. Arvanitaki (1938) earlier presented a similar classification.

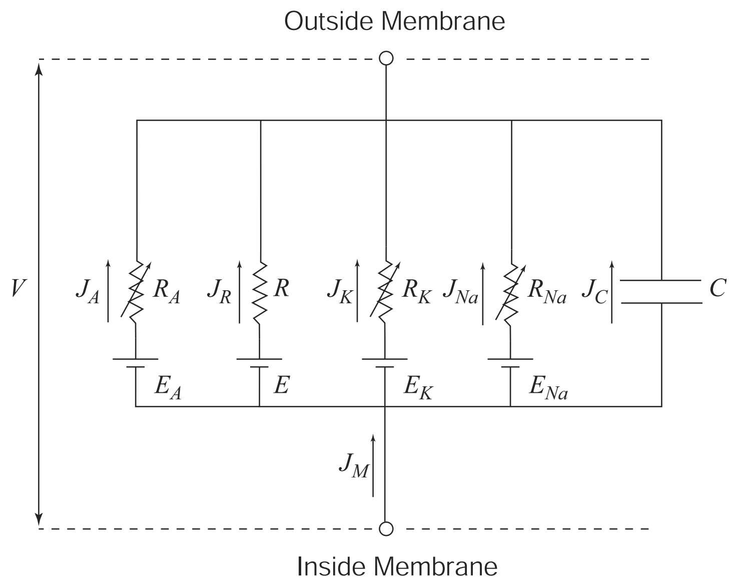


Figure 4.20
RC circuit for class I neurons. The parameters R_{Na}, R_K, R_A, and R are the sodium, slow potassium, fast potassium (A-current), and membrane leakage resistances, respectively. The equilibrium potentials, E, for the respective currents are shown as batteries. For further related discussion see section 4.5.3.

It is natural to characterize the dynamics of class I cells in the language of nonlinear systems theory (see Rinzel and Ermentrout 1989 for a discussion of nonlinear systems theory in the context of neural models). In particular, the class I cells contain what is known as a saddle-node bifurcation (Wilson 1999b; Hoppensteadt and Izhikevich in press). This occurs because of the change in recovery dynamics due to the near-rest threshold of the A-current activation and inactivation. Being able to describe the behavior of class I neurons in this general way has lead to the development of canonical models of such neurons.[18]

In order for canonical models to be useful, they need to be simple. This way, the universal properties of the entire family of models can be studied more easily. Recently, Bard Ermentrout and his colleagues have developed a simple canonical model for class I neurons called the θ-neuron (Ermentrout and Kopell 1986; Ermentrout 1996; Gutkin and Ermentrout 1998a; Gutkin and Ermentrout 1998b). The θ-neuron is a 1-dimensional model that preserves the dynamics of a saddle-node bifurcation. Essentially, the model describes the location of the neural state vector along the spike trajectory with a single phase variable,

18 A canonical model is one which any member of a family of models can be transformed into, using a continuous change of variables (Hoppensteadt and Izhikevich 1997).

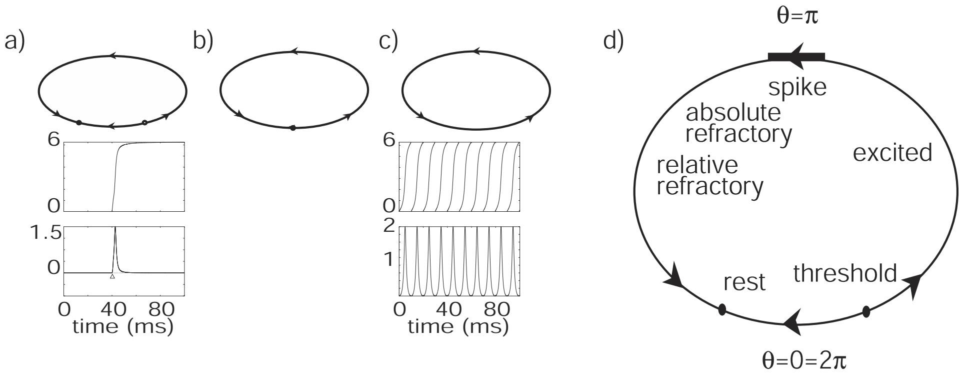


Figure 4.21
Theta-neuron behavior. In (a) and (c) the upper diagram shows the location of critical points on the invariant circle, the middle graph shows the behavior of θ, and the lower graph is the trace of $(1 - \cos\theta)$ showing the spikes. (a) Excitable regime with $\beta + \sigma = -0.3$. The stable state is the node on the right. The single spike is evoked by a pulse stimulus (marked by the triangle) that forces θ past the saddle fixed point on the right. (b) Meeting of the saddle and node points with $\beta + \sigma = 0$. The trajectory has an infinite period. (c) Oscillatory regime where the stable state is now a limit cycle with $\beta + \sigma = 0.3$. Periodic behavior of the phase variable and spikes in $(1 - \cos\theta)$ are present. (d) Phase evolution and its analog to membrane voltage states. Note that the spike occupies a small region near π. A strong enough stimulus will push θ past the threshold and into the excited region. Here the regenerative dynamics that summarize active conductances carry the phase through the spike. (Adapted from Gutkin and Ermentrout 1998a and Hoppensteadt and Izhikevich in press both © MIT Press, reproduced with permission.)

θ. The model can be written as

$$\frac{d\theta}{dt} = (1 - \cos\theta) + (1 + \cos\theta)(\beta + \sigma) \quad \text{for}\, \theta \in [0, 2\pi], \tag{4.29}$$

where β is a bias (analogous to J^{bias} in equation (2.3)) and σ is the input (analogous to J^d).

The behavior of this model is summarized in figure 4.21. As can be seen from this figure, the θ-neuron displays the main qualitative features of a class I spiking neuron, including distinct subthreshold and superthreshold regions, an all-or-none spike, a steady-state resting point, and an absolute and relative refractory period.[19]

The θ-neuron more than just captures the qualitative features of spiking neurons. (Hoppensteadt and Izhikevich) (in press) rigorously show how complex conductance models can be reduced to models like the θ-neuron. Thus, determining the information transmission characteristics of this neuron is very useful for providing insight into the behavior of a wide

19 The presence of the absolute refractory period is evident from (4.29). While the neuron is spiking, the effect of the input will be minimal since $1 + \cos\theta \approx 0$ for values of θ near π.

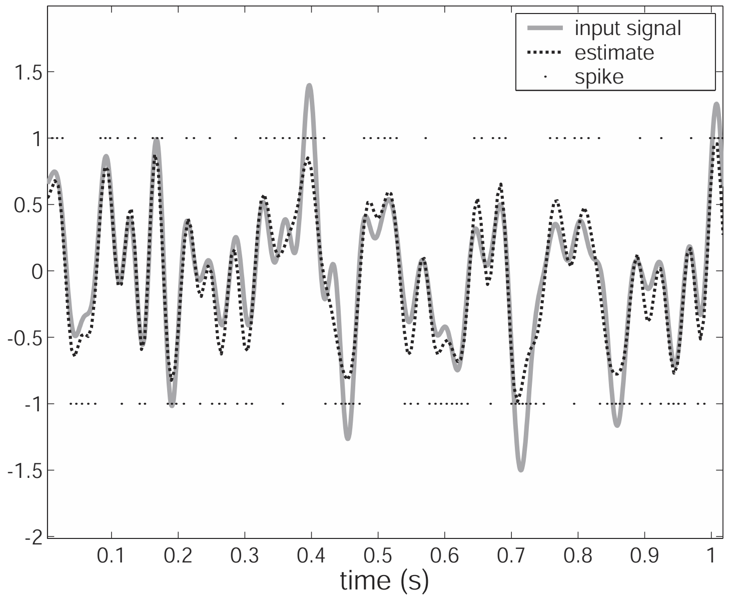


Figure 4.22
Signal reconstruction using an optimal filter and the θ-neuron model as the encoder (RMSE = 0.160).

variety of neurons. Figure 4.22 shows the decoded θ-neuron spike train for the same signal used in past examples. Note that the background firing rate and maximal firing rate over the range were matched the LIF models using β and a gain, α.

The θ-neuron compares favorably to the LIF neurons in most respects. The RMS error and information transmission rates are within 5% of each other. There is a slightly larger difference in efficiency, where the LIF (1.24 bits/spike) outperforms the θ-neuron (.96 bits/spike), but the values are comparable. The biggest difference between the models is that it takes approximately 100 times longer to run the θ-neuron. This, of course, is a major drawback when trying to run models of large, complex systems.

Given the generality of the dynamics of a canonical model like the θ-neuron, we take these results to be indicative of what should be found in all class I models. So, it is reassuring to see just how similar the information transmission characteristics are to a simple LIF model. However, despite the generality of the θ-neuron, it does suffer some important limitations. For one, the output does not map directly onto voltage changes in real neurons, so spike shape is not modeled (although spike timing is). More importantly, adaptation is not included in the θ-neuron.[20] Given the ubiquity of adaption in mammalian cortical cells, it is important that we consider models that both spike, and adapt. As well, on a more methodological note, the θ-neuron, and the methods used to reduce conductance-based models to it, are unfamiliar to most neuroscientists. In contrast, more direct reductions, like those found in the FitzHugh-Nagumo (FitzHugh 1961), Morris-Lecar (1981), and Rinzel (1985) models are likely to be more familiar. These models explicitly describe voltage dynamics, and thus produce true action potential traces, unlike the θ-neuron.

4.5.3 Adapting, conductance-based neuron

In this section, we consider the most realistic of the models we have seen so far; we call it the Wilson neuron (Wilson 1999b; Wilson 1999a). The Wilson neuron is a conductance-based model of the regular spiking neuron in mammalian neocortex. This model includes adaptation but, unlike in the adapting LIF model, the dynamics of the adaptation current are modeled directly after the calcium-dependent potassium current thought to be responsible for adaptation in class I neurons (Wilson 1999b). Like the θ-neuron, this model includes spiking dynamics. However, the reduction of this model from conductance-based models is very direct, resulting in a model that also captures the voltage dynamics observable in real neurons. As a result, this model captures features of neural spiking not addressed by the previous models. For example, changes in spike height with frequency, spike shapes, and after-hyperpolarization are captured by this model.

Let us briefly consider the reduction of complex conductance models to the Wilson neuron to show its relation to more empirically generated models (see Wilson 1999b; Wilson 1999a). To begin, consider the famous Hodgkin-Huxley (1952) model whose parameters were derived directly from experimental observations of the giant squid axon:

$$C\frac{dV}{dt} = -g_{Na}m^3h(V-E_{Na}) - g_K n^4(V-E_K) - g(V-E) + J_M \tag{4.30}$$

$$\frac{dm}{dt} = \frac{1}{\tau_m(V)}(-m+M(V)) \tag{4.31}$$

$$\frac{dh}{dt} = \frac{1}{\tau_h(V)}(-h+H(V)) \tag{4.32}$$

$$\frac{dn}{dt} = \frac{1}{\tau_n(V)}(-n+N(V)). \tag{4.33}$$

The circuit for this model is identical to figure 4.20, with the A-current removed. The parameters g_{Na}, g_K, and g are the sodium, potassium and membrane leakage conductances

20 Although it could be included, by adding a second dimension to the model analogous the slow A-current.

($\frac{1}{R}$), respectively. The parameters m, h, and n are the sodium activation, sodium inactivation, and potassium activation parameters, respectively. These parameters model the dynamics of the opening and closing of ion channels in the cell membrane. Notably, the equilibrium values of these parameters (M, H, and N) and their respective time constants (τ_m, τ_h, and τ_n) are functions of the membrane potential, V. Finally, the equilibrium potentials (shown as batteries in figure 4.20), E_{Na}, E_K, and E are the potentials for which the net ionic current across the membrane is zero (these are largely due to the ion concentration gradients across the membrane). Thus, these equations capture a fourth-order nonlinear system. Analyses of such complex systems are extremely difficult.

Fortunately, Rinzel (1985) noticed two very useful simplifications. First, he pointed out that τ_m is extremely small, so (4.31) can be eliminated by approximating m as $M(V)$ (since the equilibrium, $M(V)$, is reached quickly). Second, he realized that sodium channels close at approximately the same rate, but in the opposite direction as the potassium channels. Thus, (4.32) can be eliminated by letting $h = 1 - n$. Notably, there is now a single 'recovery' variable that results from the amalgamation of h and n, call it R. These simplifications mean that an accurate approximation to the Hodgkin-Huxley equations can be found in a two-dimensional system.

As mentioned in the previous section, the introduction of the A-current accounts for the differences between the Hodgkin-Huxley and class I neurons. A direct introduction of this current makes the two-dimensional model a three-dimensional one. However, Rose and Hindmarsh (1989) showed that a good approximation to the effects of this current is found by making the equation for the recovery variable, R, quadratic.[21] Thus, a good approximation to a class I neuron can be achieved in a two-dimensional system.

In order to introduce adaptation into the model, we must add a slow potassium current (analogous to J_{adapt} in the adapting LIF neuron), governed by the conductance variable H. Using parameters found to produce good approximations in human neocortical neurons, we can write the final Wilson model as (after Wilson 1999a; Wilson 1999b, p. 157)

$$\begin{aligned}
C\frac{dV}{dt} &= -\left(1781 + 4758V + 3380V^2\right)(V - 48) \\
&\quad -26R(V+95) - 13H(V+95) + J_M && (4.34) \\
\frac{dR}{dt} &= \frac{1}{5.6}\left(-R + 129V + 79 + 330(V+38)^2\right) && (4.35) \\
\frac{dH}{dt} &= \frac{1}{99.0}(-H + 11(V+75.4)(V+69)). && (4.36)
\end{aligned}$$

Equation (4.34) incorporates the resting potentials of sodium and potassium ions at 48 and -95 mV. The quadratic in V and the constants multiplying R and H in (4.34) are

21 See Rush and Rinzel 1994 for reservations regarding this interpretation of the A-current.

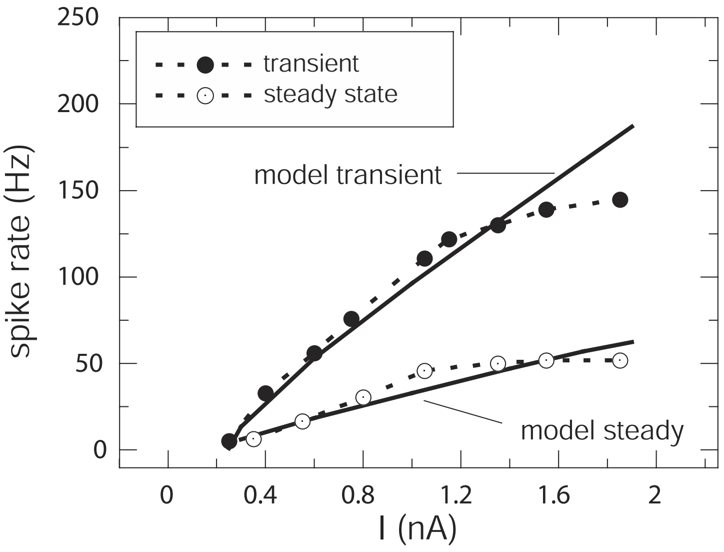


Figure 4.23
Wilson's conductance-based model compared to real data (from Avoli et al. 1994 © Springer-Verlag as in Wilson 1999b, reproduced with permission). Both transient and steady state (i.e., after adaptation) properties of a human regular spiking cortical neuron are reproduced well, and over a range of inputs.

found by fitting the nullclines of this system to the full conductance model. In (4.35), the contributions of the standard potassium current (linear) and the A-current (quadratic) have been amalgamated, as suggested by Rose and Hindmarsh (1989). Finally, (4.36) includes a term with the resting potential, -75.4 mV, so as to ensure the current has no effect at rest. Notably, the time constant of this current is very long (99.0 ms) so as to produce an appropriate adaptation without affecting the shape of the spike. As can be seen from figure 4.23, these equations do an excellent job of approximating the behavior of real regular-spiking neurons.

As shown in figure 4.24, decoding the spike train of this model again works well. The neurons used here are again matched to the original LIF model for background and maximal firing rates. In this case, the information transmission rate (91 bits/s) and RMS error are about 20% worse than for the LIF model. However, the efficiency is improved (2 bits/spike, a 30% increase). Given the results of the adapting LIF model this is likely due to inclusion of the adapting current. Again, by far the greatest difference between this model and the LIF model is that it takes approximately 600 times longer to run the same problem.

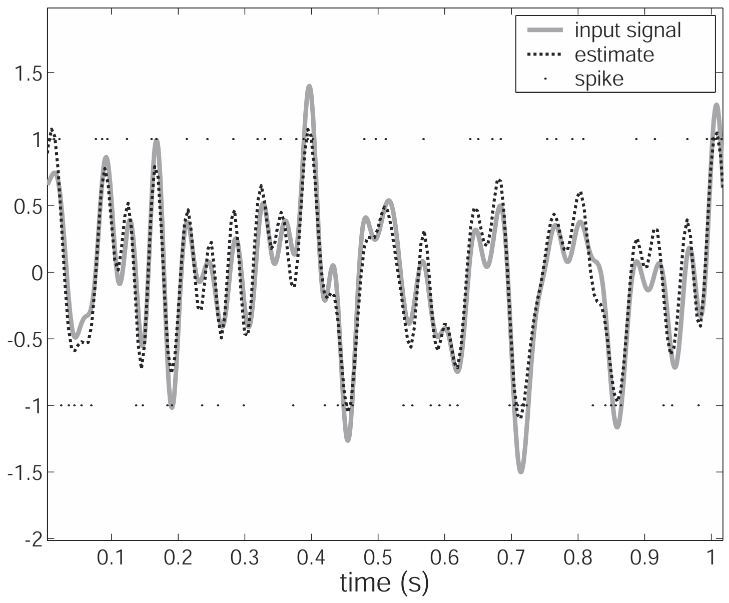


Figure 4.24
Signal reconstruction using an optimal filter and the conductance-based model as the encoder (RMSE = 0.186).

4.5.4 Discussion

In this section, we have progressed from the simple LIF model to a conductance-based, adapting model that is known to capture a wide-variety of detailed biophysical results (Wilson 1999b; Wilson 1999a). What we have shown through this discussion, is that the information transmission characteristics of a variety of single-cell models are not only very similar to one another, but also to neurons found in real neurobiological systems. Table 4.1 summarizes the results from the various models we have explored. As this table shows, there is not a very large spread in either the information transmission rates or efficiencies, although the adapting neurons are consistently more efficient. It is also important to note that the efficiencies of the models lies comfortably within the range of efficiencies reported for actual neurobiological systems, between about 1 and 3 bits/spike (see section 4.4.2). By far the greatest differences between the models lies in their run times. All in all, this tables shows that the simple LIF model is a good trade-off between a realistic neural encoder and a computationally tractable model. As models become large, computational tractability becomes extremely important. Because our focus in this book is on large-scale modeling, we adopt the LIF as our neural model of choice.[22]

22 However, the software package associated with this book allows the user to pick any of these models when constructing a simulation.

4.6 SUMMARY

In this chapter, we have been largely concerned with introducing and formally characterizing neural spiking, the most salient neural nonlinearity. We began with a study of the leaky integrate-and-fire (LIF) model, which is simple, yet incorporates this central nonlinearity. We then turned to a general discussion of temporal coding in neurons and argued that 1) it is not necessary to understand neural coding in terms of a rate code/timing code dichotomy, and 2) considering pairs of neurons as fundamental to temporal coding is highly productive. We then described a means of finding optimal temporal decoders for pairs of neurons. This analysis is very similar to those that have been done on neurons in the past, with some slight improvements (i.e., Gaussian windowing).

We employed this analysis on the LIF neuron to show that these methods effectively bridge the gap between timing and rate codes, allowing us to remain agnostic as to what 'the' code of neural systems is. As well, we showed that the simple LIF neuron has approximately the same information transfer characteristics as both real neurons and more complex neuron models. Our examination of these more complex models both showed how they could be included in this general framework and demonstrated that the LIF model strikes a convenient balance between neural plausibility and computational cost.

In addition, we addressed the issue of the biophysical relevance of the optimal decoders. We showed that the post-synaptic currents (PSCs) could be used in place of the optimal decoders with little loss in information transfer. The gain in neural plausibility is, of course, enormous. As a result of these considerations, all of our subsequent models use the LIF model and PSC temporal filtering.

Table 4.1
Comparison of information transmission characteristics of various model neurons.

Neuron	Rate	Bits/spike	RMSE	Run time (s)
LIF	114	1.24	0.153	0.18
Adapting LIF	114	2.23	0.153	0.24
θ-Neuron	109	0.96	0.160	20.1
Wilson Model	91	2.00	0.186	125.2

This page intentionally left blank

5 Population-temporal representation

To this point in the book, we have been concerned with understanding two different aspects of neural representation: 1) the representation of a static variable by a population of rate neurons; and 2) the representation of a time-varying scalar with two spiking neurons. As mentioned along the way, we need to combine these two codes in order to have a general, plausible theory of neural representation. That is the central purpose of this chapter.

5.1 Putting time and populations together again

As suggested earlier, putting population and temporal representation together ends up being a straightforward matter. This is because we have understood both kinds of representation by identifying the relevant encoding and decoding. Specifically, we consistently use nonlinear encoding and linear decoding. For both kinds of representation, the nonlinear encoding is defined by the characteristics of the neuron, and the linear decoders are found by performing a least squares optimization.

In this section, we unite population and temporal representation by considering the representation of time-varying vectors. Because we have previously related both scalar representation and function representation to vector representation, we concentrate only on vector representation for this derivation. In particular, recall that the population estimate of a vector, $\mathbf{x}$, before adding noise can be written

$$\hat{\mathbf{x}} = \sum_i a_i(\mathbf{x})\boldsymbol{\phi}_i, \tag{5.1}$$

where $a_i(\mathbf{x})$ is some nonlinear function defined by the neural response characteristics, and $\boldsymbol{\phi}_i$ is the optimal decoding weight.

In the last chapter, we defined the decoding of the 'spike' representation for a scalar which can be easily generalized to the vector case, i.e.,

$$\hat{\mathbf{x}}(t) = \sum_{i,n} \boldsymbol{\phi}_i h(t - t_{in}), \tag{5.2}$$

where $h(t)$ is our optimal decoder, and t_{in} are the occurrence times of neural spikes from neuron i (which are determined by our model of the nonlinear neural encoder). Previously, ϕ_i were ± 1 for the on/off neurons, so we can assume for the time being that the $\boldsymbol{\phi}_i$ are oppositely directed vectors in the vector space. In a sense, this is already the decoding of a population representation, although it is a somewhat degenerate population because we always assume that there are only two neurons. Just how degenerate becomes evident in the vector case, because any D-dimensional vector representation can be reduced to an equivalent one-dimensional representation. In order to capture more complex time-varying representations, we must generalize the spike representation by relating it to our general characterization of population representation.

To do so we first need to define the encoding. Of course, real neural systems have neither a population of neurons encoding a static magnitude, nor two identical but opposite neurons encoding a time-varying signal. Rather, they have populations of neurons encoding time-varying signals. So, we need to define a population of encoders of a time varying signal; i.e., we need to define some $a_i(\mathbf{x}(t))$. We begin in much the way that we did in chapter 2:

$$\begin{aligned} a_i(\mathbf{x}(t)) &= G_i[J_i(\mathbf{x}(t))] \\ J_i(\mathbf{x}(t)) &= \alpha_i \left\langle \tilde{\boldsymbol{\phi}}_i \mathbf{x}(t) \right\rangle_m + J_i^{bias}. \end{aligned}$$

Now, rather than using an activity model to define the nonlinearity G_i, we use a temporal encoding model, the spiking LIF neuron. This means that the function G_i will generate a series of spikes, $\delta(t - t_{in})$, in response to the input signal, $x(t)$. The precise spike times of neuron i, t_{in}, are determined by the parameters of the LIF model (i.e., α_i, J_i^{bias}, J_i^{th}, R_i, τ_i^{RC}, and τ_i^{ref}).

We have now defined a spiking encoder that gives rise to the activity, $a_i(\mathbf{x}(t))$. However, it is no longer clear what the relation is between this set of spikes and the continuous function $a_i(\mathbf{x})$ that we previously used in defining our population representation. In other words, we do not know what population decoders, $\boldsymbol{\phi}_i$, we can use to decode this encoding. However, looking again at the temporal decoding in equation (5.2) gives us a hint. Recall that this is a degenerate population code. When we introduced this equation in the scalar case, we noted that ϕ_i was 1 for the 'on' neuron and and -1 for the 'off' neuron. These, then play a mathematically analogous role to the decoding weights in equation (5.1). However, in a non-degenerate situation (i.e., with more than two neurons), it still is not immediately obvious what the ϕ_i will be. Only if we can relate the $h(t - t_{in})$ in the temporal representation to the $a_i(\mathbf{x})$ in the activity representation, will we show how the $\boldsymbol{\phi}_i$ in the latter case relate to the ϕ_i in the former case. That is, since $h(t - t_{in})$ and $a_i(\mathbf{x})$ act as coefficients for some decoding weights (ϕ_i and $\boldsymbol{\phi}_i$) relating the coefficients may tell us how to relate the decoding weights.

Given an analogously defined $h(t - t_{in})$ for the vector case, these relations turn out to be trivial because both $h(t - t_{in})$ and $a_i(\mathbf{x})$ are measures of the 'activity' of the *same* LIF model. In other words, the LIF response function, $a_i(\mathbf{x})$, is the instantaneous activity of the *same encoder* that generates the spikes at times t_{in} filtered by $h(t)$. Thus, as discussed in appendix C.1, we find that

$$a_i(\mathbf{x}) = \left\langle \sum_n h_i(t) * \delta(t - t_{in}) \right\rangle_T \tag{5.3}$$

$$= \left\langle \sum_n h_i(t - t_{in}) \right\rangle_T . \tag{5.4}$$

This equation should not be taken to mean that the $a_i(\mathbf{x})$ are the result of an encoding using $h(t)$. Instead, it shows that these two results of encoding can be identified.

So, as shown later in section 5.2 and appendix C.1, we can use the same ϕ_i with the spiking population as we did with the rate population. However, we do not want to average over long periods of time when we are interested in neural dynamics and real-time temporal representation. So, rather than the time-averaged $a_i(\mathbf{x})$, we can use the 'instantaneous' rate which, as T in (5.4) goes zero, is simply $\sum_n h_i(t - t_{in})$.

Thus, replacing $a_i(\mathbf{x})$ with $h(t - t_{in})$ in equation (5.1) we arrive at the population-temporal decoding:

$$\hat{x}(t) = \sum_{i,n} \phi_i h_i(t - t_{in}). \tag{5.5}$$

Note that the product of $h_i(t)$ and ϕ_i results in a scaled version of $h_i(t)$. Thus we can simplify (5.5) by defining a new 'population-temporal filter', $\phi_i(t)$, to be that product. This gives

$$\hat{x}(t) = \sum_{i,n} \phi_i(t - t_{in}).$$

Just as we did in chapter 2, we can now re-introduce noise into the representation. In this case, however, the noise is in the form of spike time jitter rather than random variations in firing rate. Notice, however, that introducing noise in spike timing is equivalent to introducing random variations in firing rate since spike timings are taken to encode instantaneous firing rate. Thus we are guaranteed that the analysis in chapter 2 regarding the effects of noise is applicable in this case as well. So, our final expression for estimating a time varying scalar signal under noise is

$$\hat{x}(t) = \sum_{i,n} \phi_i(t - t_{in} - \eta_{in}). \tag{5.6}$$

As in chapter 2, the noise term η_{in} is assumed to be drawn from a Gaussian distribution with zero mean.

In order to find the population-temporal filters that decode the encoded signal, we perform a least squares minimization, just as we did in previous chapters. First, we form our expression for the error:

$$E = \left\langle \left[x(t;\mathbf{A}) - \sum_{i,n} \phi_i(t - t_{in} - \eta_{in}) \right]^2 \right\rangle_{\mathbf{A},\eta}. \tag{5.7}$$

Now we can minimize (5.7) in one of two ways: we can either use a combination of the techniques in chapters 2 and 4; or we can just use the method from chapter 4. In general, we prefer to use a combination of techniques for two reasons: 1) as discussed earlier, we use the postsynaptic current in place of the optimal temporal decoder to improve the neurological plausibility of the models; and 2) this allows us to solve part of the problem (the population decoders) analytically, rather than relying on Monte Carlo methods (i.e., using necessarily incomplete sampling of the signal space). Furthermore, as we show in section 7.4, being able to consider just the population decoders allows for a rigorous characterization of the population representation. This is very useful for quantifying the representations in a particular population and for guiding hypotheses about the function of the system being modeled. So, in the end, despite having defined a unified population-temporal representation, it is more useful to consider the 'population' and the 'temporal' aspects somewhat independently. Nevertheless, we know that the combined decoders behave in just the ways characterized for each of the parts independently. Of course, nothing makes this more evident than examples, which we consider in section 5.3.

However, before we get to examples, it is useful to consider noise in more detail. Although we have introduced noise in (5.6), we have not demonstrated that the temporal representation we are considering is robust to the introduction of such noise. We have left this demonstration until now because only in the context of a population code can we fully characterize the effects of noise on neural representation.

5.2 Noise and precision: Dealing with distortions

Knight (1972) shows that it is important to add noise to the LIF neuron model in order to both improve the computational properties of LIF-based models and to better approximate the behavior of real neurons. On the computational side, he shows that the addition of noise stops disruptive phase-locking tendencies and thus improves the transmission of stimulus information. He also shows that the highly cusped, non-linear response function of the LIF neuron is made significantly more linear with the addition of noise, better approximating the behavior of real neurons.[1] Knight thus establishes that noise cannot be ignored when doing computational modeling with LIF neurons.

There are two ways noise can be included in our analyses when finding optimal decoders. First, we can introduce noise to spike trains before they are used to find the optimal filter; we call this ‘pre-noising’. Second, we can introduce noise to the spike train after it is used to find the optimal filter and then decode the noisy spike train; we call this ‘post-noising’. In both cases, ‘introducing noise’ means randomly jittering the spikes by some Δt picked from a normal distribution with a variance, σ^2, equal to 2 ms.

Both pre- and post-noising result in little significant effect. Specifically, for pre-noising both the RMS error and the amount of information transfered decrease by less than 1%. For post-noising, shown in figure 5.1, we see a larger effect, but the information transfer still only drops by 9.6%. A similar analysis has been done with data from real neurons, and again this kind of linear decoding was found to be similarly robust to noise (Bialek et al. 1991).

So, although in this regime of fast correlation times the spike train is more like a timing code than a rate code, highly precise timing is not mandatory. This is important because we know that real neurons operate in a noisy environment and representations that relied heavily on precise spike timing would be severely corrupted. The results presented here show that the methods we developed in section 4.3.3 appropriately generalize to these kinds of environments. So, like the population code presented in chapters 2 and 3, the temporal code is a good one under natural, noisy conditions.

However, a new concern regarding noise arises when the population and temporal codes are considered together. Namely, the temporal code introduces a possibly new kind of distortion to the population code. Specifically, using spikes decoded by either the optimal filter or the PSC filter results in significant fluctuations in the estimate of $x(t)$ (the kinds of fluctuation depend on the shape of the decoder). But, in developing the population code we used rate neurons, which had smooth, real-valued outputs, $a_i(x)$. So, it is not immediately clear what effect switching such a characterization to spiking neurons will have. Fortunately, the fluctuations introduced by neuron spiking can be treated as ‘just another’ noise term. Or, simpler yet, it can be included in the noise term already in the population analysis, σ_η^2.

To show this, consider that when the input to a LIF neuron, $x(t)$, is constant, the neuron generates a uniform spike train with interspike intervals given by $\Delta_i(x(t)) = \frac{1}{G_i[J_i(x(t))]}$ where $G_i\left[J_i(x(t))\right]$ is the function describing the spike train produced by the LIF neuron as usual. According to our characterization of temporal representation, these spike trains

1 This kind of linearization can also be accomplished by introducing adaptation (see figure 4.18). In all likelihood the results from real neurons include a combination of these two effects.

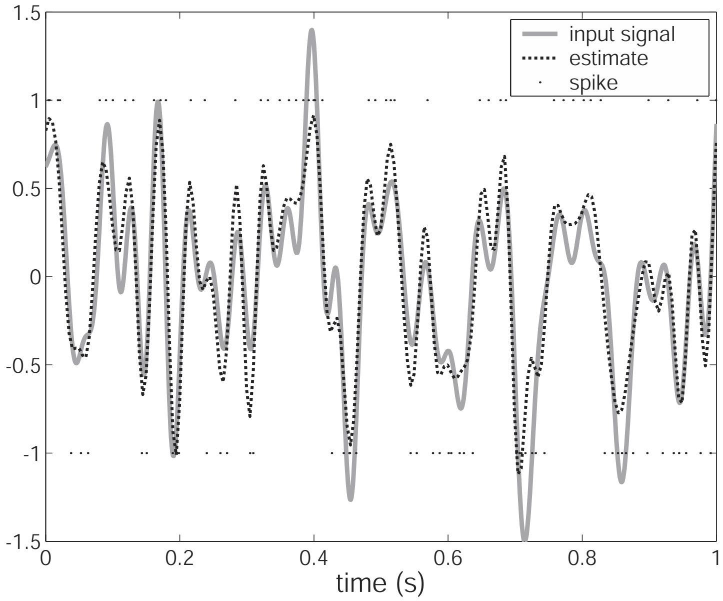


Figure 5.1
Reconstruction of a signal from a noisy spike train. The optimal filter was found using the noiseless spike train. Random spike jitter was then introduced ($\sigma^2 = 2$ms) and the signal was decoded. Total information transfer drops by 9.6% and RMSE = 0.189.

are filtered by the PSC linear filter, $h_i(t)$, resulting in a postsynaptic activity, $\alpha_i(x,t)$, given by

$$\alpha_i(x,t) = \sum_n h_i\left(t - n\Delta_i(x(t)) + t_{i_0}\right), \tag{5.8}$$

where t_{i_0} is the time of the first spike, or the phase of the spike train relative to some fixed clock. The nth spike of the ith neuron in the population will have some phase lag relative to other neurons in the population determined by t_{i_0}. For this analysis we assume that these t_{i_0} are random variables that can take on any value between 0 and $\Delta_i(x(t))$ with equal probability, and are statistically independent across neurons.

We can now find the variance of our estimate, $\hat{x}(t)$, of the signal, $x(t)$ by determining the mean activity of each neuron based on (5.8). This variance is a result of the fluctuations introduced by using spikes to transmit the signal. The derivation of the variance, $\sigma^2_{\hat{x}(t)}$, is in appendix C.1, where we show that

$$\sigma^2_{\hat{x}(t)} = \sum_i \phi_i^2 a_i(x(t)) \left[\sum_m g_i \left(m\Delta_i(x(t)) \right) - a_i(x(t)) \right],$$

where

$$g_i(\tau) = \int_{-\infty}^{\infty} h_i(t) h_i(t-\tau) dt.$$

Notably, this variance takes on a maximum value when the width of the filter, $h_i(t)$, is much smaller than the interspike interval and different neurons do not spike within the window occupied by each other's filter. This is because the filtered spike trains, $\alpha_i(x,t)$, of individual neurons will not overlap one another. If they do overlap, the result from each neuron is essentially averaged, and the fluctuations introduced by the peaked filters is reduced. In the limit where there is much overlap, then

$$\begin{aligned} \Delta_i(x(t)) \sum_m g(m\Delta_i(x(t))) &\approx \int_{-\infty}^{\infty} g(\tau) d\tau \\ &= 1, \end{aligned}$$

so

$$\begin{aligned} \sum_m g(m\Delta_i(x)) &\approx \frac{1}{\Delta_i(x(t))} \\ &= a_i(x(t)). \end{aligned}$$

In this case, the filtered spike trains look exactly like the output from the original rate model. Thus, in this limit, the optimal population decoders, ϕ_i, will be exactly the same. Furthermore, since we can characterize the fluctuations that are introduced by the spiking model in terms of the variance of the resulting estimate, $\sigma^2_{\hat{x}(t)}$, and because noise added to the neuron outputs also produces a variance in the estimate, we can capture the effects of spiking by adding an equivalent noise term to the neuron outputs.

In our discussion of population representations, we found that the error introduced into our representation was a result of both static error (from the nonlinearities in the encoders), and the error due to noise (see section 2.2.2). We can now include the error due to spiking fluctuations into this expression to determine the error from all sources in a population-temporal representation:

$$\begin{aligned} E_{total} &= E_{static} + E_{noise} + E_{fluctuations} \\ &= \frac{1}{2} \left\langle \left[x - \sum_i a_i(x)\phi_i \right]^2 \right\rangle_x + \sigma_\eta^2 \sum_i \phi_i^2 + \sigma^2_{\hat{x}(t)}. \end{aligned}$$

So, we can keep our past analyses the same and choose a variance, σ^2, that reflects both noise *and* the effects of the fluctuations due to spiking (for further discussion, see appendix C.1). Given the analysis in chapter 2, this means that the distortion effects due to spiking will also go down as $\frac{1}{N}$ (N is the number of neurons). So, our means of finding the optimal decoding weights for the population representation will be exactly the same whether we use rate neurons or spiking neurons given our characterization of temporal coding. There is no more direct way of demonstrating this than through an example.

5.3 An example: Eye position revisited

In section 2.3, we used the neural integrator as a means of introducing population coding of a scalar magnitude. In order to show that adding a temporal component to our represented signal does not adversely affect that characterization, we revisit that example in this section. In other words, we reuse the example in order to substantiate our claims in chapter 2 that the analysis there extends to more realistic neural models, and to show that our combined population-temporal decoder is a good one.

An additional benefit is that we do not have to re-describe the neural system. In fact, the system description and design specification (in sections 2.3.1 and 2.3.2 respectively) are nearly identical. In order to improve the realism of our model slightly, we no longer assume that the maximal firing rate of neurons is equal at the maximum deflection of the eyes as shown in figure 5.2 (Moschovakis 1997). We show in section 5.3.1 that this improvement does not adversely affect the expected precision of the representation. With this one alteration in mind, our specifications for the model are as follows: the variable $x(t)$ encodes eye position at a time, t; eye position neurons are well-modeled by spiking LIF neurons whose parameters are distributed as in 5.2; the range of the representation is normalized to [-1,1]; and the expected noise is independent, Gaussian distributed, has a mean of 0, and a variance of 0.1.

5.3.1 Implementation

To begin, we want to verify that simply changing the distribution of neurons in the population, by allowing a range of maximal firing rates at maximal eye deflection does not adversely affect the ability of the system to deal with noise. Figure 5.2 shows a sample of tuning curves in the population used in these simulations. Figure 5.3 shows that, for a static encoding, noise decreases as before (see section 2.3), at a rate proportional to $1/N$, where N is the number of neurons.

We can now compare these results to those found with the same set of neuron tuning curves, but using spiking neurons. As before (see section 4.3), we draw our signals from

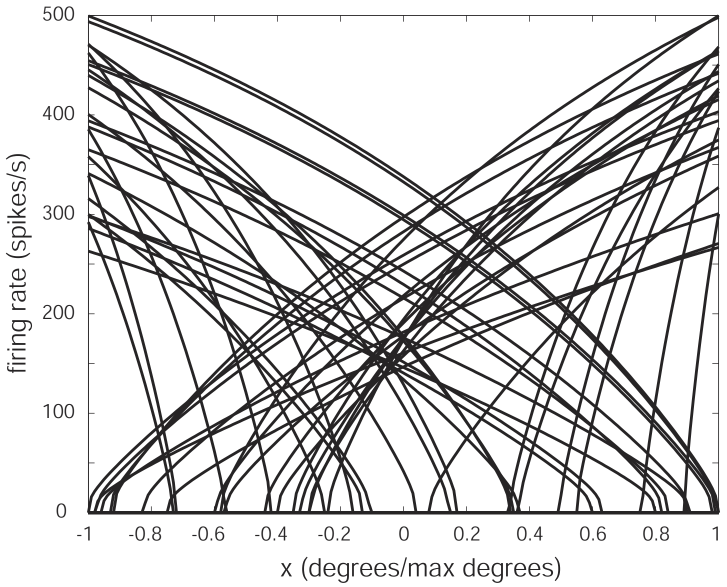


Figure 5.2
A sample of 50 nonlinear tuning curves for a population of neurons used to encode horizontal eye position.

an ensemble defined in terms of statistically independent coefficients, **A** (see equation (4.14)). In particular, we assume the coefficients are Gaussian distributed (as in equation (4.15)), and cover a bandwidth of 5 Hz. This defines a signal ensemble that is Gaussian band-limited white noise, as before (see section 4.4).

Figure 5.3 shows the decrease in error as the number of neurons increase for the activity model neurons and for the PSC-filtered spiking neurons. Although, as this graph shows, the activity neurons have a slightly smaller error for a given number of neurons, both models show a decrease in error which goes as $1/N$. Thus, for any activity model representation, we can build an equally accurate filtered spiking model representation, although we may need a few extra neurons. This demonstrates that, as expected, the analyses we perform on the simplified rate model translate to our more biologically plausible model that includes neural spikes and more realistic postsynaptic behavior.

5.3.2 Discussion

Although we do not show it here, these same results hold for vector and function representation. This should not be surprising given the close relationship we have established between scalar and vector representation and between vector and function representation.

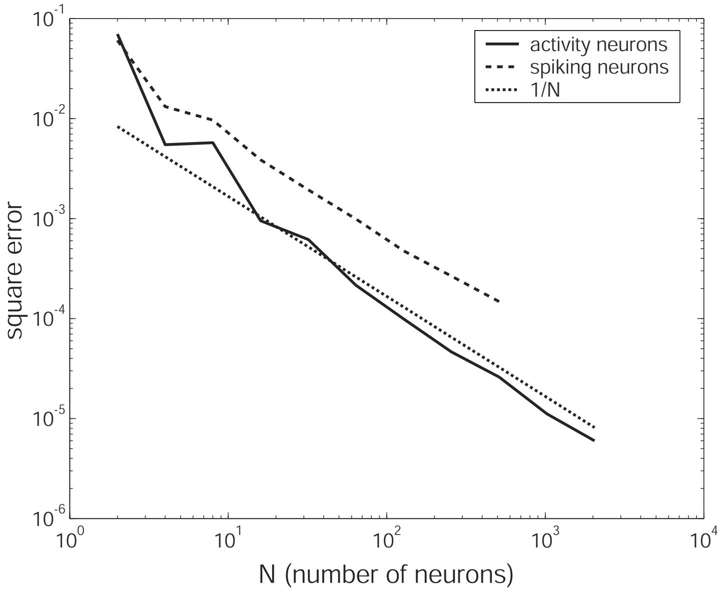


Figure 5.3
Simulation results showing the relation between the precision of representation and the number of neurons in the population. The solid line shows the decreasing error due to noise as a function of the number of neurons for the activity neurons. The dashed line indicates the same trend for the PSC filtered spiking neurons. For the spiking case, MSE is determined over a 1 second run for a 5 Hz band-limited signal. Although the activity population has slightly better absolute error, both display the same tendency to decrease error at a rate proportional to $1/N$ (the dotted line).

To conclude our overview of population-temporal representation, let us consider the population-temporal code of functions, which we have not yet directly addressed. As in the case of scalars, amalgamating temporal and population coding is a simple matter. The encoding is

$$a_i(x(\nu,t;\mathbf{A})) = a_i(t;\mathbf{A}) = G_i\left[\alpha_i\left\langle x(\nu,t;\mathbf{A})\tilde{\phi}_i(\nu)\right\rangle_\nu + J_i^{bias}\right],$$

which can be written as

$$a_i(t;\mathbf{A}) = \sum_n h_i(t - t_{in}(\mathbf{A}))$$

since ν is integrated out, as discussed in section 3.2.

Our decoded estimate of the function is thus

$$\hat{x}(\nu, t; \mathbf{A}) = \sum_i \phi_i(\nu) a_i(t; \mathbf{A}) \tag{5.9}$$

$$= \sum_i \phi_i(\nu, t) * a_i(\mathbf{A}) \tag{5.10}$$

$$= \sum_{i,n} \phi_i\left(\nu, t - t_{in}(\mathbf{A})\right), \tag{5.11}$$

where the $\phi_i(\nu)$ in (5.9) could be found by minimizing the appropriate error as in section 3.2. Alternately, we can look at equation (5.10), in which the function $\phi_i(\nu, t)$ is the population-temporal filter and $a_i(\mathbf{A})$ is the spike train produced by neuron i in response to the function characterized by $\mathbf{A}$. We can then minimize the error,

$$E = \left\langle \left[x(\nu, t; \mathbf{A}) - \sum_i \left[a_i(\mathbf{A}) + \eta_i\right] * \phi_i(\nu, t) \right]^2 \right\rangle_{\mathbf{A}, \eta, t},$$

to find the population-temporal filter. Although we have amalgamated the parameters for determining the functions of ν with the parameters for determining the temporal functions in the vector $\mathbf{A}$, this is by no means necessary. As mentioned in section 5.1, we can minimize the temporal and population errors *independently* and then construct the population-temporal filter. Or, as also mentioned earlier, we can approximate this minimization by assuming the PSCs are given as the temporal filters and then find the optimal population decoders only. In any case, this example shows that the considerations for scalar population-temporal representation generalize to the representation of functions as expected. We examine a neurobiological example of population-temporal function representation in section 8.3.

5.4 Summary

We began this chapter by showing that our previously independent population and temporal representation characterizations could be combined. As a result, we defined a population-temporal representation by identifying the encoding

$$a_i(\mathbf{x}(t)) = G_i \left[\alpha_i \left\langle \tilde{\boldsymbol{\phi}}_i \mathbf{x}(t) \right\rangle_m + J_i^{bias} \right],$$

and decoding

$$\begin{aligned}\hat{x}(t) &= \sum_i \phi_i a_i(\mathbf{x}(t)) \\ &= \sum_{i,n} \phi_i h(t) * \delta(t - t_{in}) \\ &= \sum_{i,n} \phi_i(t - t_{in}),\end{aligned}$$

where the $\phi_i(t)$ can be found directly, or by determining $h(t)$ and ϕ_i independently. As mentioned, in the remainder of the book we opt for the latter approach, as it allows easier analysis and more direct generalization.

We then showed that this representation was robust to noise. Furthermore, we demonstrated that determining the decoders under noise is the same for the rate and spiking neuron models. This is because the fluctuations due to spiking can be included in the original expression for error by simply adding another noise term.

II TRANSFORMATION

This page intentionally left blank

6 Feed-forward transformations

In the first half of this book we have concentrated on characterizing representation in neurobiological systems. Ultimately, however, we do not merely want to catalog kinds of representation, but instead to explain the *behavior* of these systems by reference to their representations. In order to do so, we can understand that behavior as a result of representations being compared, contrasted, combined, weighted, averaged or otherwise *transformed*. So, along with a story about representation, we need, just as much, a story about how representations can be manipulated. Only by understanding both representation and transformation can we begin to explain the wide variety of intriguing behaviors exhibited by neurobiological systems.

So, in the second half of this book, we characterize neurobiological transformations.[1] Doing so requires us to be concerned with a) the different *types* of transformations in neurobiological systems, b) how to *implement* transformations in neurobiological systems, and c) the *dynamics* of transformations in neurobiological systems. In this chapter, we discuss the basics of implementation. In the next chapter we present an analysis that can help us determine the kinds of transformations a neural population can support, and gives us deeper insight into the nature of neural representation. In chapter 8, we address the dynamics of neural systems. There, we provide a general characterization of neural dynamics and present a number of examples that show how the general characterization can be applied. Finally, in the last, more speculative chapter we return to issues of implementation and discuss learning and statistical inference as it can be understood within this framework. Together, these chapters provide an overview and analysis of many of the kinds of transformations found in neurobiological systems.

6.1 Linear transformations of scalars

6.1.1 A communication channel

Let us begin by considering the simplest kind of linear, feed-forward transformation: none at all. As ridiculous as this suggestion may at first seem, it helps elucidate a technique that can be used to implement significantly more complex transformations. To make the example seem more interesting, let us consider this transformation as characterizing a simple communication channel. The purpose of such a channel is to send some signal

1 A terminological note: although we use the term 'transformations' throughout, we do not object to calling these same processes 'computations'. However, the term 'computation' has, for many, a definition along the lines of "rule-governed manipulation of *discrete*, *symbolic* representations" (see, e.g., Gelder 1995; Fodor and Pylyshyn 1988). This definition is inappropriate for many of the kinds of computations we discuss.

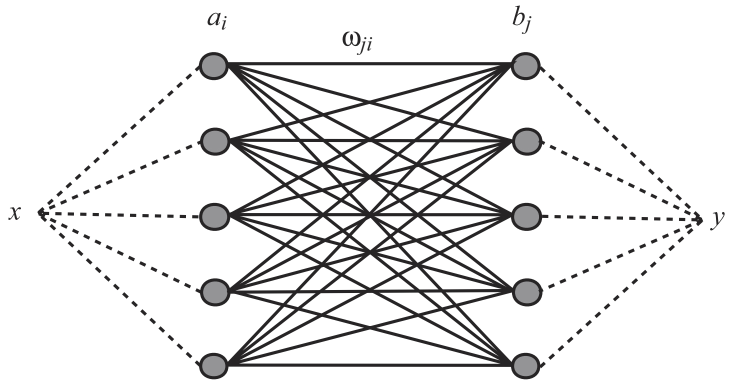


Figure 6.1
A communication channel in a neural network. Solid lines indicate possible connections. Dotted lines indicate an encoding/decoding relation.

from one location to another location. In this case, our signal will be a scalar value, x, and the two locations are neural populations a and b (see figure 6.1).

We can begin constructing our model by writing down the representations in the two neural populations, as derived in section 2.1.2. Thus, we have the population

$$\begin{aligned} a_i(x) &= G_i\left[J_i(x)\right] && (6.1)\\ &= G_i\left[\alpha_i\tilde{\phi}_i x + J_i^{bias}\right] && (6.2)\\ \hat{x} &= \sum_i a_i(x)\phi_i^x, && (6.3) \end{aligned}$$

where (6.2) defines the encoding and (6.3) defines the decoding. As before, $G_i\left[\cdot\right]$ is taken to be defined by a LIF neuron, and $\tilde{\phi}_i$ is 1 for 'on' neurons and -1 for 'off' neurons. In (6.3) we have introduced a new notation with respect to the decoding weight, ϕ_i^x. From now on, we often write the variable that the decoder is *for* (i.e., x) as a superscript on the decoder (i.e., ϕ_i), as this serves to disambiguate decoders. Disambiguation is important because, as we will see, characterizing transformations often requires identifying multiple decoders.

For population b we define the representation analogously:

$$\begin{aligned} b_j(y) &= G_j\left[J_j(y)\right] && (6.4)\\ &= G_j\left[\alpha_j\tilde{\phi}_j y + J_j^{bias}\right] && (6.5) \end{aligned}$$

$$\hat{y} = \sum_j b_j(y)\phi_j^y. \tag{6.6}$$

Now, in order to implement a communication channel using these two populations, we need to define the transformation in terms of the representations in each population. In this case, the transformation is simply $y = x$. That is, we want our population y to represent whatever is represented by x. Knowing this transformation allows us to write the activities of b in terms of those of a by substituting x for y (since $y = x$). Thus,

$$\begin{aligned} b_j(x) &= G_j\left[\alpha_j\tilde{\phi}_j x + J_j^{bias}\right] &(6.7)\\ &= G_j\left[\alpha_j\tilde{\phi}_j \sum_i a_i(x)\phi_i^x + J_j^{bias}\right] &(6.8)\\ &= G_j\left[\sum_i \omega_{ji}a_i(x) + J_j^{bias}\right], &(6.9)\end{aligned}$$

where $\omega_{ji} = \alpha_j\tilde{\phi}_j\phi_i^x$. Equation (6.8) is a result of supposing that $x = \hat{x}$ and substituting (6.3) into (6.7). In this case, ϕ_i^x is the transformational decoder (although it happens to be the same as the representational decoder because of the simple nature of the transformation that describes a communication channel). Equation (6.9) is the standard 'connection weight' form for(6.8). Equation (6.9) essentially says that the activity of some neuron in the b population is a sum of the activity of the neurons connected to it in the a population, times a weight and passed through the nonlinear (LIF) response function. In more neurobiological terms, the instantaneous firing rate of a neuron in b_j is a result of the LIF response to the current $J_j(x)$, which is determined by the sum of the dendritic currents (i.e., $\omega_{ij}a_i(x)$) plus some biasing background current J_j^{bias}. The dendritic currents are determined by the product of their synaptic efficacy (ω_{ji}) and the instantaneous firing rate of the presynaptic neuron $a_i(x)$.

In order to turn this firing rate model into a spiking model, we can employ the techniques discussed in section 5.1. There, we showed that we can write

$$\begin{aligned} \hat{x}(t) &= \sum_i a_i(x(t))\phi_i^x \\ &= \sum_{i,n} h_i(t-t_{in})\phi_i^x. \end{aligned}$$

So, the spike trains of the b population are now determined as follows:[2]

$$\begin{aligned} b_j(x(t)) &= G_j\left[\alpha_j\tilde{\phi}_j x(t) + J_j^{bias}\right] &(6.10)\\ &= G_j\left[\alpha_j\tilde{\phi}_j\sum_{i,n} h_i(t-t_{in})\phi_i^x + J_j^{bias}\right] &(6.11)\\ &= G_j\left[\sum_{i,n}\omega_{ji}h_i(t-t_{in}) + J_j^{bias}\right], &(6.12)\end{aligned}$$

where $\omega_{ji} = \alpha_j\tilde{\phi}_j\phi_i^x$ as for the rate model. Of course, in these cases, $G_j[\cdot]$ is presumed to be defined by a spiking LIF neuron. Again, the elements of equation (6.12) have correlates in neurophysiology. In particular, the ω_{ij} are synaptic efficacies, the filters (i.e., temporal decoders) $h_i(t)$ are post-synaptic currents (PSCs), the t_{in} are spike arrival times for spike n from neuron i, and J_j^{bias} is the bias current. These interpretations have each been discussed in more detail previously. We should note that the PSCs are better indexed $h_{ij}(t)$, indicating the PSC at synapse i on neuron j. However, for convenience, we assume for the time being that each neuron onto which neuron i synapses has the same (unweighted) PSC. This, of course, is emphatically not the same as assuming that all currents produced in all dendrites are the same given the occurrence of a spike. Those currents are a product of $h_i(t)$ and ω_{ji}.

In order to render our model more realistic, we can include the effects of noise on the activities. This changes the values of the decoding weights, ϕ_i^x, and results in the substitution of $a_i(x) + \eta_i$ for $a_i(x)$ above. Otherwise little in the derivations themselves changes.

Suppose, now, that we are interested in implementing a slightly more complex transformation. That of a static gain on our input (i.e., $y = cx$). The derivations follow exactly as before with the following result:

$$b_j(cx(t)) = G_j\left[\sum_{i,n}\omega_{ji}h_i(t-t_{in}) + J_j^{bias}\right], \tag{6.13}$$

where $\omega_{ji} = c\alpha_j\tilde{\phi}_j\phi_i^x$. The results of simulating this network for $c = 0.5$ is shown in figure 6.2.

2 Alternatively, we could use $\hat{x}(t) = \sum_{i,n}\phi_i^x(t - t_{i,n})$ in which case we find $b_j(x(t)) = G_j\left[\sum_{i,n}\omega_{ij}(t-t_{i,n}) + J_j^{bias}\right]$ where $\omega_{ij}(t-t_{i,n}) = \alpha_j\phi_i^x(t-t_{i,n})$. In some sense, writing the weight, ω_{ij}, as a function of time more directly reflects the observed behavior of postsynaptic cells (i.e., a scaled PSC). However, distinguishing weights from temporal decoders is, as previously mentioned, extremely important for performing useful analyses of neurobiological systems (see, for example, section 7.3).

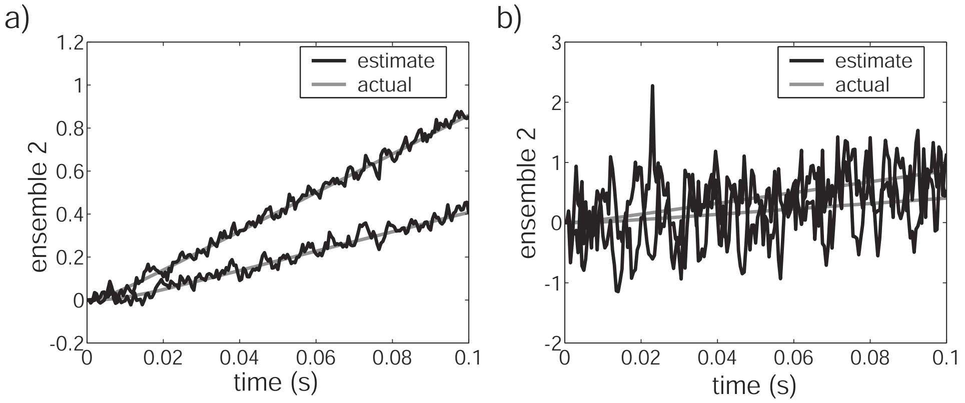


Figure 6.2
Simulation results for a communications channel with static gain using 200 spiking LIF neurons. a) Shows the results from a ramp input, where the upper line is the input (i.e., the x population) and the lower line is the output (i.e., the y population). In both cases, the grey line indicates the ideal value of the higher-level variable (i.e., x). This network includes noise. b) Shows the effects of finding the decoding vectors while ignoring the effects of noise, and then running the simulation under the same noise conditions as in a). The system becomes extremely sensitive to noise, destroying the desired behavior.

As figure 6.2 once again demonstrates, noise cannot be ignored when constructing models of neurobiological systems. The system whose results are shown in figure 6.2b has, in fact, the same amount of noise in its input signal as the system whose results are shown in figure 6.2a. The difference is only that the decoders in figure 6.2a were found under the expectation of noise. Without including this expectation explicitly, the network's behavior is ruined.

This network also demonstrates two important theoretical results of the work we have done in the first half of the book. First, it shows that we are able to analytically determine the weights needed for a network to implement a desired transformation. Analytically determining weights is one of the major strengths of this framework. For one, this means we do not have to concern ourselves with the difficulties involved in analyzing a learned network—such analyses often prove to be a momentous or fruitless undertaking. For another, we do not have to worry about whether or not our chosen training set permits the network to learn the function we would like to examine in a model network. For a third, finding the weights analytically is generally far less computationally intensive than running large training regimes. Finally, it might prove useful to consider these weights a good 'first guess' for a system that is then 'fine-tuned' by an appropriate learning algorithm. As is well known, having a good first guess can greatly speed learning. So, in general, and as we demonstrate concretely later, being able to find network weights directly allows us to efficiently generate large-scale networks with complex behaviors.[3] In sum, being able to avoid or incorporate learning as necessary makes this approach usefully flexible, and thus appropriate for many of the standard uses of neurobiological simulation (see section 9.5 for more on the relation of this framework to learning).

The second result highlighted by this simple example is that carefully characterizing neural representation is central to understanding neural transformation. This becomes even more evident in section 7.3, where we provide a quantitative demonstration of the relation between representation and transformation in a neural population. In this example, we have shown how we can 'plug' one representational characterization, x, into another, y, in order to determine the weights that implement the desired transformation between the two. Thus, the limitations or strengths of a pre-transformed representation can carry over to post-transformed representation. So knowing what those limitations and strengths are is tantamount to knowing how the resulting network functions. Let us consider some more examples to show that this approach of simply 'plugging in' representations works surprisingly well.

6.1.2 Adding two variables

In order to implement any *linear* transformation, we need to be able to multiply variables by a constant (as we have done above) and add two (or more) variables together. In order to add two variables, we need to define the representations in three neural populations. We can then arrange these populations as shown in figure 6.3, to transform the representations of the scalars in the first two populations to a representation of their sum in the third.

We begin by assuming the same representational definitions as in section 6.1.1. Thus, all three populations have encoding and decoding relations defined analogously to that shown here for the a_i population:

$$\begin{aligned} a_i(x) &= G_i\left[\alpha_i\tilde{\phi}_i x + J_i^{bias}\right] && (6.14)\\ \hat{x} &= \sum_i a_i(x)\phi_i^x. && (6.15)\end{aligned}$$

Our desired transformation in this case is simply

$$z = x + y.$$

3 Of course, none of this implies that learning is not important to a complete understanding of neural function. We return to these issues in chapter 7.

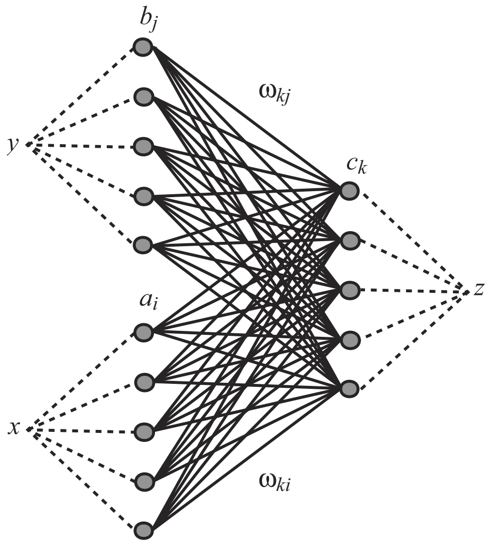


Figure 6.3
A network to sum two variables. Solid lines indicate possible connections. Dotted lines indicate an encoding/decoding relation.

So, as before, we assume our estimates are good ones (i.e., $\hat{x} = x$, etc.) and substitute the representations for x and y into the encoding rule for z to give

$$\begin{aligned} c_k(x+y) &= G_k\left[\alpha_k \tilde{\phi}_k (x+y) + J_k^{bias}\right] \\ &= G_k\left[\alpha_k \tilde{\phi}_k \left(\sum_i a_i(x)\phi_i^x + \sum_j b_j(y)\phi_j^y\right) + J_k^{bias}\right] \\ &= G_k\left[\sum_i \omega_{ki} a_i(x) + \sum_j \omega_{kj} b_j(y) + J_k^{bias}\right], \end{aligned}$$

where the weights $\omega_{ki} = \alpha_k \tilde{\phi}_k \phi_i^x$ and $\omega_{kj} = \alpha_k \tilde{\phi}_k \phi_j^y$ determine the connection strengths needed to perform the given transformation (i.e., addition). Given the resultant firing rates, $c_k(z)$, we determine the estimated value of z by using the appropriate decoding rule. Again, we can make this rate model into a spiking model by using PSCs as our linear temporal decoder, $h(t)$. Figure 6.4 shows the results of simulating such a network, using spiking LIF neurons.

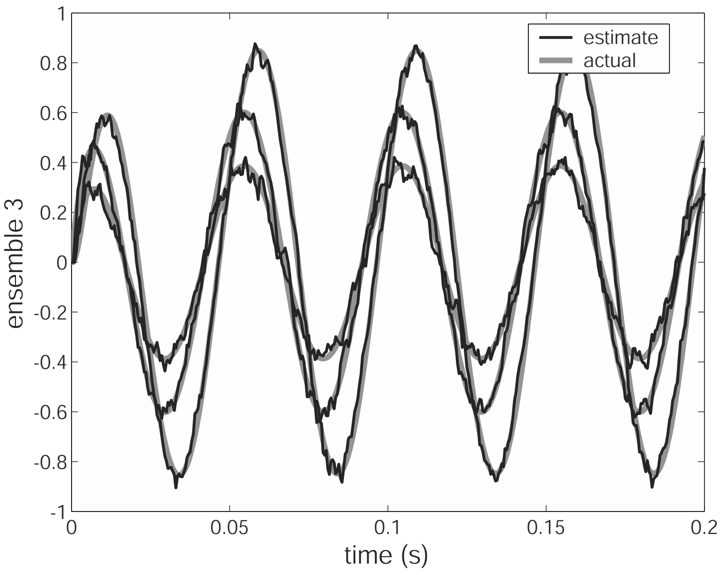


Figure 6.4
Addition of two variables using 600 spiking LIF neurons (200 in each population). The original inputs, two 20 Hz sine waves, and their sum are shown as grey lines. The decoded outputs are shown as black lines. As expected, there is a slight time lag between the inputs and their sum due to propagation delays.

Since we now have the tools to multiply by a constant and sum two variables, we can also perform scalar subtraction (i.e., $z = x + (-1) \cdot y$). In fact, we now have a general 'recipe' for constructing linear transformations of scalars in neurally plausible networks. First, define the encoding and decoding rules for however many variables are involved in the operation (e.g., equations (6.14) and (6.15)). Second, write the transformation to be performed in terms of these variables, one of which is the output variable (e.g., z above). Third, write the transformation using the decoding expressions for all variables except the output variable. And fourth, substitute this expression into the encoding expression of the output variable. Rather than performing each of these steps in consecutive layers, as we have done above, it is perfectly feasible to implement various parts of a complex transformation in parallel. In the end, the output layer's activities are written in terms of the weighted activities of the input layers.

Being able to provide a general method for implementing linear transformations shows that this framework allows for the *modular* design and implementation of a network model. Once we write the function being performed by a network in terms of some set of variables, we can then define pseudo-independent populations of neurons to represent those variables.

That is, we can define ‘representational modules’ and then interconnect them using weights found by employing these techniques. Of course, this modularity need not be obvious in the neurobiological system itself. We could, after all, randomly mix the spatial position of neurons in our model. However, the *function* will be preserved so long as we keep the right weights in the right places. With such a spatially randomized model, it may be very difficult to determine, after the fact, which neurons belong to which ‘populations’ as defined in the original modularization of the problem. This shows that comparing a model with a real system may be a subtle and difficult task. Furthermore, there will likely be many cases in which it is not reasonable to model a system as a set of modular neuronal populations, even though we *could* construct such a model. Again this demonstrates that we must always look to the neuroscientific data to determine what a reasonable modularization is for a given system. However, modularization is *still* a useful feature of the framework because often the only feasible way to model large, complex systems is by *first* decomposing them into smaller modules (Bechtel and Richardson 1993). We provide an example of this process in section 6.5.

Given the ability to modularize complex transformations, we are in a position to implement any linear combination of scalar variables in a realistic spiking network. As useful as this may be, we need to generalize it in two ways. First, we need to explore transformations of more complex kinds of representations (e.g., vectors). We do this next, in section 6.2. Second, it is clear that the set of linear operations is too limited to capture a number of interesting network-level behaviors. So, we need to expand the set of transformations available to us in order to generate models capable of exhibiting something akin to the range behaviors of neurobiological systems. This is the purpose of section 6.3.

6.2 Linear transformations of vectors

Constructing a network to perform vector addition is perfectly analogous to constructing a network to perform scalar addition. In fact, the network topology itself is identical to that portrayed in figure 6.3. The only difference is that each of the variables represented is a vector, rather than a scalar. Proceeding as before, we first define the representation of the variables involved:

$$a_i(\mathbf{x}) = G_i\left[\alpha_i \left\langle \tilde{\phi}_i \mathbf{x} \right\rangle_m + J_i^{bias}\right] \tag{6.16}$$

$$\hat{\mathbf{x}} = \sum_i a_i(\mathbf{x}) \phi_i^{\mathbf{x}}. \tag{6.17}$$

We again assume that the other variables, $\mathbf{y}$ and $\mathbf{z}$, have analogous definitions. Note that we must now include the encoding (or 'preferred direction') vector, $\tilde{\phi}_i$, in the encoding rule (6.16). Recall that this essentially converts a vector of some physical magnitude into a scalar, via the dot product. In the scalar case, we did not need to explicitly denote this sort of conversion.

Next, we define the transformation we want to implement in the network:

$$\mathbf{z} = C_1\mathbf{x} + C_2\mathbf{y}.$$

This transformation includes both a static scalar gain factor C_m and vector addition. So, as before, constructing this network shows how to construct a network to perform any linear transformation.

Finally, we write the transformation using the representations defined by (6.16) and (6.17) and substitute that expression into the encoding rule for the output variable, $\mathbf{z}$:

$$\begin{aligned}
c_k(C_1\mathbf{x} + C_2\mathbf{y}) &= G_k\left[\alpha_k \left\langle \tilde{\phi}_k (C_1\mathbf{x} + C_2\mathbf{y})\right\rangle_m + J_k^{bias}\right] \\
&= G_k\left[\alpha_k \left\langle \tilde{\phi}_k \left(C_1 \sum_i a_i(\mathbf{x})\phi_i^{\mathbf{x}} + C_2 \sum_j b_j(\mathbf{y})\phi_j^{\mathbf{y}}\right)\right\rangle_m + J_k^{bias}\right] \\
&= G_k\left[\sum_i \omega_{ki} a_i(\mathbf{x}) + \sum_j \omega_{kj} b_j(\mathbf{y}) + J_k^{bias}\right],
\end{aligned}$$

where $\omega_{ki} = \alpha_k C_1 \left\langle \tilde{\phi}_k \phi_i^{\mathbf{x}} \right\rangle_m$ and $\omega_{kj} = \alpha_k C_2 \left\langle \tilde{\phi}_k \phi_j^{\mathbf{y}} \right\rangle_m$. More generally, we can allow the scalars, C_1 and C_2, to be the matrices, $\mathbf{C}_1$ and $\mathbf{C}_2$. In this case, the resulting weights are of the form $\omega_{ki} = \alpha_k \left\langle \tilde{\phi}_k \mathbf{C}_1 \phi_i^{\mathbf{x}} \right\rangle_m$. Using matrices rather than scalars permits the incorporation of various kinds of linear operations such as rotation and scaling.

As before, we can include spiking neurons in this transformation by allowing $a_i(\mathbf{x}) = \sum_n h(t - t_{in})$. Figure 6.5 shows results from simulating this network for $C_1 = C_2 = 1$. Notice that after a brief 20 ms startup transient, the sum remains quite accurate, even under noisy conditions.

Clearly, this network does a good job of vector addition using the representations we have defined. If we want to improve the network's performance, we can simply add neurons. Although we have not discussed this transformation in the context of a specific neurobiological system, there are indications that such a network may be used in frontal eye fields for control of saccades (Bickle et al. 2000; Goldberg and Bruce 1990). Before developing an example of transformations in a specific neurobiological system, let us first consider more complex, nonlinear transformations.

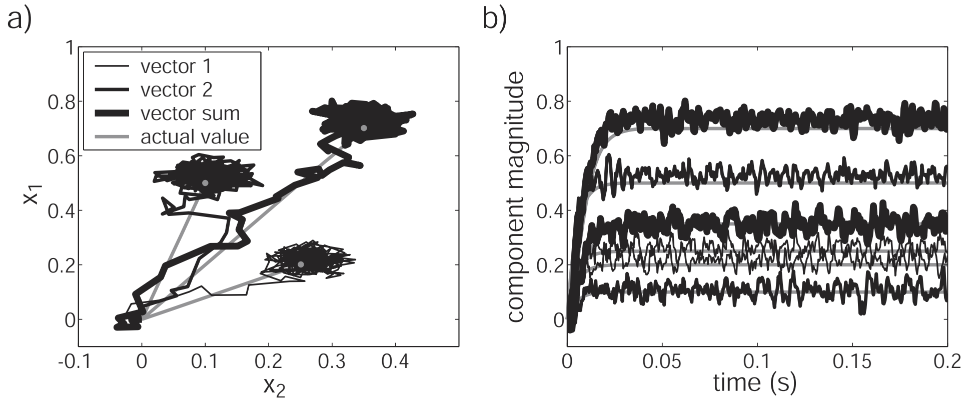


Figure 6.5
Results of vector addition where $\mathbf{x} = [.25\ .2]$ (vector 1), $\mathbf{y} = [.1\ .5]$ (vector 2) using 1200 spiking LIF neurons (400 for each population). In each case, the desired answer is shown as a grey line and the population decoding is shown as a black line. The thickest line, $\mathbf{z}$, is the sum of $\mathbf{x}$ and $\mathbf{y}$. a) Vector addition in state space. b) Vector addition over time, with each component of each vector drawn separately.

6.3 Nonlinear transformations

As mentioned previously, the impressively complex behavior exhibited by neural systems is unlikely to be fully explained by linear transformations alone. Rather, there is abundant evidence that nonlinear operations are central to neural transformations (see Koch and Poggio 1992 and Mel 1999 for a partial review).[4] For example, the responses of neurons in the visual pathway of the locust are best predicted by the *product* of the angular image velocity and a function of the angular size of the object on the locust's retina (Koch 1999, p. 347; Koch and Segev 2001). As well, in primates there is evidence that some coordinate transformations in parietal areas are a result of finding the *product* of representations of eye and head positions (Zipser and Andersen 1988). These are both examples of multiplication, one of the simplest but most important nonlinear transformations. Nonlinearities, then, span the phylogenetic tree.

The reason that nonlinearities like multiplication are so important, is that they vastly increase computational power and storage capacity (Giles and Maxwell 1987). Koch

4 Here is a non-comprehensive list of examples of multiplication in neurobiological systems: disparity tuning in binocular visual cells (Ohzawa et al. 1990), responses to illusory contours (Peterhans and von der Heydt 1989), responses to periodic gratings (von der Heydt et al. 1991), velocity tuning (Nowlan and Sejnowski 1993), various kinds of motion perception (Koch 1999; Koch and Poggio 1992; Nowlan and Sejnowski 1993; Adelson and Bergen 1985; Mel 1999), direction selectivity (Adelson and Bergen 1985), shape selectivity with spatial invariance (Fukushima et al. 1983), and modulation by selective attention (Anderson and Van Essen 1987; McAdams and Maunsell 1999).

(1999) points out that such nonlinearities are actually *necessary* for doing true information processing (p. 346). If we suppose that evolution is capable of finding near-optimal, or even good solutions to information processing problems, then we would expect to find nonlinearities in nervous systems trying to solve those problems. As a result, we take it to be a 'hard constraint' on theories of neurobiological systems that they be able to incorporate multiplication.

It may seem obvious, given the nonlinear responses of the spike generating mechanisms in neurons themselves, that nonlinearities are common in neurobiological systems. However, it is not immediately evident that neurons can easily generate *highly* nonlinear functions (like multiplication) of their input currents since the spike generating nonlinearities often exhibit a fairly linear response (and tend, if anything, to be sublinear). By far the majority of neural models assume that the soma current is simply a sum of the synaptic currents. If this is true, multiplication must be a network behavior, rather than one found in single cells. However, there is mounting evidence that soma currents are nonlinear functions of dendritic inputs (Mel 1994; Mel 1999). So, there are two options for explaining multiplication; as a network property *or* as a cellular property (or both, perhaps). Even if we have tendencies to think one explanation is more likely than the other, it is a matter of debate as to how either networks (Salinas and Abbott 1996) *or* cells (Mel 1994) can multiply inputs—even though there is ample evidence that multiplication occurs *somewhere* (see footnote 4).

In the next section we consider both the possibility that multiplication is a network property and the possibility that it is a cellular property. We show how both possibilities can be accounted for by the framework we are presenting. We cover both possibilities to show the flexibility of the framework and also to demonstrate the manner in which it can be sensitive to experimental results (since such results will presumably, on a case by case basis, determine if a given nonlinearity is a cellular or network effect). We put multiplication to *use* in a model of vestibular function in section 6.5.

6.3.1 Multiplying two variables

In this section, we discuss three different ways to understand multiplication in neurobiological systems. First, we consider the possibility that there is a dendritic mechanism that *directly* implements multiplication (or a suitable variant). In this case, we can include multiplication in our framework by simply permitting multiplication of the inputs to a neuron. Second, we consider how to implement multiplication at a network level using the framework more directly. And, third, we consider the possibility that the network level analysis can be embedded directly into the dendrites of a neuron, so as to provide a novel explanation for multiplication at the cellular level.

There have been many proposals for how multiplication can be implemented at the cellular level, usually in the dendritic tree (Mel 1994; Koch and Poggio 1992). These include, for example, the suggestion that dendrites have logarithmic response properties and thus multiplication could be done as a sum of these logarithms (Koch and Poggio 1992; Tal and Schwartz 1997; Durbin and Rumelhart 1989).[5] Or, alternatively, that shunting inhibition could provide the necessary nonlinearity (Poggio 1981; Torre and Poggio 1978). Or, perhaps best known, that presynaptic spike 'coincidence detection' can be used to implement multiplication in dendrites (see Mel 1994, pp. 1058–9 for relevant references; Stevens and Zador 1998). Coincidence detection has long been known to be able to underwrite multiplication in spiking neurons (Küpfmüller and Jenik 1961). This is becuase a receiving neuron with coincidence detection will spike at a frequency proportional to the product of two input spike train frequencies.

We have elsewhere presented a simple model which employs coincidence detection directly (Hakimian et al. 1999). In that model, we begin by defining the representations for x, y, and z as in (6.14) and (6.15) above. The transformation we wish to implement is simply $z = x \cdot y$, so we can write the firing rates for z as

$$\begin{aligned} c_k(x \cdot y) &= G_k\left[\alpha_k \tilde{\phi}_k(x \cdot y) + J_k^{bias}\right] \\ &= G_k\left[\alpha_k\left(\tilde{\phi}_k \sum_i a_i(x)\phi_i^x \cdot \sum_j b_j(y)\phi_j^y\right) + J_k^{bias}\right] \\ &= G_k\left[\sum_{i,j}\omega_{kij}a_i(x)b_j(y) + J_k^{bias}\right], \end{aligned}$$

where $\omega_{kij} = \alpha_k\tilde{\phi}_k\phi_i^x\phi_j^y$. While it is an easy matter to multiply firing rates in a computer model (since they are real valued variables), here we are interested in implementing this network using *spiking* neurons. Therefore we need to employ a biologically plausible mechanism like coincidence detection. Writing the output of the the a_i and b_j populations as spikes gives

$$a_i(x)b_j(y) = \sum_{n,m} h_i(t - t_{in})h_j(t - t_{jm}).$$

To show that this can be implemented by coincidence detection, let us suppose that the filters $h(t)$ are narrow Gaussians.[6] We can write this multiplication as (see appendix D.1

5 Recall that $xy = e^{\ln(x)+\ln(y)}$.

6 Note that the only real constraint on the choosing $h(t)$ to get the desired result is that it be narrow relative to the distance between neighboring spikes. Choosing Gaussians makes the analysis mathematically simpler.

for a complete derivation)

$$\begin{aligned} a_i(x)b_j(y) &= \sum_{n,m} e^{-(t-t_{in})^2/2\Delta^2} e^{-(t-t_{jm})^2/2\Delta^2} \\ &= \sum_{n,m} e^{-[(t-t_{in})^2+(t-t_{jm})^2]/2\Delta^2} \\ &= \sum_{n,m} e^{-\left[2\left(t-\frac{t_{in}+t_{jm}}{2}\right)^2+\frac{1}{2}(t_{in}-t_{jm})^2\right]/2\Delta^2} \\ &= \sum_{n,m} e^{-2\left(t-\frac{t_{in}+t_{jm}}{2}\right)^2/2\Delta^2} e^{-\frac{1}{2}(t_{in}-t_{jm})^2/2\Delta^2}, \end{aligned} \tag{6.18}$$

where Δ^2 is the (small) variance of the Gaussian filter, $h(t)$. Equation (6.18) implements a kind of coincidence detection. Notice that in this product, the first Gaussian is centered on the mean of the possibly coincident spike times and the second is peaked only if $t_{in} \approx t_{im}$, otherwise it is near zero everywhere. So, the resulting filtered and multiplied signal consists of narrow Gaussians near the mean of t_{im} and t_{in} only if they are approximately equal; i.e., if they are coincident. So, this kind of coincidence detection can implement the multiplication we need. Many researchers have thought that the main nonlinearity in dendritic trees must be this kind of coincidence (Softky and Koch 1995; Stevens and Zador 1998; see also Mel 1994, pp. 1058–9 for relevant references).

However, the above analysis also shows a weakness in the coincidence hypothesis. Notice that in order to implement multiplication in this fashion, each spike from a neuron in population a must be compared to *each and every* spike in population b. Because coincidence detection, if it takes place, must take place between neighboring synapses, the receiving population, c, has to have N_b synapses (where N_b is the number of neurons in population b) near each of the N_a synapses. So, for any cell in the b population, it has to make N_a connections to each cell in the c population. This kind of 'over-full' connectivity is not at all neurobiologically realistic.[7]

Notice also that although we have assumed coincidence detection is the mechanism underlying multiplication in this example, we could equally have assumed one of the other possible mechanisms, so long as the somatic current is a product of the post-synaptic currents. Perhaps there are other implementations that do not have biologically unrealistic consequences—perhaps even implementations of coincidence detection. Nevertheless, it should be clear such solutions to the problem of implementing multiplication are indepen-

7 By 'over-full' we mean to note that this needs more connections than a fully connected network. A fully connected network has one connection from each a_i and b_j to each c_k. This network requires $N_a - 1$ *more* connections from each b_j to each c_k. Note that even full connectivity is unrealistic in real nervous systems.

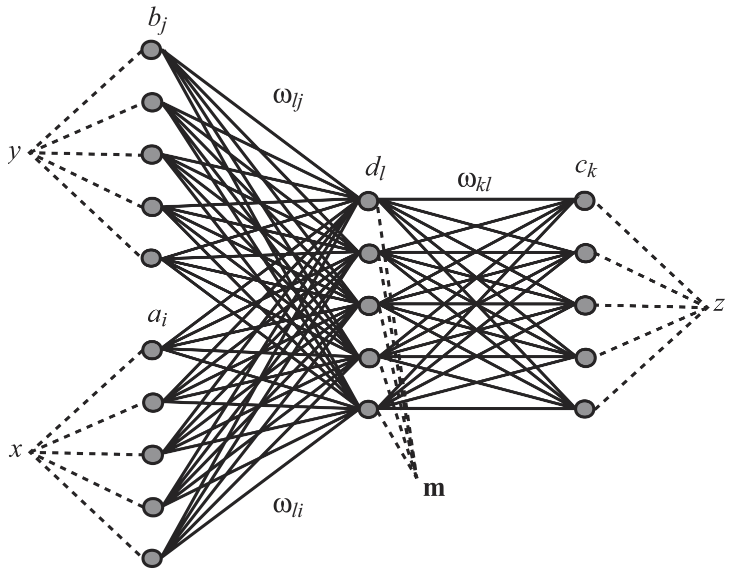


Figure 6.6
Schematic network for performing multiplication with LIF neurons. This is like the network for scalar addition (in figure 6.3), with the inclusion of a middle, or 'hidden', layer, $\mathbf{m}$.

dent of any of the central assumptions of our framework. This is both good and bad. It is good because it means that any actual dendritic mechanism that implements multiplication can be merged with the framework. It is bad because it also means that there is no explanation of such mechanism to be found in the assumptions underlying the framework. In contrast, the two subsequent solutions to implementing multiplication, which do not have the undesirable consequences of this kind of coincidence detection, are more directly tied to the assumptions underlying our approach.

Let us now consider how to implement multiplication in a network, without assuming nonlinearities in the dendrites. Perhaps surprisingly, most solutions for multiplication have been cellular solutions (although see Salinas and Abbott 1996). This is surprising because the neurons in most computational models do not have nonlinear dendrites.

In order to implement multiplication in a network, we define our representations of x, y, and z as before. However, we also include a 'middle layer', $\mathbf{m}$, (or 'hidden layer') which is a vector representation that combines the two scalar inputs such that their product can be extracted (i.e., decoded). We define the representation for $\mathbf{m}$ as we did before in equations (6.16) and (6.16) for vectors. The resulting network is shown in figure 6.6.

The transformation that relates $\mathbf{m}$ to the input values is simply $m_1 = x$ and $m_2 = y$. That is, the components of $\mathbf{m}$ are equal to the inputs. The transformation that relates $\mathbf{m}$ to the output variable is $z = f(\mathbf{m}) = m_1 \cdot m_2$. In order to implement this second transformation, we need to known how to decode $f(\mathbf{m})$, so we must also find the transformational decoder ϕ_l^f, where

$$\hat{f}(\mathbf{m}) = \sum_l d_l(\mathbf{m})\phi_l^f.$$

We can do this, as usual, by minimizing the error

$$E_f = \left\langle \left[f(\mathbf{m}) - \hat{f}(\mathbf{m})\right]^2 \right\rangle_{\mathbf{m}}.$$

We can now find the appropriate expressions for the connection weights in the network. We do this, as before, by substituting our decoding rules into the encoding rules of the population that represents the output of the transformation. In this case, we have two sets of transformations to consider. First, we must relate the input to the middle layer:

$$\begin{aligned} d_l(\mathbf{m} = [x\, y]) &= G_l\left[\alpha_l \left\langle \tilde{\boldsymbol{\phi}}_l \mathbf{m} \right\rangle + J_l^b\right] \\ &= G_l\left[\alpha_l \left(\tilde{\phi}_l^{m_1}\hat{x} + \tilde{\phi}_l^{m_2}\hat{y}\right) + J_l^b\right] \\ &= G_l\left[\sum_i \omega_{li}^{m_1} a_i(x) + \sum_j \omega_{lj}^{m_2} b_j(y) + J_l^b\right], \end{aligned}$$

where $\omega_{li}^{m_1} = \alpha_l \phi_i^x \tilde{\phi}_l^{m_1}$ and $\omega_{lj}^{m_2} = \alpha_l \phi_j^y \tilde{\phi}_l^{m_2}$. Next, to find the weights for the second transformation, we proceed along the same lines:

$$\begin{aligned} c_k(f(\mathbf{m})) &= G_k\left[\alpha_k \left(\tilde{\phi}_k f(\mathbf{m})\right) + J_k^b\right] \\ &= G_k\left[\alpha_k \left(\tilde{\phi}_k \sum_l d_l(\mathbf{m})\phi_l^f\right) + J_k^b\right] \\ &= G_k\left[\sum_l \omega_{kl} d_l(\mathbf{m}) + J_k^b\right], \end{aligned}$$

where $\omega_{kl} = \alpha_k \tilde{\phi}_k \phi_l^f$. As usual, this transformation can be directly implemented in a noisy spiking network (see figure 6.7).

Essentially, we have derived the middle (or 'hidden') layer of a standard artificial neural network (ANN) architecture. In the ANN literature, it is well known that such an architecture permits nonlinear and linear functions of the input to be learned (usually

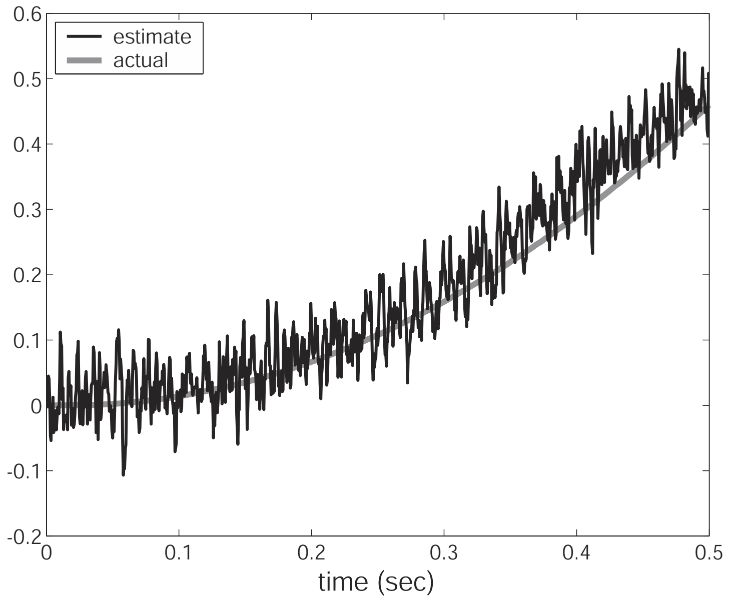


Figure 6.7
Results of multiplication by the spiking network shown in 6.6. There are 200 LIF neurons in each of the x, y, and z populations and 400 in the middle layer, $\mathbf{m}$. The network is driven by ramp input to x and y. Shown here is the decoded output variable, z, and the correct answer (grey line).

by backpropagation). Here we have shown how to derive the weights for implementing the desired nonlinear function. Better yet, we have done so assuming neurobiologically reasonable neurons (unlike most ANNs). And, since we have done so with well-understood principles, we know how the network works and we can quantify its limitations (see section 7.3 for a precise description of the principles governing this sort of network).

However, the neurophysiological evidence suggests that multiplication is not implemented (or at least is not *only* implemented) at the network level (Mel 1999; Mel 1994; Koch and Poggio 1992). It has been reasonably well-established that dendrites tend to have nonlinear interactions with other nearby dendrites (Mel 1999; Koch 1999; Bugmann 1991). If a cell has such nonlinear dendritic interactions, it should be able to perform something akin to multiplication (although the precise mechanism is not yet well understood). It has been noted that such a cell (i.e., one that has nonlinear dendrites) has the computational power akin to two-layer ANNs (Mel 1999; McKenna et al. 1992). So, it is natural to think that we could implement the final two layers of the network in figure 6.6, in a single cell.[8]

8 This is related to the historically common suggestion that neurons with nonlinear dendritic trees can be

This amounts to embedding the middle layer in this network, $\mathbf{m}$, into the dendritic trees of cells of the z population.

In doing so, we can take advantage of the flexibility of the framework. In particular, we could think of the $G_l\left[\cdot\right]$ functions as being a nonlinearity *internal* to the dendrites. However, we may not want the $G_l\left[\cdot\right]$ function to be a spiking nonlinearity (although there is some evidence of dendritic spikes; Schwindt and Crill 1998; Llinás and Sugimori 1980). Rather we want to pick whichever $G_l\left[\cdot\right]$ is relevant to the nonlinearity actually observed in dendrites. Only small variations to the 'standard' interpretation of the framework would result. For example, the connection weights that we find would not be considered synaptic, but rather they would be mapped to the efficacy with which a somatic current is produced by a particular dendritic current. Other than such interpretive changes, the relevant derivations can proceed exactly as before.

The idea of embedding this kind of network structure into dendrites follows closely standard notions of how to understand nonlinearities in neurons. For instance, the notion of a "multiplicative subunit" (Softky and Koch 1995, p. 883), or other "dendritic subunit" (Mel 1994, p. 1061), has often proved useful in understanding dendritic nonlinearities. Such subunits are a natural analog to the nodes in the middle layer of the network solution proposed above. So, thinking of this kind of network structure as being embedded into dendritic trees does not violate standard notions of the organization of dendritic function. In fact, the only major consequence for our understanding of neural systems that embedding nonlinearities into dendrites has is that it would allow a lot more to be done with a lot fewer neurons.

Before turning our attention to a neurobiological example that incorporates both linear and nonlinear transformations, we digress briefly to discuss an important problem in neural simulation that is often avoided by modelers. However, we have now developed the tools necessary for tackling this difficulty—getting rid of negative weights.

6.4 Negative weights and neural inhibition

Our approach, like most, does not distinguish between positive and negative elements of the connection weight matrix, ω_{ij}. However, in real neurobiological systems, there are significant differences between neural excitation (which correspond to positive elements of ω_{ij}) and neural inhibition (which correspond to negative elements of ω_{ij}). In fact, there is a specific subclass of neurons, called 'interneurons' or 'nonspiny neurons', that use GABA (or glycine) neurotransmitters to inhibit the firing of postsynaptic neurons.

described as implementing large networks of nonlinear operations (de Nó and Condouris 1959; Torre and Poggio 1978; Rall and Segev 1987; Koch et al. 1983; Zador et al. 1992).

However, projections to these interneurons are only excitatory. So, all projections from pyramidal cells have positive connection weights. As well, all projections from inhibitory neurons are negative. So, in neural systems, there is generally *not* a mixture of positive and negative connection weights; all connections matrices are either purely positive or purely negative (see figure 6.8). The 'problem of negative weights' is thus one of understanding how interneurons take the place of the negative elements of the connection weight matrix, ω_{ij}.

It is natural to think of the interneurons as 'inverting' the input they receive from one local population and projecting it to a subsequent population that would have otherwise received a negative weight (see figure 6.8).[9] So, the problem of understanding how to realistically implement connection matrices in a neurobiological model depends on devising a principled way of distinguishing negative from positive input and determining two new sets of connection weights (from the input population to the inhibitory neurons and from the inhibitory neurons to the output population). The reason this challenging problem is often avoided by modelers is that the 'obvious' solution—to insert an 'inverting' interneuron for every negative weight—is entirely biologically unrealistic.

6.4.1 Analysis

The general strategy we adopt for finding the appropriate weight matrices is to augment the representation, $\mathbf{x}$, in the a_i population in such a way that the bias to the resulting currents in the b_j population can be canceled out by input from the interneurons, c_k. Of course, the augmentation to the a_i representation must be sensitive to what is being represented in order to provide the appropriate bias. We can think of the augmentation as an extra dimension added to the $\mathbf{x}$ representation which is some function of $\mathbf{x}$, $f(\mathbf{x})$. We call this function the 'biasing function'.

To begin, we need to determine an appropriate form for this biasing function. We know that whatever biasing function we choose, we should be able to decoded it with only positive decoding weights. One way to find such a function is to simply set all of the decoders to 1 and determine what function is decoded using the $a_i(\mathbf{x})$ as coefficients. By doing this, we find the function that is decoded is approximately $1+|\mathbf{x}|^2$, or, more generally $K_1 + K_2|\mathbf{x}|^2$. So, adding and later subtracting a function of this form to the signal going from the a_i population to the b_j population should allow all of the weights to be positive.

In order to add this biasing function, we must define the decoding for it in terms of the a_i activities, which we can do as usual:

$$\hat{f}(\mathbf{x}) = \sum_i a_i(\mathbf{x})\phi_i^f.$$

9 Inhibitory interneurons almost always project locally (with the exception of Purkinje cells).

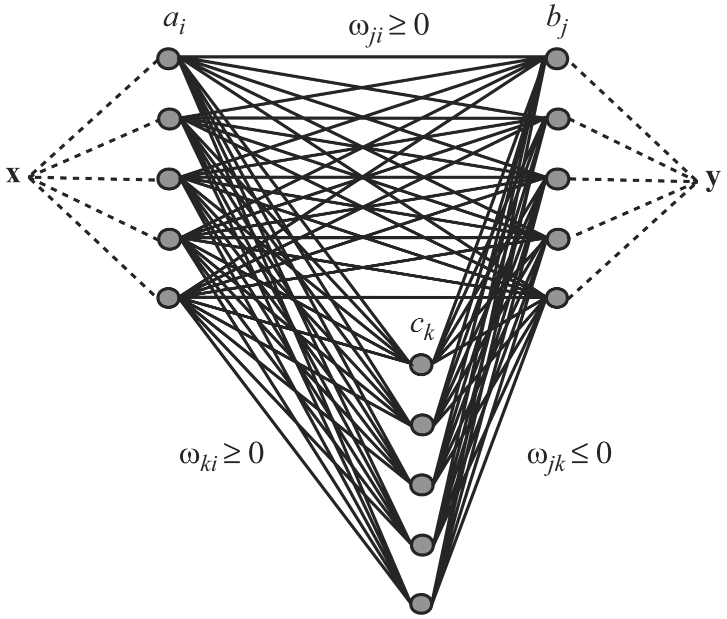


Figure 6.8
Introduction of inhibitory interneurons into a simple feed-forward communication channel (figure 6.1 depicts the original circuit).

We can now include this function as another dimension in the space being sent from a_i to b_j. If we now add this function, $f(\mathbf{x})$, into the encoding for the currents, J_j, in the b_j population, we have

$$\begin{aligned} J_j(\mathbf{x}) &= \alpha_j \left(\left\langle \tilde{\boldsymbol{\phi}}_j^{\mathbf{y}} \hat{\mathbf{x}} \right\rangle_n + \tilde{\phi}_j^f \hat{f}(\mathbf{x}) \right) + J_j^{bias} \\ &= \sum_i \left[\alpha_j \left(\left\langle \tilde{\boldsymbol{\phi}}_j^{\mathbf{y}} \boldsymbol{\phi}_i^{\mathbf{x}} \right\rangle_n + \tilde{\phi}_j^f \phi_i^f \right) \right] a_i(\mathbf{x}) + J_j^{bias} \\ &= \sum_i \omega_{ji} a_i(\mathbf{x}) + J_j^{bias}, \end{aligned}$$

where we have distinguished the $f(\mathbf{x})$ component from the other components (i.e., $\mathbf{x}$) in the vector space represented by the a_i neurons. The challenge now becomes to make the ω_{ji} elements positive. In order to do this, we need to find the appropriate encoders, $\tilde{\phi}_j^f$, and decoders, ϕ_i^f, for f such that $\omega_{ji} \geq 0$. The encoders and decoders for $\mathbf{x}$ are determined as usual.

In fact, there is a well-defined relation between the encoders and decoders for f to ensure $\omega_{ji} \geq 0$. Specifically, since

$$\alpha_j \left(\left\langle \tilde{\boldsymbol{\phi}}_j^{\mathbf{y}} \boldsymbol{\phi}_i^{\mathbf{x}} \right\rangle_n + \tilde{\phi}_j^f \phi_i^f \right) = \omega_{ji} \geq 0,$$

then

$$\frac{-\left\langle \tilde{\boldsymbol{\phi}}_j^{\mathbf{y}} \boldsymbol{\phi}_i^{\mathbf{x}} \right\rangle_n}{\phi_i^f} \leq \tilde{\phi}_j^f,$$

which will be satisfied if

$$\tilde{\phi}_j^f = \max_i \left(\frac{-\left\langle \tilde{\boldsymbol{\phi}}_j^{\mathbf{y}} \boldsymbol{\phi}_i^{\mathbf{x}} \right\rangle_n}{\phi_i^f} \right). \tag{6.19}$$

Let us consider this equation in more detail for a moment. What we want to do is ensure we have no negative weights ω_{ji}. For the original circuit (with no interneurons), these weights are simply $\left\langle \tilde{\boldsymbol{\phi}}_j^{\mathbf{y}} \boldsymbol{\phi}_i^{\mathbf{x}} \right\rangle$. The intuitive way to eliminate any negative weights in this expression is to add a bias to all of them equal to the value of the largest negative one. This is the purpose of the product $\tilde{\phi}_j^f \phi_i^f$, although what is added will depend on which neuron, i, we are considering. To figure out what we must add to the original matrix, we need to identify the largest negative weight coming from neuron i (this is done in the numerator of (6.19)). Now we must know either the encoders or decoders for $f(\mathbf{x})$ so we can determine the other. But, whichever we know, *they must be positive*. If they were not, then we might incorrectly identify the largest negative weight since the negative signs would cancel. Because we have a means of determining the decoders (below), this constraint will apply to them.

Let us now consider the decoders in more detail. Because the function $f(\mathbf{x})$ is not completely defined (i.e., K_1 and K_2 are not determined), we allow the decoders to have the following form:

$$\phi_i^f = K_1 \phi_i^{f_1} + K_2 \phi_i^{f_2}.$$

We can then solve, as usual, for the $\phi_i^{f_1}$ and $\phi_i^{f_2}$ decoders. Effectively, the first of these decoders can be used to decode 1 and the second can be used to decode $|\mathbf{x}|^2$ using the a_i activities. However, we still must find K_1 and K_2 in order to fully define the decoder ϕ_i^f.

Given the considerations regarding (6.19) above, we know that the decoders must be positive. That is

$$K_1 \phi_i^{f_1} + K_2 \phi_i^{f_2} > 0$$

or, equivalently,

$$\mathbf{K}\boldsymbol{\phi}_i^{f_{1,2}} > 0, \tag{6.20}$$

where $\mathbf{K} = [K_1, K_2]$ and $\boldsymbol{\phi}_i^{f_{1,2}} = [\phi_i^{f_1}, \phi_i^{f_2}]$. So, we can consider the problem we need to solve to be one of finding a unit vector, $\mathbf{K}$ that points in some direction, θ, in this two-dimensional space for which (6.20) is satisfied (see figure 6.9). If we find this vector, then $\mathbf{K} = [\sin(\theta), \cos(\theta)]$. In fact, we will only be able to satisfy (6.20) if all of the vectors $\boldsymbol{\phi}_i^{f_{1,2}}$ lie in a half-plane, i.e., $\theta^{max} - \theta^{min} < \pi$.

Our numerical experiments show that this condition is almost always satisfied for randomly generated neuronal ensembles at any dimension.[10] In general, there will be a range of angles, $[\theta^{max}, \theta^{min}]$, for which this condition is met. A natural choice in this range is the mean, which works well, but we have found that it is often better to weight the mean in favor of the constant term; i.e., we choose

$$\theta = 0.75\left(\theta^{max} - \frac{\pi}{2}\right) + 0.25\left(\theta^{min} + \frac{\pi}{2}\right). \tag{6.21}$$

Figure 6.9 shows an example of finding $\mathbf{K}$ for a population of 200 neurons. Now that we have completely defined ϕ_i^f, we can find $\tilde{\phi}_j^f$ using (6.19). As a result, our biasing current is now defined.

The next step is to remove that biasing current from the determination of the current in the b_j neurons. Recall that the current is introduced only so we can find positive weights. Once we have found such weights, we need to remove it to preserve the original transformation. The additional bias current is the current introduced into the b_j neurons by the biasing function, i.e.,

$$\begin{aligned} J_j^f(\mathbf{x}) &= \alpha_j \tilde{\phi}_j^f f(\mathbf{x}) && (6.22)\\ &= \alpha_j \tilde{\phi}_j^f \left(K_1 + K_2|\mathbf{x}|^2\right) && (6.23)\\ &= \alpha_j \tilde{\phi}_j^f K_1 + \alpha_j \tilde{\phi}_j^f K_2 |\mathbf{x}|^2. && (6.24) \end{aligned}$$

Subtracting these currents from the b_j currents gives the total current to each b_j neuron as

$$\begin{aligned} J_j(\mathbf{x}) &= \sum_i \omega_{ji} a_i(\mathbf{x}) + J_j^{bias} - \alpha_j \tilde{\phi}_j^y f(\mathbf{x}) \\ &= \sum_i \omega_{ji} a_i(\mathbf{x}) + J_j^{bias} - \alpha_j \tilde{\phi}_j^f K_1 - \alpha_j \tilde{\phi}_j^f K_2 |\mathbf{x}|^2. \end{aligned}$$

Because the first term in (6.24) is constant, it can be added directly into the J_j^{bias} current

10 The violations we have seen are limited to one or two neurons in populations of about 200 neurons.

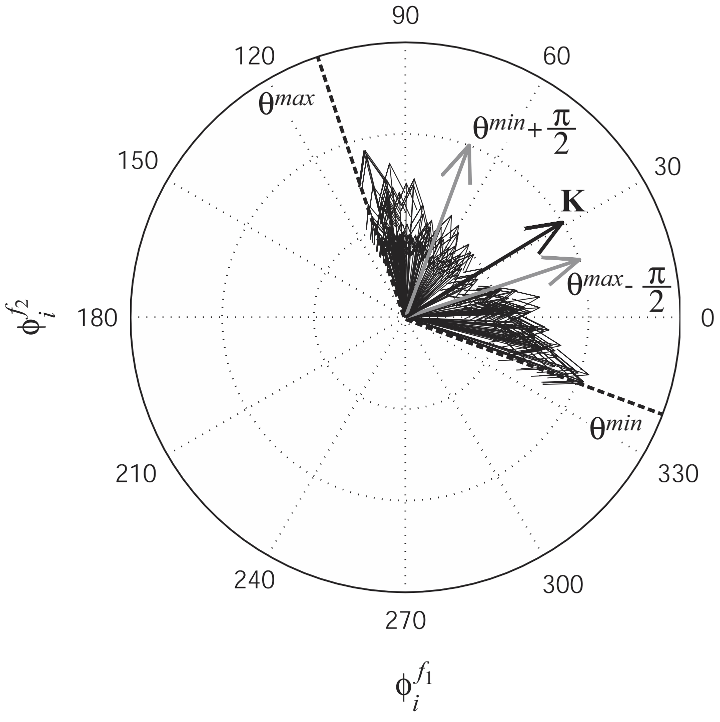


Figure 6.9
Finding the decoding vectors to remove negative weights. The preliminary decoding vectors $\boldsymbol{\phi}_i^{f_{1,2}}$ are the many thin vectors. Because these lie in a half-plane (i.e., $\theta^{max} - \theta^{min} < \pi$), it is possible to find $\mathbf{K}$, which has a positive dot product with all of them. Technically, $\mathbf{K}$ can lie anywhere between $\theta^{max} - \frac{\pi}{2}$ and $\theta^{min} + \frac{\pi}{2}$, but that found using (6.21), and pictured here, results in a smaller range of dot product values.

already present in the b_j neurons, i.e.,

$$J_j^{bias_{new}} = J_j^{bias} - \alpha_j \tilde{\phi}_j^f K_1.$$

The second term, however, is a function of $\mathbf{x}$ and thus must be provided from another input. This is a natural role for the interneurons.

In order to provide the appropriate input, the interneurons, c_k, must be provided with the $f(\mathbf{x})$ signal. Note that, regardless of the dimensionality of $\mathbf{x}$, this is always a one-dimensional signal. Furthermore, because these neurons only need to represent the quadratic component of the positive definite function $f(\mathbf{x})$, the ensemble can consist of only positively-sloped (i.e., 'on') neurons. As a result, the encoding vectors, $\tilde{\phi}_k$, will all be equal to 1. The soma current for the c_k neurons is thus

$$\begin{aligned} J_k(\mathbf{x}) &= \alpha_k \hat{f}(\mathbf{x}) + J_k^{bias} \\ &= \alpha_k \sum_i \phi_i^f a_i(\mathbf{x}) + J_k^{bias} \\ &= \sum_i \omega_{ki} a_i(\mathbf{x}) + J_k^{bias}. \end{aligned}$$

As before, the bias currents to the interneurons can be readjusted to remove the effects of the constant part of $f(\mathbf{x})$:

$$J_k^{bias_{new}} = J_k^{bias} - \alpha_k K_1.$$

This ensures that the net signal to these neurons includes on the quadratic component. This component can then be decoded from the c_k neurons using only positive decoders:

$$\widehat{K_2|\mathbf{x}|^2} = \sum_k \phi_k c_k(f(\mathbf{x})).$$

This can then be used to cancel out for the remaining bias current in (6.24). As a result, the total current to the b_j population from the a_i and c_k populations is

$$J_j(\mathbf{x}) = \sum_i \omega_{ji} a_i(\mathbf{x}) + J_j^{bias_{new}} - \sum_k \omega_{jk} c_k(f(\mathbf{x})),$$

where $\omega_{jk} = \alpha_k \tilde{\phi}_j^f \phi_k$. We have thus constructed a circuit which acts as a communication channel where all of the connectivity weight matrices are positive. As figure 6.10 demonstrates, this circuit works as expected.

6.4.2 Discussion

In essence, our strategy here has been to first minimize the amount of negative current needed to drive the b_j population (via the constant bias) and then to use the interneurons to provide whatever negative current is still needed. Notably, it is possible to skip the first step (minimizing the needed negative current) and use the techniques above for finding the negative current directly. However, taking this more direct route results in demands on the interneurons that parallel those on the excitatory neurons. This is generally not what is seen in mammalian cortex. Rather, the large majority of projections from pyramidal (i.e., excitatory) cells are to other pyramidal cells (Thomson and Deuchars 1997), suggesting that interneurons have significantly different (in some ways muted) demands placed on them.

The results in figure 6.10 suggest that the interneurons are playing a similar role here. When the interneurons are removed, the circuit actually performs quite well over the mid-

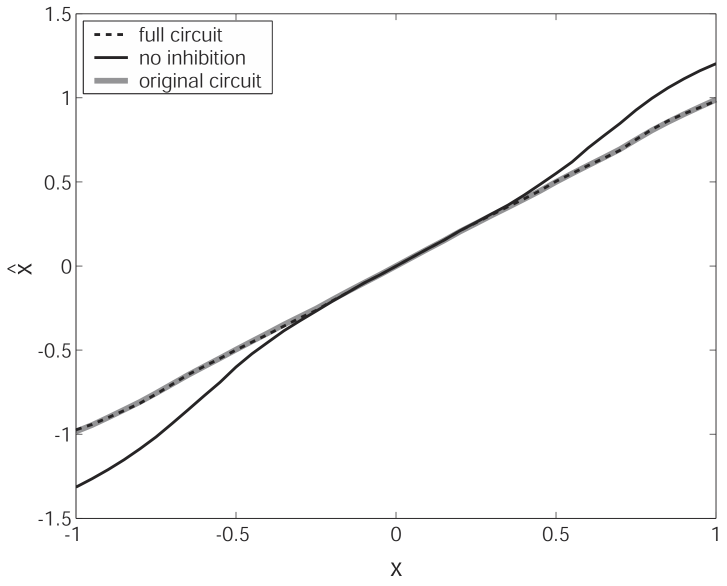


Figure 6.10
Communication channel using only positive weights. The topology of this network is shown in figure 6.8. This network performs quite well (RMSE=0.0085) and comparably to the original circuit with negative weights (RMSE=0.0044). When the inhibitory interneurons are removed, the circuit performs much worse (RMSE=.16).

dle range (i.e., between ± 0.5), so the interneurons are largely not needed. Outside of this range, they act to linearize the response, but do not need to provide an extraordinary amount of input to the b_j population. As a result, the interneurons carry much less information, and play a muted role compared to the a_i population. So we have presented a slightly less straightforward, but more biologically realistic alternative for including interneurons in our circuits.[11]

It is also important that there is nothing in the previous derivation that assumes that the transformation between the a_i and b_j populations is an identity (as it is for a communication channel). Thus, these techniques can be used to construct such circuits for any transformation that we can define in neural systems.

And finally, this solution to the negative weights problem would not have been possible without a good understanding of neural transformation and neural representation. Thus, understanding transformations can help solve traditionally difficult theoretical problems. In the next section, we show how understanding transformations can help in a more practical manner; by allowing us to build realistic, large-scale models of complex behavior.

11 In fact, there are many possible solutions to this problem that we have investigated, but this is the simplest we have found that is biologically plausible.

6.5 An example: The vestibular system

Einstein (1945) noted that measurements of linear acceleration are ambiguous. In particular, if we only know our linear acceleration, we cannot tell if we are tilting in a gravitational field, or accelerating perpendicular to that field. In the vestibular systems of animals, ranging from fish to humans, there are organs called *otoliths* that serve to measure linear acceleration. Given Einstein's insight, we know that signals from these organs cannot be used to distinguish inertial acceleration from gravitational acceleration. However, animals quite successfully distinguish tilts in earth's gravitational field from horizontal linear accelerations. Naturally, the question arises as to how this distinction is made by neurobiological systems that rely on otoliths.

Indeed, precisely how this distinction is made is a matter of some debate in contemporary neuroscience (Angelaki et al. 1999; Telford et al. 1997; Merfeld et al. 1999; Snyder 1999). The two main competing hypotheses are that either additional sensory information (usually thought to be from the semicircular canals) is used (Guedry 1974; Mayne 1974) or that, in practice, because linear accelerations tend to result in high-frequency responses, and tilts tend to result in low-frequency responses, the appropriate filtering of the otolith signal can be used to distinguish tilts from linear accelerations (Paige and Tomko 1991). In a series of experiments on macaque monkeys, Angelaki et al. (1999) show that inactivation of the semicircular canals results in precisely the kinds of eye movement errors we would expect if only the otolith were being used to distinguish translational from tilt accelerations under those conditions. Similarly, Merfeld et al. (1999) show that reflexive eye movements in humans are most consistent with a combination of canal *and* otolith information. This kind of evidence strongly supports the contention that there is a combination of sensory signals, from the otolith and semicircular canals, that is used to determine appropriate motor responses to various kinds of acceleration (see also Angelaki and Dickman 2000; Wilson and Jones 1979).

Importantly, Angelaki et al. (1999) have suggested a specific series of transformations that can be used to determine an *unambiguous* translational acceleration given the otolith and semicircular canal signals. In this section, we present the results of our collaboration with these researchers to develop a detailed neurobiological model that implements these transformations using our framework (see also Eliasmith et al. 2002). This example is useful for two reasons. First, it encompasses the major themes of this chapter: linear transformations, nonlinear transformations, and modularization. Second, it shows how constructing models can lead to useful predictions about the properties of neurons in a modeled system. In this way, it clearly demonstrates the utility of collaborative efforts involving both experimental and theoretical neuroscientists.

6.5.1 System description

The vestibular system consists of peripheral vestibular labyrinth organs and a number of nuclei in the brainstem. The model we present encompasses both the labyrinths and the parts of the vestibular nucleus that receive direct projections from the labyrinths. In this section, we specify the high-level representations used by the system, and derive a mathematical description of the system using these high-level variables.

The vestibular labyrinths are located slightly behind the jaw in primates and are part of what is often called the "inner ear." The labyrinths are composed of two main parts, the semicircular canals and the otolith organs (see figure 6.11). The functioning of the semicircular canals has been described in section 2.6.1. As noted there, there is a significant, but opposite, bias in the sensitivity of velocities encoded by the canals on each side of the head (thus the left canals are more sensitive to leftward and backward rotations and the right canals are more sensitive to rightward and forward rotations; (Kelly 1991)). So, for the purposes of our model, we take the two complimentary sets of neurons to include populations that represent the components of angular velocity biased to the left, $\mathbf{\Omega}_L(t)$, and right, $\mathbf{\Omega}_R(t)$, sides respectively.

The other main sensory components of the labyrinth are the otolith organs. These are composed of two distinct parts, the saccular macula and the utricular macula. The two maculae are very similar in structure and function, although the utricular macula is oriented in an approximately horizontal plane, and the saccular macula is oriented in an approximately vertical (saggital) plane (see 6.11 and 6.12). Each macula is a mostly flat surface that is covered by many groups of hair cells that are embedded in a gelatinous sheet. Inside this sheet are tiny crystals of calcium salts which are the actual otoliths (Greek: "ear rocks"). Flexing these hair cells causes adjoining neurons to fire action potentials. Much like in the canals, the inertia of the otoliths during an acceleration causes the gelatinous sheet to move relative to the macula, thus bending the hair cells embedded in it. This bending of the hair cells induces the afferent neurons to fire in proportion to the linear acceleration that caused them to bend. Again, there is an ipsilateral bias in encoding vectors on each side of the head (Wilson and Jones 1979, p. 111). We thus take there to be two neural populations (left and right) each of which encodes accelerations on the horizontal and vertical planes, $\mathbf{A}_L(t)$ and $\mathbf{A}_R(t)$.

It is interesting to note that these two organs employ quite different ways of representing 3-dimensional vectors. The canals do so by (approximately) sensing each of the 3 orthogonal components equally, whereas the otoliths do so by sensing each of the possible directions in two planes (see figure 6.12). These different approaches place different implementational constraints on the neural system as a whole, and are important when it comes to defining the exact nature of the transformations that the resulting representations

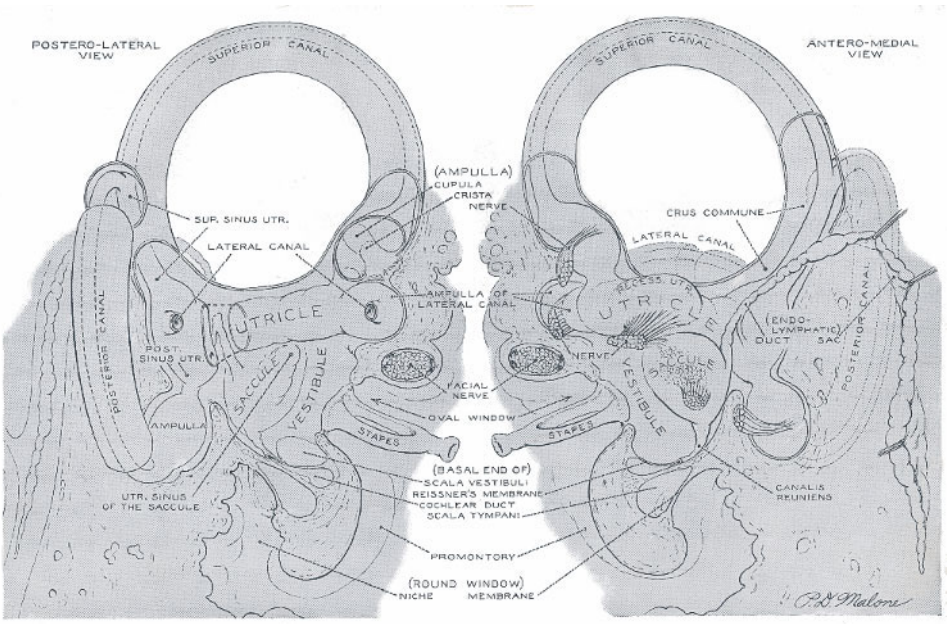


Figure 6.11
Human vestibular labyrinth organ. The two most obvious canals are those that sense ear to ear rotation. The others are cut away. The otoliths are located just below these two canals (from Brödel et al. 1946).

take part in. This again shows why it is important to look at the relevant neurobiology in some detail when attempting to construct a useful model of a neural system.

The purpose of modeling this system is to test an hypothesis about how the signals provided by the otoliths can be disambiguated. As noted in the introduction, there is evidence that the signal from the semicircular canals is used to extract the acceleration and head tilt information from the otolith signal, but how?

We know that the otolith signal, $\mathbf{A}$, is equal to the sum of the gravitational, $\mathbf{g}$, and inertial accelerations, $\mathbf{I}$:

$$\mathbf{A} = \mathbf{I} + \mathbf{g} \tag{6.25}$$
$$\mathbf{g} = \mathbf{A} - \mathbf{I} \tag{6.26}$$
$$\mathbf{I} = \mathbf{A} - \mathbf{g}. \tag{6.27}$$

We may now recall, from physics, that for a frame of reference rotating with angular

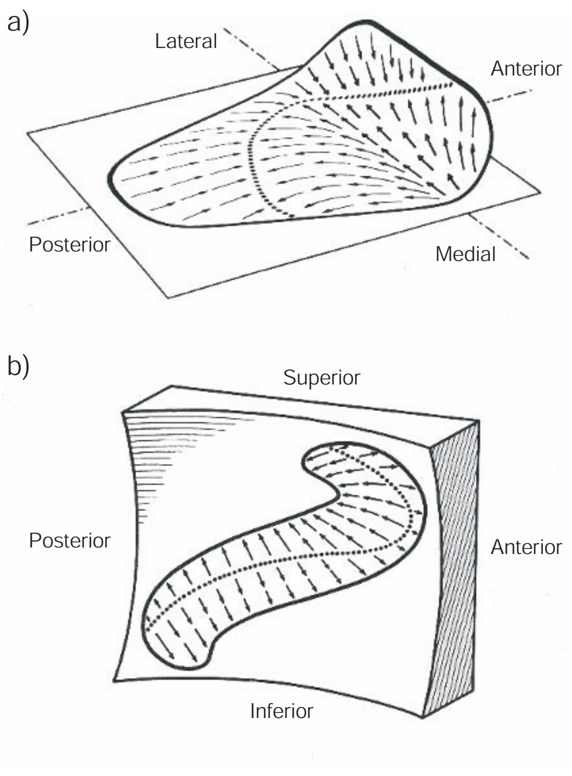


Figure 6.12
Otolith preferred direction vectors. a) Utricular macula (horizontal). b) Saccular macula (vertical). Between the utricle and saccule, all three directions are covered with preferred direction vectors (Fernández and Goldberg 1976a; Fernández and Goldberg 1976b; Fernández and Goldberg 1976c). (From Spoendlin 1966 © University of Pennsylvania Press, reproduced with permission.)

velocity Ω, and some vector $\mathbf{g}$, $\dot{\mathbf{g}} = \mathbf{\Omega} \times g + [\dot{\mathbf{g}}]_{rot}$.[12] In this problem, the rate of change of gravity is always zero in the inertial frame, so $[\dot{\mathbf{g}}]_{rot} = 0$. As a result, when we take the temporal derivative of 6.27, and substitute this expression for $\dot{\mathbf{g}}$, we have

$$\dot{\mathbf{I}} = \dot{\mathbf{A}} - \mathbf{\Omega} \times \mathbf{g}. \tag{6.28}$$

Substituting (6.26) into (6.28) gives

$$\dot{\mathbf{I}} = \dot{\mathbf{A}} + \mathbf{\Omega} \times [\mathbf{I} - \mathbf{A}]. \tag{6.29}$$

12 The notation $[\cdot]_{rot}$ indicates the vector observed from the rotating frame of reference.

As suggested in a recent paper by Angelaki et al. (1999), this provides a good, quantitative hypothesis about one function of the vestibular nucleus (see also Hess and Angelaki 1997).

We can now characterize the dynamics of the neural population that represents this inertial acceleration, using the first order Taylor expansion:[13]

$$\begin{aligned} \mathbf{I}(t+\tau) &\approx \mathbf{I}(t) + \tau\dot{\mathbf{I}}(t) && (6.30)\\ &= \mathbf{I}(t) + \tau\left[\dot{\mathbf{A}}(t) + \boldsymbol{\Omega}(t) \times [\mathbf{I}(t) - \mathbf{A}(t)]\right] && (6.31) \end{aligned}$$

Together, these equations suggest a neural circuit like that shown in figure 6.13. This diagram shows a particular modularization of the computation needed to determine the true inertial acceleration given the angular velocity, Ω, and the linear acceleration, $\mathbf{A}$.

We think that this modularization is a reasonable one given what is currently known about vestibular neuroanatomy. As already discussed, there is good evidence that angular velocity and linear acceleration are represented by the labyrinths. The afferent neurons from the labyrinths project to various sub-nuclei in the vestibular nucleus (Büttner-Ennever 1999). These projections are represented by the leftmost arrows in figure 6.13. In particular, Deiters' nucleus and the descending nucleus primarily receive otolith projections, whereas the medial and superior nuclei mainly receive canal projections. There are strong commissural fibers linking the two sides of the brain stem in these nuclei, thus allowing information from both the left and right labyrinths to be combined. The former nuclei are thus modeled by the a_i population and the latter nuclei by the b_j population in figure 6.13. However, there are parts of both Deiters' and the medial nuclei that receive input from *both* systems. As again shown in figure 6.13, we model this convergence of information from the canals and otoliths as taking place in the c_k population. It is here that we presume the central nonlinear transformation (the cross product) takes place. There is less evidence as to where the representation of inertial acceleration in the d_l population might reside in the vestibular system. This is one way in which future experiments may be able to challenge or confirm this modularization.

In the case of the otoliths, there is physiological evidence that a population of neurons in the vestibular nuclei have responses that are a combination of the afferent responses. Specifically, it has been shown that there is a broad spread of preferred direction vectors in the nuclei neurons throughout the 3-dimensional space, with only slight biases in favor of the horizontal and vertical planes (Wilson and Jones 1979, p. 165). In contrast, the vestibular nucleus neurons that the canals project to preserve the strong distinction between the three orthogonal directions of the canals, although there are physiological differences. In particular, the background firing rates of the vestibular nuclei neurons tend

13 Note that this is the same as approximating a derivative with Euler's method: $\frac{\Delta d(t)}{\Delta t} = \frac{d(t+\Delta t)-d(t)}{\Delta t}$.

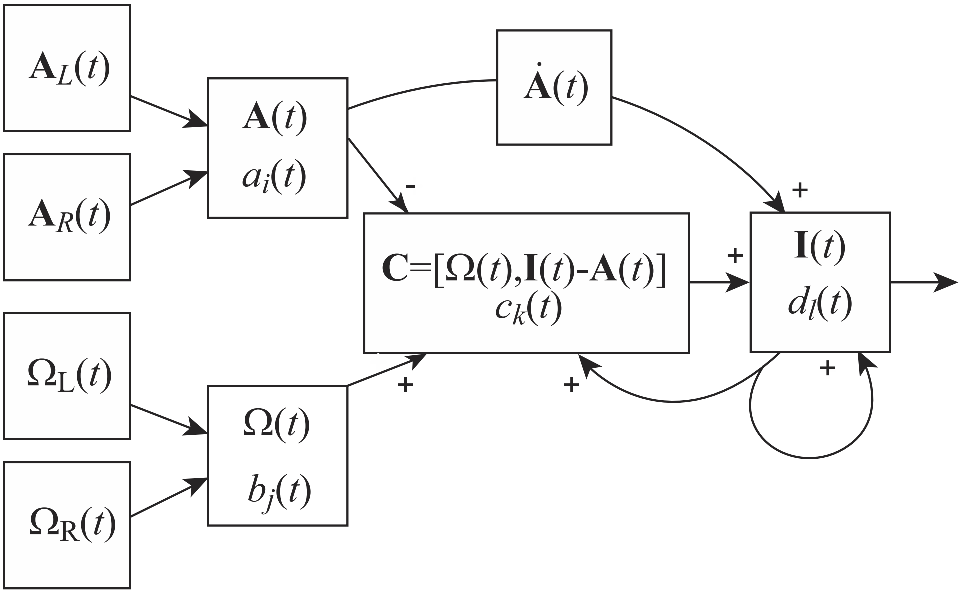


Figure 6.13
A high-level modularization of a vestibular circuit that can estimate the true inertial acceleration given the labyrinth inputs. Note that $\Omega_L(t)$, $\Omega_R(t)$ and $\Omega(t)$ each have 3 sub-populations (one for each preferred direction). Similarly, there are horizontal and vertical sub-populations in the $A_L(t)$ and $A_R(t)$ (although not in $A(t)$, as explained in the text).

to be about half that of the afferent neurons (Wilson and Jones 1979, p. 152). Given that the spontaneous activity in the afferents covers a range of 18–160 spikes/s with a mean of about 110 in primates, these vestibular nuclei neurons likely have rates between about 9–80 spikes/s with a mean of about 45 (Wilson and Jones 1979, p. 92). The background firing in the otolith neurons are similarly related, although they tend to be about 30% lower in both cases (Fernández and Goldberg 1976a).

We now have a relatively complete characterization of the tuning curves of cells encoding $\mathbf{A}(t)$ and $\mathbf{\Omega}(t)$, but we have said little about the cells involved in computing the cross product. Anatomy suggests, and physiology bears out, that there are strong convergent projections of both the otolith and canal systems within the vestibular nucleus (Wilson and Jones 1979, p. 166; Curthoys and Markham 1971; Searles and Barnes 1977; Büttner-Ennever 1999). This kind of convergence could support the sorts of transformations necessary to compute the inertial acceleration as shown in figure 6.13. In fact, the tuning curves and other physiological properties of these two populations are precisely what we are most interested in, given our assumptions. Determining what they look like in the model is intended to help physiologists know what to look for in the real system. Note that because we have assumed linear dendrites in this model, it is likely that the tuning curves in the $\mathbf{I}(t)$ population will be most informative about tuning curves in the real system (since they will be the same regardless of whether or not the neurons in this system are best modeled as having linear or nonlinear dendrites). Because the biophysical properties of these neurons are similar to other cells in the vestibular nucleus, we model them using similar parameter regimes. So it is hypotheses regarding the tuning curves and other characteristics of these populations that we are interested in discovering by constructing this model.

6.5.2 Design specification

In the case of vestibular responses, it is difficult to find experiments elucidating the precise range over which the sensory organs operate. Rather, most experiments are designed to determine the range over which the organs are linear, in order to contrast those responses with some observed nonlinear responses. In the case of both the canals and the otolith organs, that linear range is quite large (Wilson and Jones 1979). For this reason, we confine ourselves to constructing a model that is designed to provide insights into this broad, but linear, range. Thus, we take the range, precision, and noise of the higher-level variables to be determined by the behavior of single cells in the ranges that are best documented in the current literature.

Not surprisingly, the range of linear responses of the sensory afferents includes the most common kinds of motion experienced by the system (Wilson and Jones 1979, p. 54). In the case of the canals, it has been shown that the gain is essentially flat for all cells with stimuli ranging from .1 to about 1 Hz. Furthermore, the majority of cells have a flat gain up to about 8 Hz (Wilson and Jones 1979, p. 100). In the case of the otoliths, Fernández and Goldberg (1976a) showed that the frequency gain of the otoliths is flat over a range from DC to about 2 Hz (which was the maximum tested). This corresponds to a linear response between about 0 and 1 g.[14] In a separate set of experiments, these researchers tested single afferents over a range of -1.23 to 1.23 g and found a slightly lower gain to negative accelerations (i.e., accelerations opposite to the preferred direction vector) than to positive accelerations. However, both responses were quite linear, with only the negative response saturating. This suggests that these neurons respond much like LIF neurons with a threshold of about -1 g. The cutoff of the real neurons was much less abrupt than that of an idealized LIF, but this difference can be accounted for by the addition of noise or moderate adaptation, as discussed previously.

14 Note that 1 g = acceleration due to gravity $\approx$10 m/s^2.

To determine the range of gains found in these populations, we can look to the results on the sensitivity of single cells. For the canal afferents, the sensitivity has been found to range between 0.1 and 0.8 spikes/s per degree/s^2, with an average value of about 0.4 (Wilson and Jones 1979, p. 98). In the otolith, the sensitivity has been reported to be about 35 spikes/s per g (Fernández and Goldberg 1976a). Notably, the vestibular neurons that the afferents project to tend to have a sensitivity that is about double the afferents (Wilson and Jones 1979, p. 155), suggesting that they also have gains that are approximately double. In each case, we assume a Gaussian noise with $\sigma^2 = .1$, as usual.

6.5.3 Implementation

In order to construct a neural model of this system, we need to embed each of the higher-level representations and transformations identified in figure 6.13 into the relevant neural populations. The representations and transformations up to the vestibular nucleus should be evident given previous discussions, so we do not consider them further. We focus instead on the representation and transformations of the vestibular neurons that perform the cross product, a highly nonlinear operation.

As in the multiplication example, we take the first population of neurons involved in the cross-product transformation, $c_k(t)$, to construct an integrated representation of the input space, $\mathbf{C}$, by encoding $\mathbf{\Omega}(t)$ and $\mathbf{I}(t) - \mathbf{A}(t)$ simultaneously using a six-dimensional encoding vector, $\tilde{\phi}^{\mathbf{C}}$:

$$\begin{aligned} c_k(\mathbf{C} = [\mathbf{\Omega}, \mathbf{I} - \mathbf{A}]) &= G_k\left[\alpha_k \left\langle \tilde{\phi}_k \mathbf{C}\right\rangle_n + J_k^{bias}\right] \\ &= G_k\left[\alpha_k \left\langle \tilde{\phi}_k^{\mathbf{\Omega}} \mathbf{\Omega} + \tilde{\phi}_k^{\Delta} (\mathbf{I} - \mathbf{A})\right\rangle_n + J_k^{bias}\right] \\ &= G_k\left[\sum_j \omega_{kj} b_j + \sum_i \omega_{ki} a_i + \sum_l \omega_{kl} d_l + J_k^{bias}\right], \end{aligned}$$

where $\omega_{kj} = \alpha_k \left\langle \tilde{\phi}_k^{\mathbf{\Omega}} \phi_j^{\mathbf{\Omega}} \right\rangle_n$, $\omega_{ki} = \alpha_k \left\langle \tilde{\phi}_k^{\Delta} \phi_i^{\mathbf{A}} \right\rangle_n$, and $\omega_{kl} = -\alpha_k \left\langle \tilde{\phi}_k^{\Delta} \phi_l^{\mathbf{I}} \right\rangle_n$.

Note that the canal input, $\mathbf{\Omega}$, is represented by three sub-populations. Thus, each of the components of the $\mathbf{\Omega}$ input is represented by a distinct population of neurons, which together comprise $b_j(t)$. In contrast, the otolith representation, $\mathbf{A}$, is distributed across a whole host of preferred direction vectors that effectively tile the three-dimensions of possible linear accelerations. This difference is not evident from looking at the c_k population, but is an important part of the model.

Now that we have defined the encoding of $\mathbf{C}$, we also need to define the decoding, in order to have defined the representation. As usual, we take the decoding to be

$$\mathbf{C} = \sum_k \phi_k^{\mathbf{C}} c_k(\mathbf{C}). \tag{6.32}$$

However, we also need to use this encoding in the c_k population to compute the cross-product of elements in the $\mathbf{C}$ space. Thus, we need to define the decoding for the function $P(\mathbf{C})$ which we take to be the cross product of the first and last three elements of the vector $\mathbf{C}$. This decoding is

$$P(\mathbf{C}) = \sum_k \phi_k^{P} c_k(\mathbf{C}). \tag{6.33}$$

For both (6.32) and (6.33), we find the decoding vectors by minimizing the relevant errors, as before.

We are now in a position to characterize the final transformation of the circuit, which results in the estimate of $\mathbf{I}$. We take the encoders of the final population, d_l, to consist of three dimensional vectors that adequately tile the $\mathbf{I}$ space. The decoders, $\phi_l^{\mathbf{I}}$, are determined as usual. What is of greatest interest for the d_k population is the transformation that determines the neuron activities at any given time, t. To describe the dynamics of this population, we can use equation (6.30) and directly write $d_l(\mathbf{I}(t+\tau))$:[15]

$$d_l(\mathbf{I}(t+\tau)) = G_l\left[\alpha_l\left\langle\tilde{\phi}_l^{\mathbf{I}}\mathbf{I}(t+\tau)\right\rangle_n + J_l^{bias}\right].$$

Substituting from (6.30) we have

$$\begin{aligned} d_l(\mathbf{I}(t+\tau)) &= G_l\left[\alpha_l\left\langle\tilde{\phi}_l^{\mathbf{I}}\left(\mathbf{I}(t) + \tau\left[\dot{\mathbf{A}}(t) + \mathbf{\Omega}(t)\times(\mathbf{I}(t) - \mathbf{A}(t))\right]\right)\right\rangle_n + J_l^{bias}\right] \\ &= G_l\left[\sum_{l'}\omega_{ll'}d_{l'}(t) + \tau\sum_k \omega_{lk}c_k(t) + \tau\sum_i \omega_{li}a_i(t) + J_l^{bias}\right], \end{aligned}$$

where $\omega_{ll'} = \alpha_l\left\langle\tilde{\phi}_l^{\mathbf{I}}\phi_{l'}^{\mathbf{I}}\right\rangle_n$, and $\omega_{lk} = \alpha_l\left\langle\tilde{\phi}_l^{\mathbf{I}}\phi_k^{P}\right\rangle_n$. Because this transformation includes the derivative of the otolith signal, $\dot{\mathbf{A}}$, we should consider the use of ω_{lk} as short-hand for a more complex projection which we do not discuss here.

We can now test our model by comparing it to experimental results gathered from monkeys. In order to show that the vestibular system is able to estimate the true inertial acceleration, despite the ambiguity in the otolith signal, the following set of three experiments was performed on macaque monkeys (see figure 6.14). First, the animal was subjected only to translational (i.e., inertial) acceleration. Second, the animal was subjected to a transla-

15 Also see section 8.2.1 for a more detailed discussion and justification.

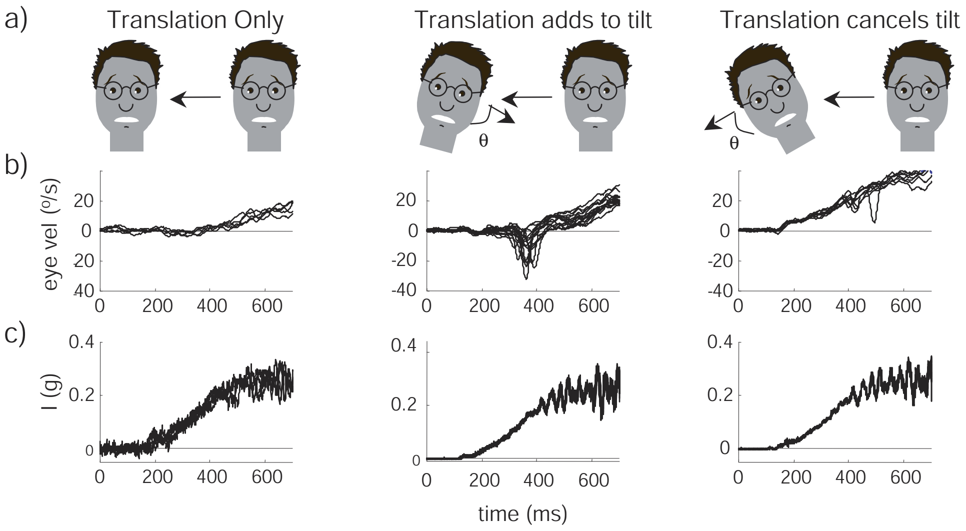


Figure 6.14
Experiments showing that the otolith is not solely responsible for tVOR. In row a) are depicted the movements the animal is subjected to. Row b) shows the resulting eye velocity in each case. Row c) shows the true translation acceleration of the animal in each case. (Data provided by Dr. Dora Angelaki.)

tional acceleration coupled with an angular rotation chosen to double the otolith signal. Third, the animal was subjected to a translational acceleration coupled with an angular rotation chosen to cancel the otolith signal. Thus, any estimate of translational acceleration based solely on otolith information would be severely adversely affected in the last two of these three cases. Nevertheless, eye position changes resulting from the translational vestibular ocular reflex (tVOR), are very similar in each of these three cases (as shown in figure 6.14). This suggests that the otolith signal is not solely responsible for the tVOR.

Figure 6.15 shows the results of the model compared to both a direct solution of equation (6.29) and the averaged data from the rightmost column of figure 6.14. As can be seen here, the model compares favorably with the experimental results.

In figure 6.16a we have shown the model results from all three cases. As can be seen, in each case the estimate of the true translational acceleration is similar despite the widely varying otolith input. Thus the model verifies that the hypothesis is a reasonable one, and that it can be implemented in a biologically plausible, spiking network.

Notably, once we have constructed the higher-level model described by figure 6.13, we can run our simulation in a number of ways. The results we have discussed so far are for the fully spiking network. However, we can, for instance, run the simulations using the higher-level representations directly (i.e., encoding and decoding the higher-level variables rather

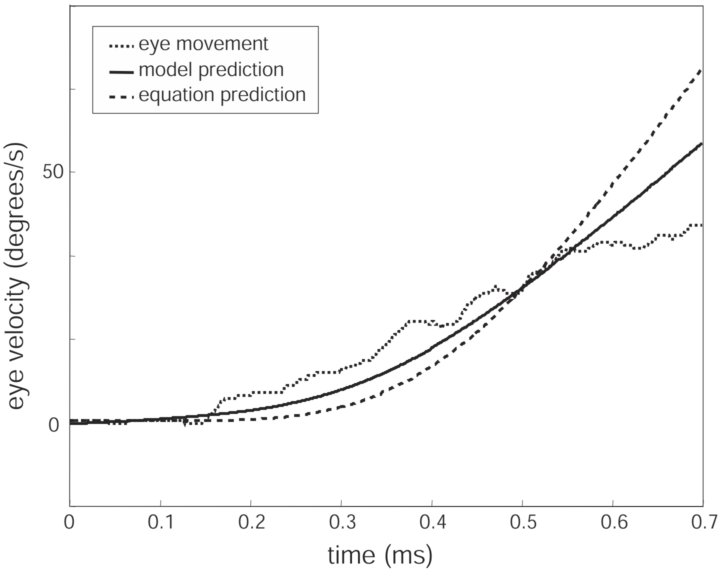


Figure 6.15
Experimental data, equation solution, and model solution for the case in which the otolith signal is canceled. The model consists of a total of 5000 spiking LIF neurons, and incorporates representations of all three spatial dimensions. Comparisons between the model output and the eye position data incorporate a gain reduction which has been experimentally verified for tVOR, and can range between .1 and .25.

than calculating all the weights and generating spikes) for some or all of the populations. Alternatively, we can run the model using rate neurons, or any other neural model. These kinds of alternative simulations can provide for more insight into the effects of changing single cell properties, or very significant computational savings. In figure 6.16 we compare the spiking simulation and the rate simulation. This comparison verifies that, indeed, the rate neuron simulation mimics the behavior of the spiking neuron solution, although there are some differences that might be of interest. Of course, the rate neuron solution is much faster to run; in this case just over 10 times faster. Being able to run simulations quickly can be very useful when debugging, or otherwise ‘playing’ with a model.

As we have emphasized, one of the reasons for constructing this model is to determine what the tuning properties of certain populations in the model are expected to look like. Of particular interest are those neurons performing the cross-product, since it would be a boon to this hypothesis to verify that such a computation is actually done in the vestibular nucleus. On the basis of this model, we expect these neurons to have preferred direction vectors in a six-dimensional space, which is sensitive to both angular velocity and acceleration (specifically the difference between the otolith signal and the estimate of

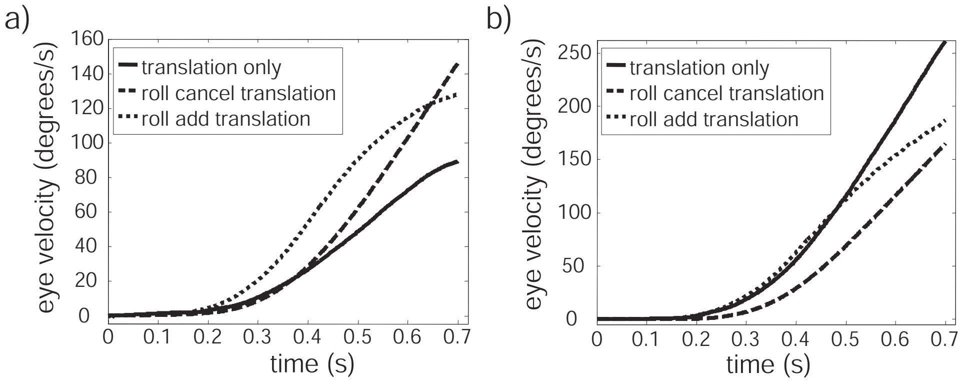


Figure 6.16
A comparison of a) the spiking neuron model for all three cases and b) the rate neuron model for all three cases. The simulation in b) runs just over 10 times faster.

true inertial acceleration). However, this expectation is based mainly on our chosen means of implementing the equation. It is possible, for instance, to perform the cross product as a series of multiplications between components of these vectors. In this case, neuron preferred direction vectors would lie in two dimensional spaces. However we choose to implement the cross product, there must be neurons whose preferred direction vectors *mix* the otolith and vestibular signals. Furthermore, given the fact that there is a nonlinear transformation being performed, we expect neurons in this population to have fairly low background firing rates and more often than not require significant input to begin firing. This is due to the fact that a greater range is required farther from zero for nonlinear computations than for linear computations.

In addition, the dynamics of the network provide predictions regarding the kinds of receptors we expect to find on neurons in the various populations. Although we have not discussed the dynamics in detail here, they provide an indication of some differences between the populations in the network. In particular, based on this model, we expect the neurons participating in the **I** population to have recurrent connections to NMDA-type receptors because the recurrence requires slow synaptic decays to function well. The remaining connections can be fast, AMPA-type receptors. This suggests that some cells in the **I** population will have both kinds of receptors on the same cell. This is consistent with known properties of neurons in the vestibular nucleus (Chen et al. 2000).

6.5.4 Discussion

We have chosen this example because it helps highlight both practical and theoretical issues. On the practical side, this model provides a good demonstration of how models can, and should, influence experimentation. This model provides a simulation of a specific hypothesis made by experimentalists based on their observations of, and theorizing about, a particular system. Adopting some reasonable assumptions about neural function, we can generate specific predictions about what else should be found in a system that abides by that original hypothesis. It should now be possible for the neurophysiologists to re-examine the vestibular system with an eye to finding particular kinds of cells, whose presence would strongly support their original hypothesis. Of course, if no such cells are found, there are any number of possible reasons why, but if they are, it is probably not just a coincidence.

One of the reasons our predictions may not be born out is because writing down the transformation does not *determine* the implementation—it only determines *possible* implementations. So, for example, there are other modularizations of this equation that do not need a population that represents $\dot{\mathbf{A}}(t)$, as this one does (see figure 6.13). So, we should construct models to explore the various possible implementations and determine which, if any, predictions are consistent across them. At the same time, we could eliminate possible implementations by comparing them to available neuroscientific data. As a result, the interplay between experiment and model should be ongoing and mutually beneficial.

On the more theoretical side, this example exposes an important tension in the framework as we have so far discussed it. Recall that in section 6.1.2 we noted that it is useful to define representational modules, such that a particular variable can be said to be represented by a particular neural population. We also know that in order to define a representation, we need to identify both the encoding and decoding processes. Thus, different decodings of the same neural population would lead us to make different representational claims about a given neural population. But, in this model we identify *multiple decodings for the same population*. Specifically, in the population that encodes the $\mathbf{C}$ space, we define the decoding function over the $\mathbf{C}$ space in (6.32), but it is never used in the model. We only ever use the decoders, ϕ_l^P, from (6.33). How, then, can we consistently claim that the population can be treated as a module *representing* $\mathbf{C}$ if only the $P(\mathbf{C})$ decoders are used?

The answer is reasonably straightforward: *we take a population to represent whatever variable* each *decoding population's response can be written as a function of.* So for example, if each population that receives projections from c_k represents some function of $\mathbf{C}$, then it makes sense to claim that c_k represents $\mathbf{C}$. In a sense, then, $\mathbf{C}$ is something like the 'primary' representation of that population, despite the fact that one population may decode it as representing $\mathbf{C}^2$ and another may decode it as representing $\mathbf{C}^3$. This is true even if no population decodes it as representing $\mathbf{C}$. Given this definition, why claim that c_k represents $\mathbf{C}$ given that there is only one way in which this information is decoded? There are two reasons. First, we do not suppose that, in a real brain, these neurons only project to this one population, d_l. Presumably, there are many other populations that these neurons would project to, and we think it most likely that those decodings can be written as a function of $\mathbf{C}$, given how $\mathbf{C}$ is defined. Second, to choose a more complex representation for the population of neurons would mean identifying a more complex encoding function. This is because whatever is represented by the population must be encoded by the population. Thus, if we thought c_k encoded the cross product of its inputs, we would have to identify *encoding* functions that mapped the input, $\mathbf{\Omega}$ and $\mathbf{I}-\mathbf{A}$, to their cross product. It is highly unlikely that such functions, if we could even find them, would result in reasonable tuning curves for neurons in c_k. In contrast, we can find reasonable encoding functions that map $\mathbf{\Omega}$ and $\mathbf{I}-\mathbf{A}$ into $\mathbf{C}$, making it better to claim that c_k represents $\mathbf{C}$.

There is one final constraint to consider regarding claims about what a particular module represents: our current physical theories. This constraint distinguishes the claim that the afferents from the otolith represent linear acceleration from the claim that they represent the square of linear acceleration. Of course, any function that can be written in terms of one can be re-written in terms of the other. The reason we choose linear acceleration to be represented by otolith afferents is that we already have physical theories that take *linear acceleration* (not its square) to be foundational. This may seem obvious, but it nevertheless deserves mention because it helps show what our claims about representation in neurobiological systems really depend on.

To clarify these points, figure 6.17 shows a hypothetical example in which each of these considerations is relevant. Suppose v_v and v_h are vertical and horizontal velocities respectively. We would claim that the $a_i(t)$ population represents the velocity vector $\mathbf{v}$ because each of the populations that it projects to uses some transformation of $\mathbf{v}$. This is true despite the fact that $\mathbf{v}$ is never decoded in the neural system. As well, the encoding vectors we would need to map v_h and v_v into $\mathbf{v}$ would be simple and result in reasonable neuron tuning curves. Finally, we would claim that it represents *velocity* because this is a concept that we are very familiar with from our physical theories. This, then, is how we should substantiate the claim that $a_i(t)$ can be understood as a representational module that represents $\mathbf{v}$, where that representation is defined by the relevant encoding and decoding vectors for $\mathbf{v}$ (and not any function of $\mathbf{v}$). Still, that population is *used* to carry information about many things other than $\mathbf{v}$ (in fact, *only* things other than $\mathbf{v}$) by subsequent populations.

There is one final consequence of this view of representation that needs to be discussed. Note that in figure 6.17, there is some neural population representing I that projects to the $a_i(t)$ population. Suppose that the neurons $a_i(t)$ actually respond to changes in I (perhaps

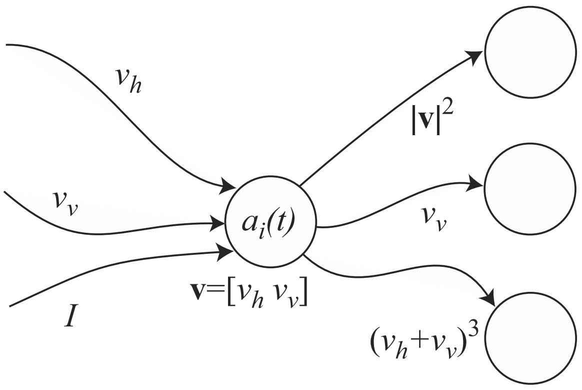


Figure 6.17
Considerations when justifying representational claims. What we take to be represented by the population $a_i(t)$ depends on how the information encoded by the population is used. What is represented is what all decodings can be written as functions *of* ($\mathbf{v}$ in this case). If information carried by $a_i(t)$ is not decoded, it is not represented (like I, or blows to the head).

I is intensity). Even under these conditions, we would not claim that the population represents I. We are assuming, of course, that we have identified *all* of the projections *from* this neural population as well. Since none of the populations it projects *to* uses any information about I, it would be entirely gratuitous to write the relevant transformations as functions of I. Therefore, I does not help define the representation of the neural population and is not represented by that population—regardless of the fact that the $a_i(t)$ neurons respond to changes in I. If this seems counter-intuitive, note that there are many things that influence neural firing that we would not want to count as being represented by that neural firing. Consider, for instance, blows to the head. Such blows would certainly affect neural firing, and may even do so in a systematic manner. Nevertheless, none of the changes in neural firing are used by the system to represent blows to the head. Therefore, blows to the head are not *represented* by those neurons even though their activity is affected by such blows. As odd as this example may sound, it makes clear that understanding how a population's activity is *used* (i.e., decoded) is essential to understanding representation in that population.

6.6 Summary

The discussions in the first half of the book relating to neural representation allowed us to move quickly from basic linear transformations to nonlinear transformations. This is due to the tight relation between representation and transformation captured by the second principle of neural engineering (see section 1.4.2).

We began by showing how linear transformations for both scalars and vectors could be defined by determining the relevant transformational decoder. We then discussed various possible implementations of nonlinear transformations, showing that these could be realized using linear decoding as well. Being able to characterize both linear and nonlinear transformations, we set out to tackle the problem of negative weights that is often left unaddressed by modelers. We showed that given an appropriate augmentation to the representation in a neural population, we can construct a biologically plausible implementation of any transformation (i.e., one with no mixture of positive and negative connection weights).

Finally, in section 6.5, we presented a large-scale model of the vestibular system that both accounts for current data and makes some predictions about the system under study. This model includes both linear and nonlinear transformations implemented in a network of spiking LIF neurons.

This page intentionally left blank

7 Analyzing representation and transformation

To this point, we have presented examples demonstrating the utility of adopting this sort of understanding of neural representation and transformation. However, one of the great benefits of most good theories is that they provide a means of dissecting, at an abstract level, the notions employed by the theory. In this chapter, we discuss analyses of both representation and transformation. In particular, we show how to characterize the *possible* transformations a neural population can support, and we describe a means for determining what can be *usefully* represented by a neural population.

In order to present these analyses concisely, we begin, in sections 7.1 and 7.2, by presenting some of the technical background relevant to understanding the subsequent discussion. Because different disciplines use diverse terminology, notation, and presentation of this material, introducing it here permits us to present our preferred vocabulary, and show how it relates to our previous characterizations of neural representation and transformation.

We conclude, in section 7.5, by identifying an important new category for the analysis of neural representation and transformation that has not received sufficient attention. We argue that the *heterogeneity* of neuron tuning curves found in neural systems plays an important role; a role central to constructing good representations and determining transformations supported by a system.

7.1 Basis vectors and basis functions[1]

Representation, as we have characterized it, is defined by the combination of encoding and decoding. These processes map elements under one description to elements under some other description. Mathematically, we can characterize these processes as functions mapping one 'space' onto another. Ideally, the encoding and decoding functions are inverse mappings between spaces. Thus, any object mapped from one space onto an object in another space by the encoding function would be mapped back onto the object in the original space by the decoding function. To understand how to characterize such abstract spaces, and the mappings between them, we can turn to linear algebra. As originally conceived by Descartes, linear algebra is *for* expressing geometrical relations using arithmetic. So, it should not be surprising that linear algebra is central to a quantitative understanding of abstract representational 'spaces'.

A representational space, like any other, is a specific kind of vector space. *Vectors* are simply collections of mathematical objects, be they numbers, functions, or other vectors.

1 Much of this section may be (tedious) review for those familiar with these topics, and can be skipped without loss of continuity.

A vector *space* is a set of vectors that is closed under addition and multiplication (meaning that a sum or multiple of any two vectors in the set is also in the set). This definition usually results in *many* vectors being in a vector space. So it is important to be able to characterize such a space compactly (i.e., not by writing down all the members). Doing so introduces the notion of the *basis* of a vector space. A basis is an *independent* set of vectors that *span* the vector space.

A set of vectors $\mathbf{x}_n$ is *independent* if

$$a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \ldots + a_n\mathbf{x}_n = 0$$

only when

$$a_1 = a_2 = \ldots = a_n = 0,$$

where a_n are scalars. As a result, n must be equal to the dimension of the vectors in $\mathbf{V}$ in order for $\mathbf{x}_n$ to be independent.

A set of vectors *spans* a vector space if any vector in that space can be written as a linear sum of those vectors. That is, if for all $\mathbf{x} \in \mathbf{V}$ there are some a_n,

$$a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \ldots + a_n\mathbf{x}_n = \mathbf{x},$$

then the set of vectors $\mathbf{x}_n$ span the vector space $\mathbf{V}$.

Note that if the vectors in $\mathbf{x}_n$ are independent, then the set of coefficients, $\mathbf{a}$, is unique for each $\mathbf{x}$. In this case, $\mathbf{x}_n$ is called a *basis*. The standard Cartesian basis in any number of dimensions is comprised of the unit vectors along the principle axes. All the vectors in the standard bases are orthogonal (meaning that the dot product of any two will be zero; i.e., $\mathbf{x} \cdot \mathbf{y} = \langle \mathbf{xy} \rangle_n = \sum_n x[n]y[n] = 0$). This kind of basis is called an *orthogonal* basis.[2]

It should be clear that bases are useful because they let us succinctly describe all members of a vector space. So far, the definitions we have provided are the standard ones that are found in any linear algebra text. These definitions are suitable for characterizing *abstract* spaces and mathematical objects. But, when we are concerned with *real physical systems*, like neurobiological systems, less strict constraints on bases are relevant. If we relax the constraint that the vectors have to be independent, we have what is called an *overcomplete* basis (or sometimes ‘frame’). Of course, overcomplete bases are (strictly speaking) redundant for defining the vector space. Thus, descriptions employing overcomplete bases are are not as succinct as those employing complete, or orthogonal bases. But, in the uncertain, noisy world of physical systems, this redundancy can prove invaluable for error correction and the efficient use of available resources (Goyal et al. 2001). In other words, when real-

2 As well, if the length of all the vectors in an orthogonal basis is equal to one (i.e., if they are all unit vectors) then we have an *orthonormal* basis (e.g., the standard Cartesian basis).

world constraints come into play, the succinctness of our description of the relevant vector space is not all that matters—'usability' matters too.

Before addressing the importance of such constraints directly, let us first consider the relation between bases and representation as we have so far understood it. Intuitively speaking, we can think of each basis vector as 'taking a measurement' of where a particular data point lies in the vector space. We can thus make use of the redundancy of an overcomplete basis by 'cross-checking' our answers between different vectors and thus making our representation more robust. In order to see how bases relate to representation, and how they can be used to make the representation robust, let us consider a simple example (see figure 7.1). Let the vector $\mathbf{x}$ be written in a standard Cartesian basis (e.g., in two dimensions, $\mathbf{x} = [x_1, x_2] = x_1\mathbf{i} + x_2\mathbf{j}$). If we have a system that uses a different basis, say one in which both basis vectors (i.e., $\mathbf{i}$ and $\mathbf{j}$) are rotated by 45 degrees (call these ϕ_1 and ϕ_2), we need to re-write $\mathbf{x}$ in this basis in order to relate it to other objects in that system.

In particular, we can write the *encoding* of $\mathbf{x}$ into this basis as

$$a_i = \langle \mathbf{x}\phi_i \rangle_n, \tag{7.1}$$

which is simply the dot product projection of the vector onto the new basis. To find out where the point $\mathbf{a} = [a_1, a_2] = a_1\phi_1 + a_2\phi_2$ lies in the original space, we can *decode* $\mathbf{x}$ from this new basis (i.e., write it in terms of the standard basis) using

$$\mathbf{x} = \sum_i a_i \phi_i. \tag{7.2}$$

This way, we can move back and forth between bases (see figure 7.1a). Notice that if we substitute (7.1) into (7.2), we recover the same vector we originally encoded. Equation (7.1) thus defines the representation of $\mathbf{x}$ in terms of $\mathbf{a}$ in the new, rotated space. Given the encoding and decodings we have defined, we know exactly how to map the a_i coefficients onto the x_i ones. Thus we can think of the coefficients a_i as 'representing', or 'carrying the same information', or 'encoding' the original coefficients x_i. This simple example shows how basis functions and representations are related: representation relations are defined by identifying and relating bases. In this example, that definition was very straightforward since both bases were orthogonal and thus independent.

Let us now consider an example in which the encoding basis is overcomplete. Suppose that we do not know what the decoding basis is, but we do know what the encoding basis is (this is the problem we have been addressing in neurobiology, i.e., we take the *encoding* to be defined). We can guarantee that the encoding basis is overcomplete by using redundant, non-orthogonal encoders (see figure 7.1b). Let us choose the encoding basis that consists of *three* vectors in the Cartesian plane, equally spaced at 120 degree

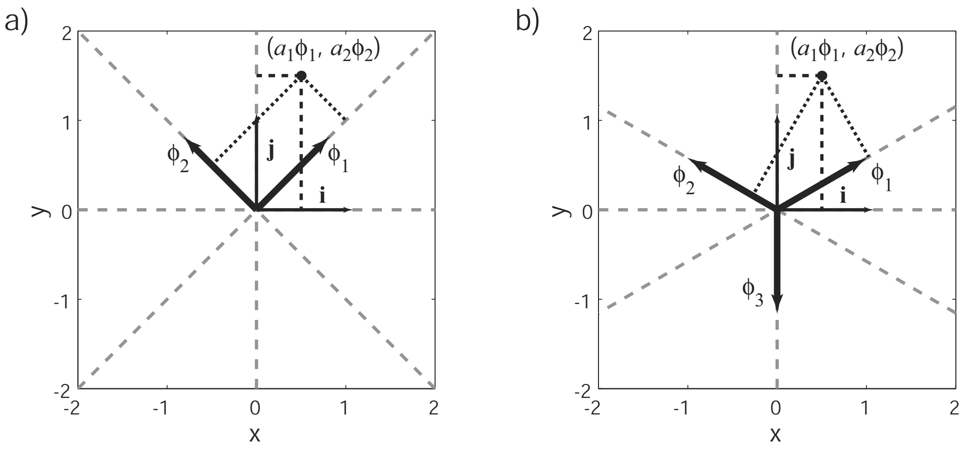


Figure 7.1
Representing a single point using different basis vectors. In both cases, the standard Cartesian basis is shown (**i** and **j**). The projection of the point shown is $[0.5\mathbf{i},\ 1.5\mathbf{j}]$, as indicated by the dashed lines. In a) an orthonormal basis rotated by $45°$ is used. The projection of the point is thus $\left[\sqrt{2}\boldsymbol{\phi}_1,\ \frac{\sqrt{2}}{2}\boldsymbol{\phi}_2\right]$, as indicated by the dotted lines. In b) an overcomplete basis consisting of three equally spaced unit vectors is used. The projection of the point onto this basis is $\left[\frac{\sqrt{3}+3}{4}\boldsymbol{\phi}_1,\ \frac{\sqrt{3}-3}{4}\boldsymbol{\phi}_2,\ -1.5\boldsymbol{\phi}_3\right]$, as indicated by the dotted lines (note the projection onto $\boldsymbol{\phi}_3$ is the same as onto **j** and is thus a dashed line).

intervals (i.e., $\tilde{\boldsymbol{\phi}}_1 = [\frac{\sqrt{3}}{2}, \frac{1}{2}]$, $\tilde{\boldsymbol{\phi}}_2 = [-\frac{\sqrt{3}}{2}, \frac{1}{2}]$, and $\tilde{\boldsymbol{\phi}}_3 = [0, -1]$). Now, since these encoders are not independent, (7.1) and (7.2) do not hold as they did before (which can be easily verified by substitution). So, we need to identify the set of vectors that span the space into which our vector $\mathbf{x}$ is being encoded (i.e., the decoding basis). To find this basis, we first assume, as before, that

$$\mathbf{x} = \sum_i a_i \boldsymbol{\phi}_i, \tag{7.3}$$

and

$$a_i = \left\langle \mathbf{x}\tilde{\boldsymbol{\phi}}_i \right\rangle_n. \tag{7.4}$$

We can now substitute (7.4) into (7.3) to give

$$\mathbf{x} = \sum_i \left\langle \mathbf{x}\tilde{\boldsymbol{\phi}}_i \right\rangle_n \boldsymbol{\phi}_i.$$

Writing the dot product explicitly, we get

$$\begin{aligned} x[m] &= \sum_{i,n} x[n]\tilde{\phi}_i[n]\phi_i[m] \\ &= \sum_n x[n] \sum_i \tilde{\phi}_i[n]\phi_i[m]. \end{aligned}$$

Since

$$x[m] = \sum_n x[n]\delta_{nm},$$

we know that

$$\delta_{nm} = \sum_i \tilde{\phi}_i[n]\phi_i[m]_i$$

or, in matrix notation,

$$\mathbf{I} = \boldsymbol{\phi}\tilde{\boldsymbol{\phi}}, \tag{7.5}$$

where $\mathbf{I}$ is the identity matrix, the columns of $\boldsymbol{\phi}$ are the $\boldsymbol{\phi}_i$, and the rows of $\tilde{\boldsymbol{\phi}}$ are the $\tilde{\boldsymbol{\phi}}_i$. We can thus solve for the decoders

$$\boldsymbol{\phi} = \left(\tilde{\boldsymbol{\phi}}^T\tilde{\boldsymbol{\phi}}\right)^{-1}\tilde{\boldsymbol{\phi}}^T,$$

providing the desired vectors $\boldsymbol{\phi}_i$. Performing the calculations for this example gives $\boldsymbol{\phi}_1 = [\frac{\sqrt{3}}{3}, \frac{1}{3}]$, $\boldsymbol{\phi}_2 = [-\frac{\sqrt{3}}{3}, \frac{1}{3}]$, and $\boldsymbol{\phi}_3 = [0, -\frac{2}{3}]$, i.e.,

$$\boldsymbol{\phi}_i = \frac{2}{3}\tilde{\boldsymbol{\phi}}_i. \tag{7.6}$$

These, then, are the vectors that form the overcomplete basis for the vector space in which the a_i are coordinates. Thus, they are also the decoding vectors which help define the representation of $\mathbf{x}$ by the coefficients a_i. It is, of course, the $\boldsymbol{\phi}_i$ and the $\tilde{\boldsymbol{\phi}}_i$ *together* that define the representation completely. The latter tell us how to encode elements into this space, and the former tell us how to decode elements from this space back to the original Cartesian space.

We are now in a position to understand why this kind of redundant representation can be important for systems in the real world. As we have emphasized, any physical device is subject to noise. Given the way we have characterized representation, representing the physical properties measured by $\mathbf{x}$, means storing the coefficients a_i. Noise, of course, disrupts our ability to store those values perfectly. If we use one physical device for each coefficient we have to store, it is reasonable to assume that the noise affecting one device is independent of the noise affecting another. As is well known, the mean square error will

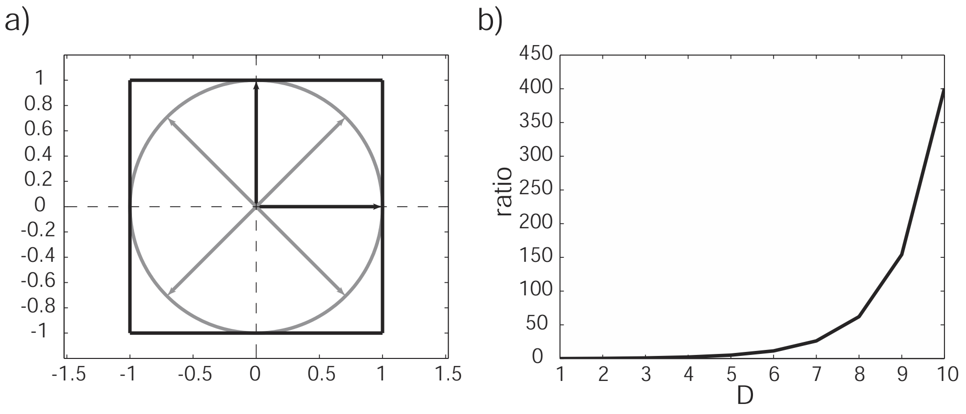


Figure 7.2
Comparing complete and overcomplete representation. a) The shape of complete (black lines) and overcomplete (dashed grey lines) represented spaces. b) The relative increase in the corner areas of a D-dimensional hypercube compared to a hypersphere.

decrease as $1/N$ where N is the number of measurements made in this kind of system. This far, things are the same for the complete and overcomplete representation.

However, this $1/N$ decrease is true *over the whole space represented by the chosen basis*. As shown in figure 7.2a, complete and overcomplete basis vectors represent *different* spaces. Specifically, a D-dimensional complete representation equally well represents all parts of a D-dimensional hypercube whereas an overcomplete representation equally well represents all parts of a D-dimensional hypersphere. Since there are the same total number of measurements, N, for each kind of representation, the overcomplete representation will have less error for a point randomly chosen in the space. This is because the overcomplete representation evenly allocates its resources over the whole space. The complete representation, in contrast, allocates more of its resources to representing the corners of the hypercube. In other words, the complete representation assumes that there is a bias in the space to be represented (i.e., there will be more information in the corners). In fact, as shown in figure 7.2b, this bias becomes worse as the dimensionality of the space, D, increases—almost exponentially worse.

So, if there is no particular structure to the space to be represented, the overcomplete representation will be more robust to noise than the complete representation (Goyal et al. 1998). It is only in the special case that the space has the most vectors to be represented in the corners of a hypercube that the complete representation will do better. If there is some structure to the space, but it is not of this exact form (i.e., with clustering in the corners), then there will be an overcomplete representation that can represent the space better (although it may not be a basis that is equally spaced over the area of the unit circle). So, in general, overcomplete representations are less subject to error than orthogonal ones.

Before we move on, a few comments are in order. First, it is somewhat arbitrary which set of vectors we call the overcomplete basis and which set we call the encoders. It can be shown that both ϕ and $\tilde{\phi}$, in fact, form a basis for the same space. It can also be shown that given either ϕ or $\tilde{\phi}$ as the encoders (or decoders), the other set will be found as the decoders (or encoders). Second, any set of overcomplete encoding/decoding vectors found in this manner will satisfy equation (7.5), and are therefore said to be *biorthogonal*. This is another way of expressing the close relation between the two bases. This is because *together* ϕ and $\tilde{\phi}$ act like an orthogonal basis. That is, they define a means of moving between vector spaces. Third, and perhaps most importantly, the ϕ and $\tilde{\phi}$ we use throughout this book do not, technically, satisfy the definitions we have provide here. *However*, they nearly satisfy these definitions, and the definitions provide a useful way of understanding what the relation between our encoders and decoders is. The reason that the definitions are not satisfied by our ϕ and $\tilde{\phi}$ is that we always pass the encoding (i.e., $\left\langle \mathbf{x} \cdot \tilde{\phi} \right\rangle_n$) through a *nonlinear* neuron response function (i.e., $G\left[\cdot\right]$). But notice that neuron response functions are not *that* nonlinear: generally speaking, the stronger the input current, the higher the firing rate (i.e., they are monotonically increasing). More to the point, neurons are linear *enough* that the preceding discussion is quite relevant to understanding how they represent. Of course, we have to do extra work (e.g., distortion analyses, noise analyses, etc.) to see just how far off neurons are when we treat them as if they were more linear than they actually are.

Finally, we have so far only talked about vectors, but this discussion applies equally to basis *functions*. Just as basis vectors define a vector space, so basis functions define a function space. Perhaps the most widely known basis functions are the sines and cosines. The Fourier decomposition is one that defines some function $f(x)$ in terms of coefficients, A_i, times a basis, $\cos(\omega_i x)$ and $\sin(\omega_i x)$ (see equation 3.3). This is an orthonormal basis. But, just as in the vector case, we can have overcomplete basis functions as well. These are like the $\tilde{\phi}_i(\nu)$ and $\phi_i(\nu)$ we encountered in our characterization of function representation. Recall that these were used to characterize the tuning curve. So, in a sense, the tuning curves are like a set of overcomplete basis functions for the space they represent. As we discuss in section 7.3, looking in more detail at the space defined by the tuning curves helps us understand what functions can be computed using different neural populations. However, before we can turn to this problem, we need to be familiar with one more analytical technique, singular value decomposition (SVD).

7.2 DECOMPOSING Γ

In order to better understand neural representation and transformation, we need methods for analyzing the Γ matrix introduced in chapter 2. Recall that, in the vector case with noise, Γ is defined by the equation

$$\Gamma_{ij} = \langle a_i(\mathbf{x})a_j(\mathbf{x})\rangle_{\mathbf{x}} + \sigma^2\delta_{ij}. \tag{7.7}$$

To simplify matters, let us temporarily consider the representation in a noise-free system. Thus, we are concerned with

$$\gamma_{ij} = \langle a_i(\mathbf{x})a_j(\mathbf{x})\rangle_{\mathbf{x}}. \tag{7.8}$$

The rows of the γ matrix consist of the projections (dot products) of each neuron tuning curve on itself and every other tuning curve. In general, the result of a projection can be thought of as a measure of similarity. So, each entry in the γ matrix measures the similarity between two tuning curves. Taken together, all of these similarity measures in the γ matrix make it something like a covariance matrix. That is, any entry in the matrix tells us how well-correlated the firing of any two neurons in the population is.[3] In a sense, then, this matrix carries the information about the structure of the space being represented by those tuning curves. That is why it is essential to analyze this matrix in more detail.

To begin, let us introduce a new, but convenient matrix notation for our characterization of neural representation. First, recall that for the estimate of $\hat{\mathbf{x}}$ we typically write

$$\hat{\mathbf{x}} = \sum_i^N a_i(\mathbf{x})\boldsymbol{\phi}_i, \tag{7.9}$$

where we have assumed that the relevant preferred direction vectors, $\tilde{\boldsymbol{\phi}}_i$, are chosen randomly from a uniform distribution on the surface of the unit hypersphere (so the magnitude $|\tilde{\boldsymbol{\phi}}_i|^2 = 1$). As well, the elements of the encoded vector, $\mathbf{x}$, are presumed to be drawn from a uniform distribution inside the unit hypersphere (so the magnitude $|\mathbf{x}|^2 \leq 1$).[4]

3 The γ matrix is a kind of matrix sometimes called the 'Gram' matrix. Strictly speaking Gram matrices are not the same as covariance matrices. In particular, the covariance matrix measures variance around the mean, $C_{ij} = \langle (a_i(x) - \bar{a}_i(x))(a_j(x) - \bar{a}_j(x))\rangle_x = \gamma_{ij} + \bar{a}_i(x)\bar{a}_j(x)$. Nevertheless, the second moments (singular values) of both matrices can be used to characterize neural transformations. We prefer to analyze the Gram matrix because it permits us to explicitly extract and exploit the population bias (e.g., this plays a central role in characterizing negative weights). Performing a similar analysis on the correlation matrix is equivalent to principle components analysis (PCA).

4 This second assumption can easily be relaxed, but simplifies the current analysis.

Let us now assume, for the time being, that $\mathbf{x}$ is discretized over the range $\mathbf{x} \in \{\mathbf{x}_{min}, \mathbf{x}_{max}\}$, with a spacing of $\Delta_{\mathbf{x}}$. Let N, N_v, and N_Δ denote the number of neurons, the number of vector elements in $\mathbf{x}$, and the number of elements in the discretization of $\mathbf{x}$, respectively. We can then write our expression for our estimate of $\mathbf{x}$ as a matrix multiplication, i.e.,

$$\hat{\mathbf{X}}_{N_\Delta \times N_v} = \mathbf{A}_{N_\Delta \times N} \boldsymbol{\phi}_{N \times N_v},$$

where

$$\mathbf{A}^T = \begin{bmatrix} a_1(\mathbf{x}_{min}) & a_1(\mathbf{x}_{min} + \Delta_{\mathbf{x}}) & \cdots & a_1(\mathbf{x}_{max} - \Delta_{\mathbf{x}}) & a_1(\mathbf{x}_{max}) \\ a_2(\mathbf{x}_{min}) & & & & a_2(\mathbf{x}_{max}) \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ a_{N-1}(\mathbf{x}_{min}) & & & & a_{N-1}(\mathbf{x}_{max}) \\ a_N(\mathbf{x}_{min}) & a_1(\mathbf{x}_{min} + \Delta_{\mathbf{x}}) & \cdots & a_1(x_{max} - \Delta_{\mathbf{x}}) & a_N(\mathbf{x}_{max}) \end{bmatrix}.$$

That is, the $\mathbf{A}$ matrix is the transpose of the neuron tuning curves, $\hat{\mathbf{X}}$ is a matrix in which each row is the estimate of $\mathbf{x}$ at the corresponding discretized value of $\mathbf{x}$, and $\boldsymbol{\phi}$ is a matrix in which each row is a decoding vector.

As we know, the central problem for characterizing representation is that of finding the optimal $\boldsymbol{\phi}$ given a set of vectors, $\mathbf{X}$, we wish to encode. To solve this problem, we take $\mathbf{X}$ as given and solve for $\boldsymbol{\phi}$ as follows:

$$\begin{aligned} \mathbf{X} &= \mathbf{A}\boldsymbol{\phi} \\ \mathbf{A}^T\mathbf{X} &= \mathbf{A}^T\mathbf{A}\boldsymbol{\phi} \\ (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{X} &= \boldsymbol{\phi}. \end{aligned} \tag{7.10}$$

Presuming we take the inverse of $\mathbf{A}^T\mathbf{A}$ in the right way, this is the same as minimizing the mean square error between the originally encoded value and our estimate. That is, we find[5]

$$\begin{aligned} \gamma &= \mathbf{A}^T\mathbf{A} \\ \Upsilon &= \mathbf{A}^T\mathbf{X}, \end{aligned} \tag{7.11}$$

5 These are the equivalent expressions to those we found earlier, without the discretization of $\mathbf{x}$; namely, $\gamma_{ij} = \langle a_i(\mathbf{x}) a_j(\mathbf{x}) \rangle_{\mathbf{x}}$ and $\Upsilon_j = \langle a_j(\mathbf{x})\mathbf{x} \rangle_{\mathbf{x}}$.

so,

$$\begin{aligned}\phi &= \gamma^{-1}\Upsilon \\ &= (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{X}.\end{aligned} \tag{7.12}$$

We can now write an expression for the estimate of $\mathbf{X}$ in matrix notation as

$$\begin{aligned}\hat{\mathbf{X}} &= \mathbf{A}\phi \\ &= \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{X}.\end{aligned}$$

Of course, the major assumption we have made is that we can take the inverse of γ 'in the right way'. Notice that because this matrix has no noise term, and because *some* tuning curves are likely to be similar for a large population, the matrix γ is likely to be singular, or nearly singular so it is *not* invertible (i.e., it is ill-conditioned). As a result, we need to use analytical tools suitable for handling singular matrices in order to characterize γ.

There exists a general, and very powerful, technique for analyzing singular matrices called singular value decomposition (SVD). Very briefly, an SVD decomposition of an $M \times N$ matrix, $\mathbf{B}$, results in three matrices whose product gives $\mathbf{B}$, i.e.,

$$\mathbf{B}_{M\times N} = \mathbf{U}_{M\times N}\mathbf{S}_{N\times N}\mathbf{D}^T_{N\times N}.$$

The matrix $\mathbf{S}$ is a diagonal matrix whose entries are called the *singular values* of $\mathbf{B}$. In the case when $\mathbf{B}$ is square (as with γ), the matrices $\mathbf{U}$ and $\mathbf{D}$ are both square and orthogonal, meaning:

$$\mathbf{U}^T\mathbf{U} = \mathbf{U}\mathbf{U}^T = \mathbf{D}^T\mathbf{D} = \mathbf{D}\mathbf{D}^T = \mathbf{I}.$$

This also shows that the inverse of the matrices $\mathbf{U}$ and $\mathbf{D}$ are equal to their transposes. Given that the inverse of a diagonal matrix is simply 1 over the diagonal elements, the inverse of the original matrix $\mathbf{B}$ is:

$$\mathbf{B}^{-1} = \mathbf{D}\mathbf{S}^{-1}\mathbf{U}^T. \tag{7.13}$$

In the case where $\mathbf{B}$ is singular (or nearly so), some elements of $\mathbf{S}$ are zero (or very small), so the inverse of $\mathbf{S}$ includes infinite (or very large) terms, meaning the inverse of $\mathbf{B}$ is ill-defined (as expected for singular or near singular matrices). In this case, the SVD 'pseudo-inverse' is defined as in (7.13) where for $S_i = 0$, the inverse is set to 0.

If a matrix, such as γ, is singular, then part of the space it maps (in this case ϕ) is mapped to $\mathbf{0}$. This subspace of ϕ is called the *null space* of γ, and its dimension is called the *nullity*, $\mathcal{L}$, of γ. The elements of ϕ that are not mapped to $\mathbf{0}$ are mapped to what is called the *range* of γ. The dimensionality of the range is called the *rank, $\mathcal{R}$,* of the matrix, γ. For an $N \times N$ square matrix, $\mathcal{R} + \mathcal{L} = N$.

Even in the case when a matrix is singular (or nearly so), SVD can be very informative. In particular, the columns of $\mathbf{U}$ whose corresponding singular values are non-zero form an orthonormal basis that spans the range of $\mathbf{B}$. Similarly, the columns of $\mathbf{D}$ whose corresponding elements are zero form an orthonormal basis that spans the null space of $\mathbf{B}$. Because, in our case, $\mathbf{B}$ (i.e., γ) is a symmetric matrix (i.e., $B_{ij} = B_{ji}$) $\mathbf{D}$ and $\mathbf{U}$ are the *same* matrix, meaning that bases for both the null space and range are in one matrix. We can show this as follows:

$$\begin{aligned} \mathbf{B} &= \mathbf{B}^T \\ \mathbf{USD}^T &= \mathbf{DS}^T\mathbf{U}^T. \end{aligned}$$

Therefore, $\mathbf{D} = \mathbf{U}$ since $\mathbf{S}$ is diagonal (i.e., $\mathbf{S} = \mathbf{S}^T$) and since SVD gives a unique enough decomposition.[6] So, we can write the decomposition of γ as

$$\gamma = \mathbf{USU}^T,$$

or equivalently, in summation notation,

$$\gamma_{ij} = \sum_m U_{im} S_m U_{jm}.$$

This decomposition is useful for characterizing representation and transformation for a number of reasons. First, as already mentioned, the relevant $\mathbf{U}$ matrix provides an orthogonal basis for both the range and nullity of γ. Because γ tends to be singular, both bases are important. Second, when a vector in Υ lies in the range of γ, the SVD pseudo-inverse guarantees that the corresponding vector from ϕ found from (7.12) *minimizes* the length of that ϕ vector. This is important because given that γ is singular, there are an infinite number of solutions for ϕ. The solution that provides the shortest vector is a natural and compact choice from the set. Third, when a vector in Υ lies in the nullity of $\mathbf{A}$, the SVD pseudo-inverse guarantees that the best (in the least squares sense) ϕ given Υ will be found. In other words, this 'pseudo-inverse' minimizes the error in (7.19). Thus, we can use SVD to find the optimal decoding functions as in (7.10) which we can now write as

$$\phi = \mathbf{US}^{-1}\mathbf{U}^T\mathbf{A}^T\mathbf{X}. \tag{7.14}$$

6 The decomposition is unique enough in that linear combinations of any columns whose corresponding S values are identical can replace those columns. However, since the relevant matrices are the same for SVD of symmetrical matrices, any such manipulation on $\mathbf{D}$ can be done to $\mathbf{U}$. Note that this result can also be proven using eigenvector decomposition theorems that show that any symmetric matrix can be written in the form $\mathbf{S} = \mathbf{U}^T\mathbf{AU}$ (Shields 1968, p. 214).

Given the properties of SVD, we know that this is the same solution we would find by constructing the error explicitly (i.e., $E = \left\langle \left[\mathbf{X} - \hat{\mathbf{X}}\right]^2 \right\rangle_{\mathbf{x}}$), taking the derivative, and setting it to zero, as we have previously done.

While SVD turns out to be useful for understanding both representation and transformation, some of the representational analyses make more sense after we have characterized transformation. As a result, we begin by considering transformation first.

7.3 DETERMINING POSSIBLE TRANSFORMATIONS

We have already presented examples of both linear and nonlinear transformations using neural populations, but we have not presented a general, theoretical characterization of transformation. While such examples are useful, they do not show how to answer a pivotal question about transformations and neurobiology, namely: Given a neural population, what are the *possible* transformations that it can support? Here, we present a general analytical technique for answering this question.[7]

In section 7.2 we used SVD to find the representational decoders of $\mathbf{x}$. Here, we perform a similar analysis to find an expression for the decoding vectors needed to decode any *transformation* of $\mathbf{x}$:

$$\begin{aligned} f(\mathbf{X}) &= \mathbf{A}\phi^f \\ \mathbf{A}^T f(\mathbf{X}) &= \mathbf{A}^T\mathbf{A}\phi^f \\ \phi^f &= (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T f(\mathbf{X}) \\ &= \gamma^{-1}\Upsilon \end{aligned}$$

Performing SVD on $\mathbf{A}^T\mathbf{A}$ to find the inverse, as before, gives

$$\phi^f = \mathbf{U}\mathbf{S}^{-1}\mathbf{U}^T\mathbf{A}^T f(\mathbf{X}), \tag{7.15}$$

where ϕ^f are the linear decoders for estimating the transformation $f(\mathbf{X})$. So, the representational decoder, ϕ, is found in the special case where $f(\mathbf{X}) = \mathbf{X}$. Notice that regardless of which transformation we need decoders for, we always perform SVD on the same matrix, $\gamma = \mathbf{A}^T\mathbf{A}$. This suggests that understanding the properties of γ can provide general insight into all possible decodings of the population, a_i.

Recall that singular value decomposition finds the matrix $\mathbf{U}$, whose columns are orthogonal vectors spanning the range and null-space of γ, and a diagonal matrix of singular values, $\mathbf{S}$. The singular values are useful because they tell us the *importance* of

7 We have previously presented a similar analysis in Westover et al. (2002).

the corresponding $\mathbf{U}$ vector. There are a number of ways of thinking about 'importance' in this case. For instance, we can think of 'importance' as being related to the error that would result if we left a particular vector out of the mapping. If we remove a vector with a high singular value from the decomposition and then reconstruct γ, we introduce a lot of error into the reconstructed matrix (and therefore into estimates of the inverse which determine the decoders). Conversely, if we remove a vector with a low singular value, we introduce little error. Or, we can think of 'importance' as being related to the variance of population firing along the vectors in the γ matrix. Under this interpretation, the $\mathbf{U}$ vector with the largest corresponding singular value can account for the most variance in the population firing over the range of the population's representation. The $\mathbf{U}$ vector with the second largest singular value explains the next most variance, and so on. Relatedly, we can think of 'importance' as being the amount of (independent) information about changes in population firing that can be extracted by looking only at data projected onto the corresponding $\mathbf{U}$ vector.

In general, we can think of the magnitude of the singular value as telling us how relevant the dimension defined by the corresponding $\mathbf{U}$ vector is to the identity of the matrix we have decomposed. Since the matrix we have decomposed is like the correlation matrix of the neuron tuning curves, the large singular values are most important for accounting for the structure of those correlations. It is that structure that encodes the input variable, so accounting for the correlational structure is equivalent to accounting for the encoding of the input variable. Knowing exactly what information was encoded permits us to determine what we can decode.[8]

It is also relevant that the vectors in $\mathbf{U}$ are orthogonal. Since these vectors can be used to reconstruct to original γ matrix, they provide an (ordered) orthogonal *basis* for that matrix. This is very useful because the original γ matrix was generated by a non-ordered non-orthogonal basis; the neuron tuning curves. To make this point more quantitative, let us begin by defining a point in the 'neuron space' (i.e., the space spanned by the overcomplete neuron tuning curves) as

$$\mathbf{a} = a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + \ldots + a_N\mathbf{e}_N.$$

In this notation, the vectors $\mathbf{e}_i$ serve as axes for the state space of the neural population. A point in this space is defined by the neuron firing rates from each neuron in the population (which, taken together, form the vector $\mathbf{a}$).

Because the neural responses are non-independently driven by some variable, $\mathbf{x}$, only a *subspace* of the space spanned by the $\mathbf{e}_i$ vectors is ever *actually* occupied by the population. The γ matrix, because it tells us the correlations between all neurons in the population,

8 See appendix E.1 for some related and important practical considerations regarding decoding of $f(\mathbf{x})$.

provides us with the information we need to determine what that subspace is. When we find the $\mathbf{U}$ vectors in the SVD decomposition, we have characterized that subspace because those are the orthogonal vectors that span it. Better yet, as discussed previously, we know which vectors are most important and which least important for defining the subspace.

This information is exactly what is needed in order to determine what functions can be computed by the particular encoding of $\mathbf{x}$ found in the a_i population. To see why, let us use (7.14) to write our estimate of $\mathbf{x}$ in matrix form as

$$\hat{\mathbf{X}} = \mathbf{A}\mathbf{U}\mathbf{S}^{-1}\mathbf{U}^T\mathbf{A}^T\mathbf{X},$$

or, more simply

$$\hat{\mathbf{X}} = \chi\Phi, \tag{7.16}$$

where

$$\chi = \mathbf{A}\mathbf{U},$$

so

$$\chi_m(\mathbf{x}) = \sum_i a_i(\mathbf{x})U_{im}, \tag{7.17}$$

and

$$\begin{aligned}\Phi &= \mathbf{S}^{-1}\mathbf{U}^T\mathbf{A}^T\mathbf{X}\\ &= \mathbf{U}^T\mathbf{U}\mathbf{S}^{-1}\mathbf{U}^T\mathbf{A}^T\mathbf{X}\\ &= \mathbf{U}^T\boldsymbol{\phi},\end{aligned}$$

so

$$\Phi_m = \sum_i U_{mi}\boldsymbol{\phi}_i.$$

Notice that χ and Φ in (7.16) are rotated versions of $\mathbf{A}$ and $\boldsymbol{\phi}$ respectively. Specifically, they are rotated into the coordinate system defined by $\mathbf{U}$. So we can think of $\mathbf{U}$ as the rotation matrix that aligns the first axis of the coordinate system along the dimension with the greatest variance in the encoding of $\mathbf{x}$, the second axis along the dimension with the second greatest variance, and so on. As a result, and as shown in appendix E.2, the χ vectors also end up being orthogonal and ordered by importance. That is, they tell us what can be extracted, and how well it can be extracted, from the neural population.

We can think of the components of χ as *basis functions* of the space that includes the ensemble of transformations definable on $\mathbf{x}$ using the encoding in the population a_i. This is more evident in (7.17) where we write χ in summation notation. There it is quite

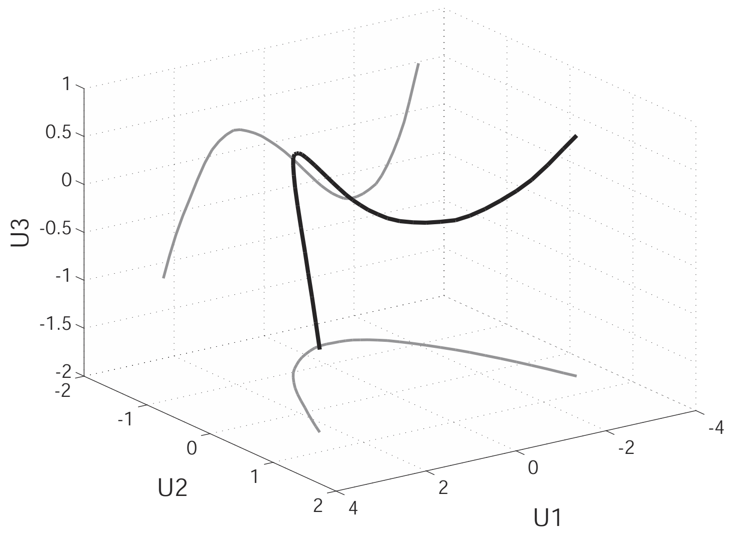


Figure 7.3
An example of a neural subspace (black line) and its projection onto the $\mathbf{U}$ axes (grey lines). This example is taken from a population with monotonically increasing tuning curves (see section 7.3.1). The different scales of the axes indicate their relative importance, i.e., the magnitude of the associated singular value.

obvious that $\chi_m(\mathbf{x})$ at a particular value of $\mathbf{x}$ is the neuron firing rates at that value of $\mathbf{x}$ projected onto the mth orthonormal $\mathbf{U}$ vector (see figure 7.3). Whichever $\chi(\mathbf{x})$ functions have reasonably large associated singular values, are exactly the functions that we can do a good job of extracting from our encoding of the input space, $\mathbf{x}$. Of course, we can also extract any linear combinations of those $\chi(\mathbf{x})$ functions quite well. But, because these functions are ordered, the more useful the 'first' $\chi(\mathbf{x})$ function is for reconstructing some transformation, $f(\mathbf{x})$, the better we can extract that transformation, $f(\mathbf{x})$.

In practice, we can look at the resulting $\chi(\mathbf{x})$ functions and determine what sort of basis we seem to have. For example, if the $\chi(\mathbf{x})$ look like sines and cosines, we have a Fourier basis. So from that encoding of $\mathbf{x}$, we can compute any functions that can be constructed with the sines and cosines available. As we show in the next section, it is somewhat surprising, but very convenient, that we find recognizable bases when we analyze realistic neural populations.

Before seeing two examples of this, it is useful to note that introducing noise into the γ matrix does not greatly change this analysis. Rather than think of the neural state vectors as defining points in the neural space, we can think of them as defining clouds of points, where the structure of the cloud is determined by the neural noise. Thus the subspace defined by the encoding of some variable, $\mathbf{x}$, also becomes blurred. So, for instance, in figure 7.3, the thick dark line would look something like a tube whose boundary is determined by the magnitude of the noise. Notably, the noise is likely to have a constant variance along each axis in the $\mathbf{U}$ space, so it will have a greater effect on axes with small corresponding singular values. Thus decoding functions that are linear combinations of $\chi(\mathbf{x})$ functions with large singular values will also be the most robust to corruption by noise.

Finally, before concluding this section, let us briefly discuss transformations of function representations. Not surprisingly, the analysis of such transformations is a straightforward extension of the analysis of vector transformations. This is true for two reasons. First, any function representation in the face of noise, will be equivalent to some high-dimensional vector representation under no noise. Second, we define function representations as expansions in some orthonormal basis, with only a certain number of terms in the expansion. Thus, any function in such a representation can be perfectly described by its finite coefficients, which *themselves* can be treated as a vector space (as discussed in section 3.3). We provide more detailed examples of function transformations in the next chapter, but suffice it to say that, in some ways, they are nothing new.

7.3.1 Linear tuning curves

While it is useful to know the general theory that applies to any population, examining a specific example shows how the theory can be useful for analyzing real neural systems. In this section, we present the results from applying the above analysis to a population of neurons with broad, nearly linear tuning curves, like those seen in the neural integrator (see figure 7.4). These kinds of neurons are most common in peripheral motor systems, but are also found in vestibular systems and cortex. We first present an analysis of encoding a scalar variable, x, and later discuss encoding of a vector, $\mathbf{x}$, which introduces some additional complexities to the analysis.

As mentioned in section 7.3, in order to get a good idea of the kinds of functions that can be encoded by a given population, we must look at the $\chi(x)$ functions to see what kind of basis they form. For the population in figure 7.4, these functions are given by

$$\chi_m(x) = \sum_i a_i(x) U_{im}.$$

Recall that the $\chi_m(x)$ are the projection of the population state space onto the orthonormal $\mathbf{U}$ vectors (see figure 7.3). Plotting the first five (i.e., the five with the highest corresponding singular values) of these functions gives us a good sense of what basis spans the neural space (see figure 7.5a). It is clear from this figure that this set of functions closely resembles a *polynomial* basis because the first function is approximately constant, the second approximately linear, the third approximately quadratic, etc. One of the most

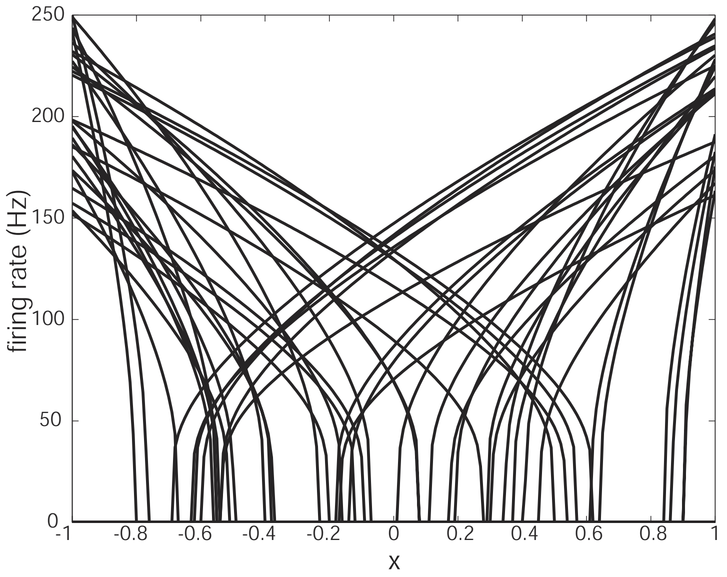


Figure 7.4
Broadly tuned population of neuron tuning functions encoding the scalar x. Fifty of the 200 neurons used are shown here. These resemble the population of neurons found in the neural integrator (see section 2.3).

common polynomial bases used in mathematics is the Legendre basis, $l_i(x)$, which is defined over the interval [-1,1] and results from the orthogonalization of x^n.[9] Scaled versions of the first five elements of this basis are plotted in figure 7.5b. Surprisingly, the $\chi_m(x)$ very closely approximate $l_i(x)$.

This analysis highlights an important difference between our analysis of Γ and a principle components analysis (PCA) of the correlation matrix (see footnote 3). In particular, we find that $\chi_1(x)$, an approximately constant function, can be well-approximated by this population (in fact it has the highest singular value). This fact is essential to our earlier solution to the problem of negative weights that we discussed in section 6.4, because we rely heavily on decoding a constant value.

Generally, the similarity between $\chi_m(x)$ and $l_i(x)$ means that this neural population supports the extraction of functions that can be well-estimated using the standard Legendre basis. Recall, however, that the $\chi_m(x)$ functions are ordered by their singular values. Thus, the higher-order polynomial terms are not as well encoded by our population as the lower-order ones. So, computing functions that depend strongly on precise high-order terms will be prone to error.

9 One way of expressing the basis is: $l_i(x) = \frac{(-1)^i}{2^i i!} \frac{d^i}{dx^i} \left[\left(1 - x^2\right)^i \right]$.

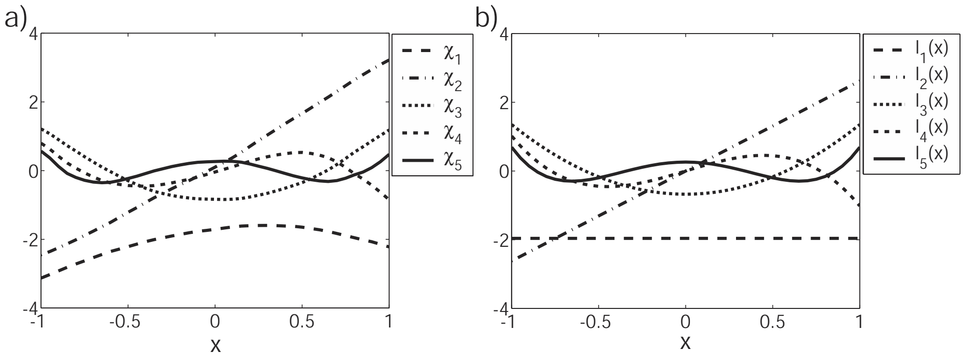


Figure 7.5
a) The first five $\chi(x)$ functions for the neural population in figure 7.4. b) The first five functions in the Legendre polynomial basis. These functions have been normalized to match the magnitude of the $\chi(x)$.

In some ways, this is a natural basis to be found from the tuning curves in the population in figure 7.4. The tuning curves are very broad, and the polynomial basis is also very broad. These tuning curves are approximately linear, and the more linear basis functions are also the first ones. The Legendre polynomial basis is ordered by decreasing linearity so it should not be too surprising that this population supports the functions in precisely that order. However, none of this would be true if the population did not do a good job of evenly tiling the input space. If, for example, there were only high gain neurons, whose slopes were the same sign as their x-intercepts (i.e., if the ‘on’ and ‘off’ sub-populations were ‘clumped’ near x_{max} and x_{min} respectively), we would not expect the linear term to be better supported than the quadratic term. In this sense, the heterogeneity of the population helps it support the more natural ordering of the polynomial basis; clumping would defeat this ordering. Thus, this particular set of $\chi_m(x)$ functions does not just depend on the general ‘shape’ of the neuron tuning curves, but also on which neurons are included in a population, i.e., the degree of heterogeneity (see section 7.5 for more on the importance of heterogeneity).

The preceding discussion and examples all assume transformations of a scalar variable, x. These results generalize as expected to transformations of a vector, $\mathbf{x}$. However, there are additional analyses we can perform for vectors of two or more dimensions. When computing transformations of populations encoding n-dimensional vectors, we must realize that there are additional cross terms (e.g., x_1x_2) that introduce variations in the encoding that are not present in scalar transformations.[10] For example, consider encoding

10 In fact, we found the cross-term decoder in our previous example of multiplying two variables discussed in section 6.3.1.

a 2-dimensional vector with a population like that in figure 7.4. As in the scalar case, transformations that can be written as polynomial expansions are best supported by this population. However, the expansions are now of the form

$$\begin{aligned} f(\mathbf{x}) &= c_0 + c_1 x_1 + c_2 x_2 + c_3 x_1^2 + c_4 x_2^2 + c_5 x_1 x_2 + \ldots \\ &= \sum_{l=0}^{N_{order}} \sum_{n=0}^{l} c_{n,l-n} x_1^n x_2^{l-n}, \end{aligned}$$

where N_{order} is the highest order term in the transformation and l indexes the lth order terms in the expansion (all terms whose exponents *sum* to l are considered lth order terms). As we can see, the cross terms (i.e., $x_1^n x_2^{l-n}$) are quite common. In order to characterize how well an arbitrary function can be decoded from a representation of $\mathbf{x}$, we need to know how big the singular values of these cross terms are as well.

A quick inspection of the singular values of the population reveals that all terms of a given order have singular values of approximately the same magnitude (see figure 7.6). Between the singular values of different orders, however, there is a significant change in magnitude of the values. This means that a given population equally supports transformations of any given order, l. As well, there is an exponential decay in the magnitude of the singular values as a function of the order of the polynomial. This means that lower-order functions are *significantly* better supported by these kinds of broadly tuned neural populations. That is, the ability to extract higher-order functions drops more quickly with an increase in the order of the function than compared to the scalar case.

In fact, we can determine exactly how many singular values, N_S, there should be for each order, l, in the polynomial for a input space of size D by using the equation for determining l-combinations of a multi-set:

$$N_S(l, D) = \frac{(l + D - 1)!}{l!(D - 1)!}. \tag{7.18}$$

Figure 7.6 shows that this equation roughly predicts the large magnitude changes in the singular values, especially for the lower singular values. This tells us how many $\mathbf{U}$ basis functions there are for each polynomial order, and that they are (nearly) equally important to computing transformations of that order. Since the function in (7.18) goes as approximately D^l, we know that computing higher-order terms for high-dimensional input spaces requires many, many neurons. This is because we need a population of neurons that can precisely encode many degrees of variation (i.e., have a large number of singular values significantly higher than the background noise). Of course, this is to be expected—more complex and higher-dimensional functions take more resources to compute.

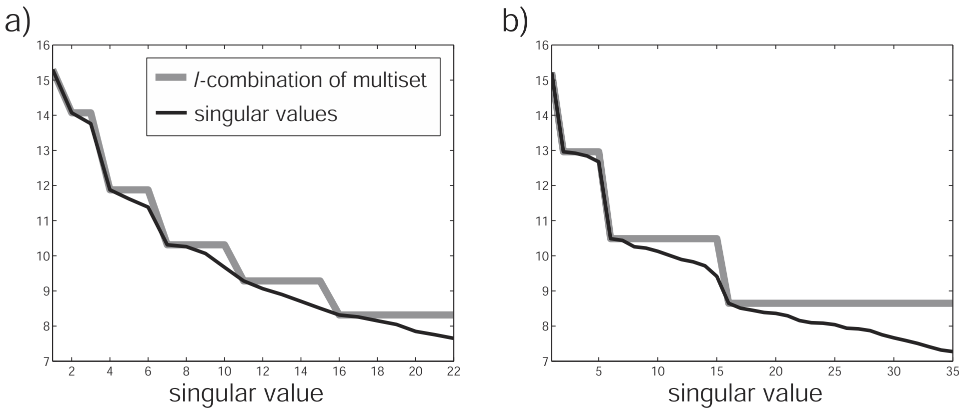


Figure 7.6
Singular values compared to l-multi-set predictions for D-dimensional input spaces. In a) $D = 2$ in b) $D = 4$. Clearly the multi-set predictions get worse for higher singular values, especially for small D. Nevertheless, it does capture the main structure of the more important singular values.

7.3.2 Gaussian tuning curves

Perhaps the most common kind of tuning curve found in the central nervous systems of mammals is that with 'bell-shaped' (or Gaussian or cosine) tuning (see figure 7.7). It is useful to see how this kind of population differs in the transformations it supports from the broadly tuned linear population we have already discussed.

Figure 7.8 shows the first five $\chi_m(x)$ functions for this population. These functions look quite different from those in figure 7.5. First, there is no linear term. This lack of a linear term suggests that this population will not do as well supporting a linear transformation as the previous population. In general, the $\chi_m(x)$ for this Gaussian population more closely approximate a Fourier basis than a polynomial basis. The main difference between this basis and a Fourier basis is that these are defined on a finite domain [-1,1]. If the x-axis was periodic, the $\chi_m(x)$ functions are even more similar to the standard Fourier basis. As a result of getting this particular set of $\chi_m(x)$, we know that this population supports transformations that can be defined using something like a Fourier series expansion, particularly if the low frequencies are most important.

To more directly compare the relative strengths of the linear and Gaussian encodings, we can look directly at how the singular values decline in each case. This is shown for both populations in figure 7.9. This figure demonstrates that both populations do an equally good job of encoding the constant term. However, the linear population also encodes the linear term extremely well, whereas the Gaussian population drops off steeply for its first sine

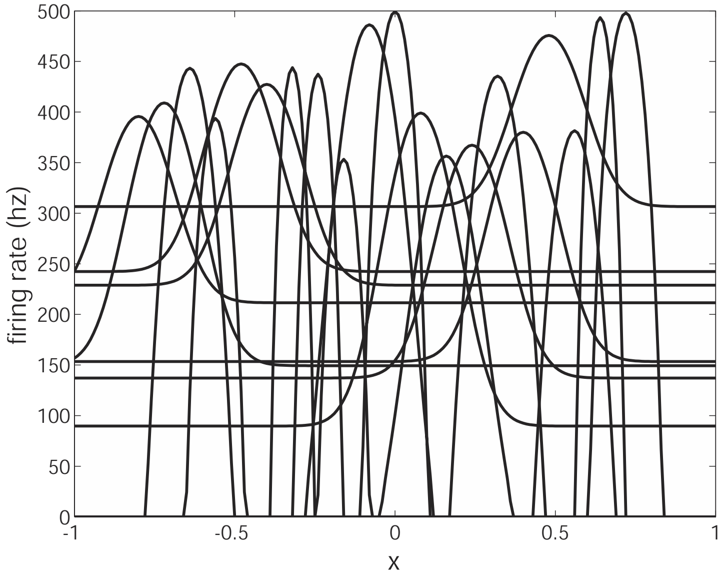


Figure 7.7
Sample tuning curves from the Gaussian tuned population. These are typical cortical tuning curves found, for example, in area LIP (see section 3.4).

term. After that, however, the Gaussian singular values fall off less quickly than those for the linear population. Thus higher frequency transformations, those with many ‘wiggles’, are better encoded by the Gaussian population. Nevertheless, the linear population has a clear advantage for more linear and low-order transformations. We can understand this difference as being a result of the fact that populations with broad tuning curves have many neurons firing at each value of the encoded variable, permitting a good reconstruction of *each* value (i.e., a linear reconstruction). In contrast, Gaussian populations have only a few neurons firing for each value of the variable, making it more difficult to reconstruct a linear function.

In general, then, we can examine the $\chi_m(x)$ functions, which span the active state space of a population, to determine what kinds of transformations are and are not well-supported by that population. This can be extremely useful for constraining hypotheses about the kinds of transformations we expect to see in a *real* neural population once we have a sense of what kind of encoding is being used. Being able to thus generate constraints on the function of neural populations based on easily observable properties (tuning curves in this case), should be a boon to our understanding neurobiological systems.

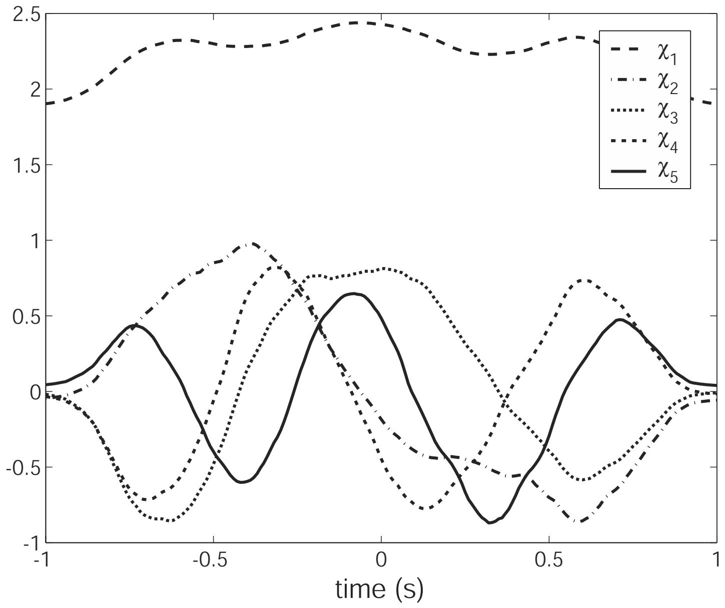


Figure 7.8
The first five $\chi_m(x)$ functions for the population in figure 7.7. The resulting basis is like a Fourier basis defined over a finite interval.

7.4 Quantifying representation

As with transformations, our characterization of neural representation to this point has primarily consisted of examples. As a result, much of the generality of the approach has been missed. For this reason we now turn to a more theoretical discussion of neural representation. The main advantage of taking this abstract point of view is that it allows us to directly examine some of the general properties of neural representation, thus providing deeper insight into the representational characterization of neural systems.

The two properties of neural representation that we concern ourselves with are representational capacity and robustness, or usefulness, of a representation. In this section we define these notions explicitly. In section 7.5, we show how they apply to characterizing neurobiological representation.

7.4.1 Representational capacity

It is natural to suggest that a particular vector $\mathbf{x}$ is 'perfectly' represented by our encoding just in case we can (exactly) get $\mathbf{x}$ back once we have encoded it. In other words, it makes

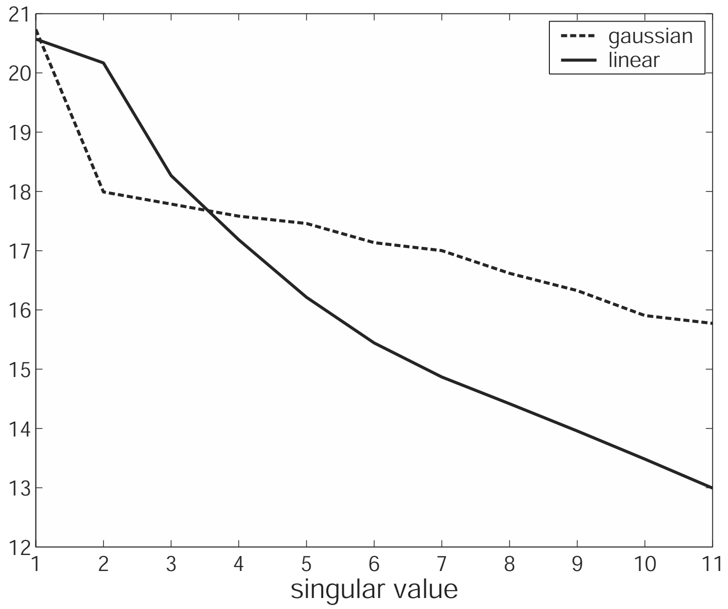


Figure 7.9
Comparison of the change in magnitude of singular values for the neural populations with linear and Gaussian tuning curves. These show that linear and low-order transformations are better handled by the linear population and high frequency transformations are better handled by the Gaussian population. See text for further discussion.

sense to take a system to be 'perfectly' representing a value only when the encoding of that value can be used (i.e., decoded) to represent *that value*. We can generalize this intuitive notion of a 'perfect' representation to give us a preliminary measure of representational capacity. In particular, let us take the capacity of a representation to be the number of values that can be represented with an error of zero.[11] This is clearly different from mean square error (MSE) as a measure of the goodness of a representation. But, since we minimize MSE in finding our representation, they are often closely related (i.e., the higher the capacity, the lower the MSE).

In order to find the capacity of the representation, we begin by writing the difference between our estimate and the original vector (i.e., the error) as

$$\begin{aligned}\mathbf{E} &= (\mathbf{X} - \hat{\mathbf{X}}) \\ &= (\mathbf{X} - \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{X}) \\ &= (\mathbf{I} - \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T)\mathbf{X}, \end{aligned} \tag{7.19}$$

11 Of course noise will makes it technically impossible to recover an exact value. This is why we consider the effects of noise in the next section.

where $\mathbf{I}$ is the identity matrix (i.e., ones on the diagonal, zeros elsewhere). We can see from (7.19) that $\mathbf{E} = \mathbf{0}$ when

$$\mathbf{I} = \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T. \tag{7.20}$$

More specifically, some particular vector, $\mathbf{x}_i$, is perfectly represented when $\mathbf{E}_i = \mathbf{0}$, meaning that the ith row of the estimate is equal to the ith row of the identity matrix $\mathbf{I}$. So, the more rows of the estimate that are equal to their respective identity vectors, the higher the representational capacity of the network whose tuning curves make up the matrix, $\mathbf{A}$. More generally, the more similar the rows are to the identity matrix, the *better* the representation (in the mean-square sense).

This notion of representational capacity extends to both scalars and functions. Most obviously, this is because scalars are 1 dimensional vectors. To carry out the same analysis as above for functions, we can discretize the dimension, ν, much as we did for $\mathbf{x}$ earlier (see section 7.2). Note that in both cases, nothing in the analysis depends on our choice of step size for Δ_ν or $\Delta_\mathbf{x}$. Letting these go to the limit zero, we see that the above results follow for continuous valued functions as well. Having defined representational capacity, it is natural to ask how we can maximize it. We address this question in section 7.5.

7.4.2 Useful representation

First, let us consider how to characterize the space that can be *usefully represented* by the set of neuron tuning curves that make up $\mathbf{A}$. By 'usefully represented' we mean that the representation has a reasonably high signal-to-noise ratio (SNR), which is conventionally taken to mean a SNR of 1 or greater. The reason that the cutoff is usually put at a SNR of 1 is that any output signal at this SNR has an equal chance of being the result of noise or transmitted signal. Beyond this point, information transmission becomes very slow, and thus wasteful of resources (i.e., 'useless'). No matter what cutoff point we choose, we must determine the sensitivity of our representation to noise. It is this sensitivity that determines how good the representation is relative to the chosen cutoff. In order to determine that sensitivity, we must again consider the noiseless system. That is, we must again look carefully at the inverse of γ, i.e., $(\mathbf{A}^T\mathbf{A})^{-1}$ in (7.19).

In particular, the characterization of our estimate of $\mathbf{x}$ that depends on finding the orthogonal basis of γ, can be used to characterize the expected error. Recall, then, that we can write

$$\hat{\mathbf{X}} = \chi\Phi, \tag{7.21}$$

where $\chi = \mathbf{AU}$ and $\Phi = \mathbf{U}^T \phi$. This expression allows us to determine that the residual error is (see appendix E.2)

$$E_r = \left\langle \mathbf{x}^2 \right\rangle_{\mathbf{x}} - \sum_m \frac{\langle \chi_m \mathbf{x} \rangle_{\mathbf{x}}^2}{S_m + \sigma_\eta^2}, \tag{7.22}$$

where S_m is the mth singular value and σ_η^2 is the variance of the Gaussian distributed noise.

We can see from (7.22) by how much the mth basis function, χ_m, reduces the error in our estimate under the influence of noise. Specifically, (7.22) tells us that as the singular value, S_m, approaches the value of the variance of the noise, σ_η^2, the corresponding mth element does not usefully contribute to the representation. This is because the S_m term in (7.22) acts to normalize the effects of the projection onto the non-normalized basis, χ_m. When the noise becomes near the magnitude of that normalization term (i.e., SNR = 1 or less), the projection onto the relevant χ_m becomes 'mis-normalized' and thus contributes incorrectly to the representation. That is, it will *introduce* error into the representation. This tells us that *those basis functions whose corresponding singular value is equal to or smaller than the noise, should not be used if we want a good representation*. Recall that we began this analysis in order to characterize what the 'useful' representational space of some set of neuronal tuning curves is. We now have a precise way of finding this space. The useful representational space is that space spanned by the basis functions, χ_m, whose corresponding singular values, S_m, are greater than the variance of the noise, σ_η^2, affecting a single neuron.

So, given this analysis we can appreciate an intimate connection between the number of singular values we preserve in the γ matrix and the sensitivity to noise. In particular, since any singular values smaller than the amount of noise add little information to our representation, we can truncate or ignore the singular values and their corresponding basis functions below the noise level. Thus, rather than including noise when finding the optimal decoders, we can leave noise out and truncate the singular values appropriately, using just the remaining singular values to perform the necessary inversion (i.e., to find Γ^{-1}). Another way of saying this is to note that we can perform the pseudo-inverse of the γ matrix (which is likely to be singular) using only the singular values greater than the expected noise. We can also take the opposite perspective on finding the inverse and directly choose a number of singular values that we want to preserve. Of course, this determines an assumed noise level in the encoding. Such predictions can be useful for comparing simulation results to what is observed in real systems.

Because standard algorithms that implement SVD result in a decreasing ordering of the singular values, in practice it is easy to establish a 'cutoff' value (equal to the variance of the noise), after which the corresponding bases are simply ignored in the representation. The exact choice of a cutoff is not crucial, because this decomposition is known to give the minimal entropy and the lowest error for whatever limited number of singular values are taken (Watanabe 1965).

Interestingly, it is sometimes the case that truncating the singular values *higher* than the expected noise level results in a better decoding. This is especially likely if the population is being subsequently decoded for only the lowest-order terms. This is because a fairly unreliable encoding of higher-order terms may actually *introduce* error into the decoding of the lower-order functions. If that higher-order information is never used, it can be better to ignore it all together.

In addition, increasing the dimensionality of the input space (e.g., encoding vectors instead of scalars) demands more singular values. As demonstrated in figure 7.6, more singular values are needed to decode the same order function from encodings of higher dimensional spaces. These considerations highlight that there are intimate ties between noise, singular values, the dimensionality of the input space, the usefulness of the representation, and the functions that can be decoded from the population.

7.5 The importance of diversity

In section 7.4, we identified two ways in which we could measure the 'goodness' of a set of neurons for representing some vector, $\mathbf{x}$. First, we defined the representational capacity of a population of neurons as the number of perfect representations (i.e., $\hat{\mathbf{x}} = \mathbf{x}$) it supports. Second, we showed how to determine the space that could be well-represented under noisy conditions by a given population of neurons. In this section, we show how the diversity, or '*heterogeneity*', of the population of neurons relates to its ability to support *good* representations.

When we look at neurobiological systems, heterogeneity is immediately evident (see, e.g., Miller et al. 1991). Neurons seldom have the same tuning curves or response profile as one another. However, there are clearly *similar* neurons. In fact, much of cortex seems to be organized on the principle of having similar neurons near one another. But, 'similarity' and 'sameness' are two very different things. Nevertheless, modelers and theoreticians have often assumed that populations of neurons have *exactly* the same tuning properties and/or response profiles. This is done because it often serves to simplify the relevant mathematics (it is much easier to generalize over sets of the same object, or objects that vary along a few parameters). As well, systems built from devices with very nearly identical response characteristics are far more familiar to many modelers and theoreticians. This is because when it comes to *engineered* devices, we go to great lengths to manufacture components that meet strict standards. These standards (as the name implies) guarantee near identity of responses under specified conditions. Thus, the systems that are built with such components *are* systems that can be analyzed by assuming each component is identical. Our purpose in this section is to show that this common assumption is a poor one in the context of neural representation. In fact, it results in a systematic *under-estimation* of the representational abilities of neural populations.

To show this, let us consider the representational capacity and noise tolerance of populations of leaky integrate-and-fire neurons representing some scalar quantity, x. In order to compare different assumptions about the similarity of these populations, we consider the four different kinds of populations shown in figure 7.10. The first two populations depicted in this figure represent the standard modeling assumptions about neural populations: a) all the neurons are nearly identical; or b) all the neurons have the same tuning curves shifted by some fixed amount. The first assumption is perhaps simplest of all: let all of the devices be (essentially) the same. The second assumption is slightly more sophisticated in that it takes into account the fact that certain nonlinearities in the devices can be countered by using the same device to represent different parts of the total range of the variable. Both of these populations are homogeneous insofar as the properties determining the response function are very similar across the population. The second two populations are more reasonable assumptions about the distribution of *neuron* tuning curves: c) the neurons are evenly spaced over the range of the variable; or d) the neurons are randomly distributed over the range of the variable. However, in both of these cases, the neurons have heterogeneous tuning curves; each of τ^{RC} (RC time constant), τ^{ref} (absolute refractory period), J^{bias} (bias current), and α (gain) vary widely (although within physiological ranges). In fact, assuming the even distribution of x-intercepts is not biologically realistic, but as we show, it provides a suggestion as to what might be the optimal strategy that biological systems are approximating.

We can now compare each of these distributions using the two previously derived measures of representational goodness. The results of such a comparison are shown in figures 7.11 and 7.12. First let us consider figure 7.11, a comparison of representational capacity. Clearly, the even distribution for both the shifted neurons and the heterogeneous neurons perform the best, since their representational capacity scales linearly with the number of neurons. The randomly distributed neurons do not perform as well, and the neurons from the maximum distribution perform worst of all. The reason that the latter two populations do not perform as well is that any linearly dependent tuning curves greatly reduce the representational capacity (as can be seen from (7.20)). Since the randomly chosen x-intercepts have a chance of being the same (or similar enough), they have a chance of resulting in linearly dependent curves that reduce the capacity (and this chance increases with the number of neurons chosen). Since the maximum distribution

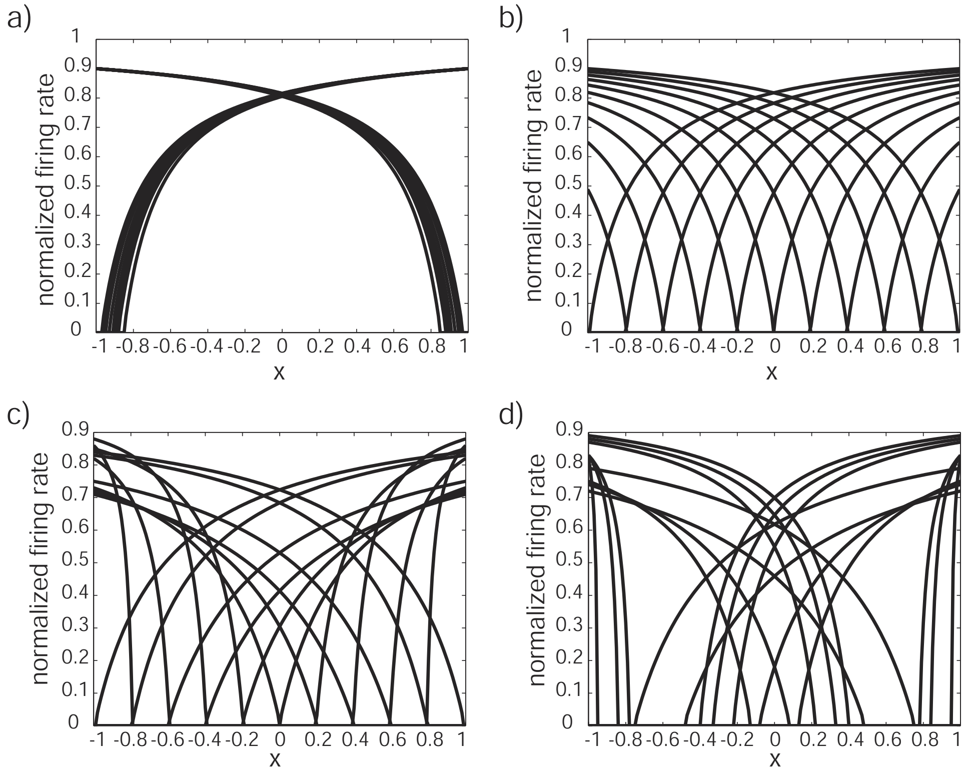


Figure 7.10
Various distributions of x -intercepts in a population of LIF neurons. a) A set of x-intercepts randomly chosen from a distribution in which the intercepts are near the maximum (i.e., the 'maximum' distribution). b) An even distribution of x-intercepts where all of the neuron parameters are otherwise the same. c) An even distribution of x-intercepts with randomly chosen neuron parameters. d) A set of x-intercepts randomly chosen from an even distribution with randomly chosen neuron parameters. Different distributions of x-intercepts that have very different representational properties as discussed in the main text.

constrains the choice of curves far more than the even distribution (and all curves have the same maximum firing rate at $x = \pm 1$), its representational capacity falls off sooner. The reason the capacity in this distribution actually *decreases* as the number of neurons increases is that the chances of any particular neuron *not* being linearly dependent on some other neuron decreases as we continue to choose tuning curves from this same, limited distribution.

In contrast, the population that is randomly chosen from the even distribution fairs quite well. In fact, in the limit of large numbers of neurons, it should perform the same

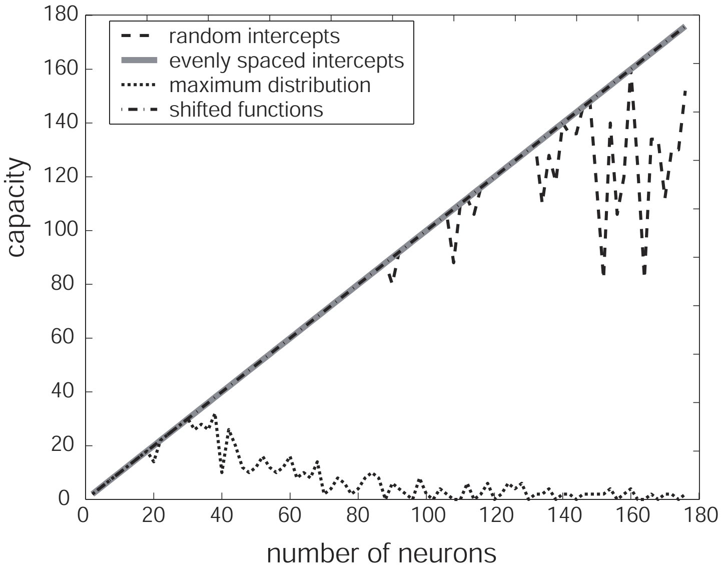


Figure 7.11
Representational capacity as a function of the number of neurons for the different neural populations shown in figure 7.10. The maximum distribution performs the worst. The two even distributions perform about the same. The population with randomly selected x-intercepts performs comparably although as the number of neurons becomes large, redundancy in the random selection becomes more likely and the capacity begins to fluctuate. In the limit, however, this population will have the same capacity as the evenly distributed population.

with respect to representational capacity as neurons that are forced to be exactly evenly spaced. As a result, if there was a high cost associated with enforcing a strict even spacing, relaxing this constraint should not result in much loss in representational capacity. One way to enforce a strictly even spacing in neural systems, would be to encode the precise physiological properties of the neurons in each population. This, of course, is a ridiculously high price for natural systems to pay, especially when an approximation through more-or-less random selection does nearly as well. Thus the heterogeneous tuning curves found in natural systems strikes a balance between representational capacity and the kinds of cost associated with more precise spacing.

Of course, this analysis completely ignores noise, so such conclusions are entirely preliminary. When we turn to examining the usability of these different representations, the fact that the shifted function distribution fairs well with respect to capacity is overshadowed by the fact that it performs significantly worse under noisy conditions. As can be seen from figure 7.12, both the random and the even distributions of realistic, heterogeneous tuning curves are more resilient to noise as the number of neurons increases than either of the other distributions. In particular, they clearly improve at a faster rate.

Worst off, again, is the population whose x-intercepts are chosen from the maximum distribution. This is because adding more of the (nearly) same nonlinear components will not help improve any approximation to a linear function (in this case, the linear function is $x = x$). We cannot, in other words, take any linear combination of the curves depicted in figure 7.10a and get a straight line over all values of x. In fact, the reason engineers can usually assume that adding more of some component *will* improve the performance of the system under noise is that the components *themselves* are quite linear. When this is not the case, as it clearly isn't for neurons, adding more of the same component will not help much at all.

The reason that the shifted neurons eventually perform less well than the more randomly distributed neurons is somewhat more subtle. Clearly, figure 7.12 shows that this strategy is pretty good for small numbers of neurons (i.e., <100).[12] However, as the number of neurons increases beyond these small numbers, the heterogeneous populations do significantly better (and improve at a faster rate). Essentially this is because the heterogeneous populations provide more flexibility for fitting a curve. Perhaps an analogy will help. Suppose that we have the task of fitting a set of curved edged blocks inside a straight edged figure. We have two sets of blocks to choose from. One, the homogeneous set, in which all of the blocks are of the same shape (though of every different size), and one, the heterogeneous set, in which the blocks are randomly picked to be of various shapes (of every different size). Suppose also that the shape in the homogeneous set fits fairly well into the straight-edged figure and that it is one of the possible shapes in the heterogeneous set. Now, when we have a few blocks, the homogeneous set will be quite good (by design) at fitting in the figure. However, in the same case, the heterogeneous set has a low chance of having members whose shape is the same as the members of the homogeneous set, and thus only has a few such shapes. When the number of blocks gets sufficiently large, the homogeneous set still only has one kind of shape. In contrast, the heterogeneous set has all kinds of shapes, many of which are going to be the same as that in the homogeneous set. Thus, the heterogeneous set can fit the straight edged shape at least as well as the homogeneous set *and* it can probably do even better since one of the many *other* shapes will likely

12 We should note here that for the shifted population, we chose a neuron response curve that gave the *best* answer (i.e., best noise characteristics and representational capacity). Many realistic response curves when shifted this way are even worse on this measure than the maximum distribution population. This is because they are more linear over the range of x, but still highly nonlinear when the neurons 'turn on'. As a result, the beginning nonlinearity cannot be compensated for by the remaining, fairly linear response curve. Since we are doing a linear decoding, we can only multiply any neurons in the population by a constant when using them to approximate some function (a line in this case). If we pick a large constant, the error from the nonlinearity will be large. If we pick a small constant, the error from the nonlinearity will be small, but so will the correction to our estimate of the line. This is not true for the population we used because the tuning curves continuously get shallower over the range of x. In sum, the strategy of 'shifting' the exact same function over the range of x is only successful for particular functions, and it is unsuccessful for many functions that look like realistic neuron tuning curves.

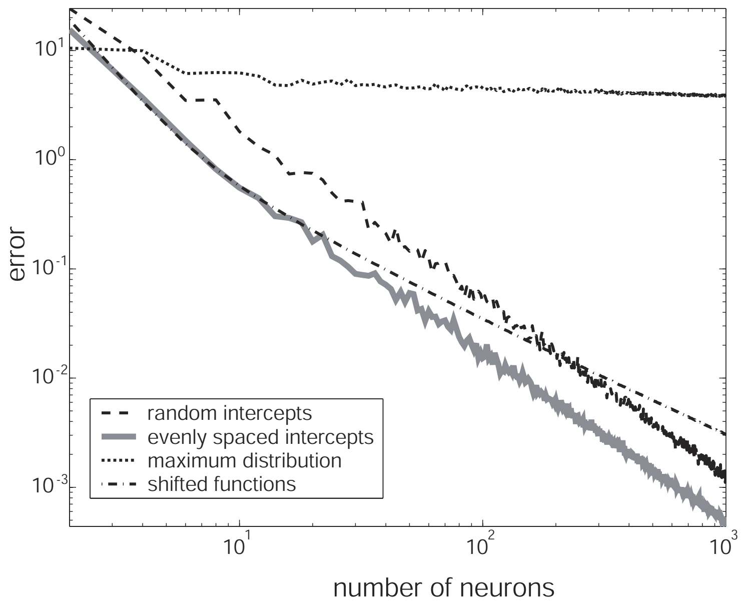


Figure 7.12
The effect of x-intercept distribution on error. Neurons with more diverse x-intercepts are significantly better for estimating x than the same number of neurons all with the same x-intercept. As well, the heterogeneous populations eventually outperform, and improve at a faster rate than, the shifted population. On average, choosing random intercepts from a distribution approaches the enforced even spacing of intercepts and so the two heterogeneous populations do nearly equally well (results averaged over 15 runs).

do a good job of filling in the small holes left by that (homogeneous) shape. This analogy gives a sense of how heterogeneity can be uniquely well-exploited to fit desired functional shapes.

Notice that there is again a clear difference between the two heterogeneous populations as well. As we see from figure 7.12, a population with perfectly evenly spaced x-intercepts has consistently lower error than a population whose x-intercepts are randomly chosen from an even distribution over the range of x (all else being equal). However, the *rate* of decrease in the error is the same for both populations. But, the random population is far more neurobiologically plausible. So, while the evenly spaced distribution is useful for showing a lower bound on the error, it is not likely to be found in real neural systems. Again, the reason is likely that it is far too costly to worry about 'engineering' precisely spaced x-intercepts when random ones do nearly as well. So, the heterogeneity found in real neural populations is both easy to generate and provides for very good representations.

Notice also that our analysis in section 7.3 provides evidence for the importance of heterogeneity to transformations as well. The heterogeneity, or lack thereof, in a given neural population greatly affects the ability of that population to support certain kinds of transformations. In particular, a homogeneous population is one that acts much like the orthogonal basis discussed in section 7.1. That is, it assumes that there is a particular structure to the space to be represented.

This suggests two things: 1) given the observed heterogeneity in real systems, they are likely able to compute a wide variety of functions very well; and 2) approximating real systems by using multiple instances of the same tuning curve is a bad idea since such systems will not support the variety of transformations that are supported in real systems. Rather, homogeneous approximations will 'clump' the encoding of the input space in such a way as to seriously alter the decodable transformations. A less biased tiling of the input space comes for free with heterogeneous response properties. This has important consequences for the kind of information reported by experimentalists about the systems they study. Typically, a few of the 'best' neurons are plotted, analyzed, and discussed in detail in research reports and journal articles. But, what is important for getting a sense of both the representational and transformational power of the system in general, is the *distribution* of the tuning curves in the whole population. It is that distribution that determines the goodness of the representation and the possible transformations it can support. So, as theoretically troublesome and empirically messy as heterogeneity may be at times, it is an important property of neurobiological systems that should not be ignored.

Despite such considerations, an emphasis on the importance of heterogeneity for good representations is scarce in neuroscience. However, it is clear that along with notions like orthogonality and overcompleteness, *heterogeneity* is an important representational category. The heterogeneous population is clearly overcomplete, but so are the shifted and maximum distribution populations. The fact that it is heterogeneous is what matters to good representational capacity, improved noise tolerance, and its ability to support the appropriate transformations.

7.6 Summary

In order to characterize neural representation and transformation generally, we began by comparing the standard algebraic notion of a complete orthogonal basis to the less-typical notion of an overcomplete basis. We showed why overcomplete bases are generally useful for unbaised representation in systems subject to noise. We then described the very powerful technique for analyzing singular matrices called singular value decomposition (SVD). We showed that applying this technique to the γ matrix resulted in the identification of an orthogonal basis, **U**, spanning the space represented by the neurons in a given population. As well, it identifies the diagonal matrix **S** of singular values that is a natural measure of the importance of the corresponding basis vector.

We then employed these techniques to determine how well, and what, transformations could be supported by a given neural population. We showed how the tuning curves of the population determined what transformations could be decoded using that population. Interestingly, for both the linear and Gaussian tuning curves we examined, a familiar set of basis functions emerged.

We then turned to the task of understanding the form of neural representation. We used the preceding characterization to define two general properties of neural representation, representational capacity and usability. By considering four different populations of LIF neurons, we demonstrated that the population that most closely mirrored those found in real systems does a good job of maximizing both of these properties, while minimizing demands for constructing such a population. In particular, we argued that the heterogeneity of the populations was responsible for these desirable results. Thus, heterogeneity is an important property to consider when characterizing neural representation and transformation.

This page intentionally left blank

8 Dynamic transformations

To this point, we have characterized both linear and nonlinear transformations of representations in neurobiological systems. In classical cognitive science, this is all that is needed to completely understand a cognitive system (Newell 1990). However, it is increasingly common for contemporary cognitive scientists to argue that we must add one more ingredient to any theory attempting to provide a satisfactory explanation of cognitive systems; *dynamics*. Not only do we need to know that a system *can* get from one state to another, we also need to know *how long* it takes to do so. In order for a system to successfully perform some functions (e.g., catching a baseball), it must perform them within a certain time frame (e.g., before the ball has moved out of range). So, if we want to know which functions a system can *actually* perform in the real world, we must characterize the system's dynamics.

While it may be understandable that dynamics was initially ignored by those studying cognitive systems,[1] it would be strange, indeed, to leave dynamics out of the study of neurobiological systems. Even the simplest nervous systems performing the simplest functions demand temporal characterizations (e.g., locomotion, digestion, sensing, evasion, stalking, mating, etc.). As a result, single neurons have almost always been modeled by neuroscientists as *essentially* dynamic systems (see Bower and Beeman 1995, or chapter 4 for a review). As well, electrophysiologists analyze cellular responses in terms of 'onsets', 'latencies', 'stimulus intervals', 'steady states', 'decays', etc.—these are all terms describing the temporal properties of a neuron's response. The fact is, the systems under study in neurobiology *are* dynamic systems and as such they make it very difficult to ignore time. The fundamental importance of time to understanding neural systems is not surprising as these are systems that have evolved in dynamic environments to which their own dynamics must be closely matched; otherwise they would not be here to study.

Engineers have also had a longstanding interest in understanding dynamic systems. In order to build something that works in the real world, the dynamics of the real world must be taken into account. For this reason, many useful quantitative tools have been developed by mathematicians, physicists, and engineers for the study of dynamics. Chief amongst these is linear systems theory, the study of 'simple' or 'simplified' dynamic systems. These systems are simple precisely because they are linear. Qualitatively speaking, however, they can be highly complex, sophisticated, and very useful kinds of systems to understand. In fact, very few real-world systems are truly linear. But, in practice, many are well-

1 Dynamics may have been initially ignored because classical cognitive science relied heavily on the 'mind as computer' metaphor. For Turing machines, which were taken to define the class of computers, time is utterly irrelevant.

approximated by a linear model. As a result, linear systems theory has found broad application, ranging over areas as diverse as economics, circuit analysis, aeronautics, communications, and robotics. It has, in fact, been applied to nearly every area of study by mechanical, electrical, systems, chemical, and industrial engineers.

An important subfield in linear systems theory is linear control theory. Linear systems theory provides a means of *analyzing* linear systems, whereas linear control theory provides a means of designing and *synthesizing* such systems. Linear control theory is concerned with notions like the stability, controllability, invertability, and observability of linear systems. In order for such notions to be of much use, it is generally necessary to perform a detailed analysis of the system under consideration. Like neurobiologists, then, control theorists need to analyze real-world dynamic systems. Perhaps these disciplines can share an approach to understanding such systems and learn from each other in the process.

The fact that control theorists and neurobiologists have many of the same interests has not gone unnoticed. Perhaps most famously, the interdisciplinary movement founded in the early 1940s known as 'cybernetics' was based on precisely this insight (Weiner 1948; Rosenblueth et al. 1943). Of the more well-known figures in the movement, two were neurophysiologists (Warren McCulloch and Arturo Rosenblueth), one a mathematicians (Norbert Weiner) and one an engineer (Julian Forrester). Cybernetics was inspired by the observation that closed-loop negative feedback could be used to guide a system to its goal. Furthermore, cyberneticists claimed that living systems were goal-oriented systems. Thus, closed-loop control, it was argued, should be a good way to understand the behavior of living systems. However, much of the focus of cybernetics was on characterizing only the output behavior of systems, not their internal processes. At the time, control theory (now deemed 'classical' control theory) focused solely on the *external*, input/output description of a system. With the so-called 'cognitive revolution' of the mid-1970s, interest in cybernetics waned due in part to its close association with behaviorism (also focused solely on input/output descriptions), which fell out of intellectual favor.

In the 1960s, many of the limitations of classical control theory were rectified with the introduction of what is known as 'modern' control theory (Lewis 1992). Modern control theory introduced the notion of an *internal* system description; i.e., one that includes system state variables as part of the description. It is interesting that with the cognitive revolution researchers interested in the behavior of living systems realized they needed to 'look inside' the systems they were studying and, at about the same time, researchers interested in controlling engineered systems began to 'look inside' as well. However, most of those interested in human behavior adopted the digital computer as a metaphor for the workings of the mind. The ubiquity of this metaphor has served to distance the cognitive sciences from modern control theory. Nevertheless, modern control theory offers tools better suited than computational theory to understanding biological systems as fundamentally temporal, dynamic systems operating in changing, uncertain environments. For this reason, we adopt modern control theory as a means of understanding neurobiological systems.

After a brief introduction to modern control theory, we spend the remainder of the chapter developing successively more involved biological examples. These examples show how control theory is a natural ally of the account of representation and transformation developed so far.

8.1 CONTROL THEORY AND NEURAL MODELS

8.1.1 Introduction to control theory

There are two equations that effectively summarize linear control theory:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \tag{8.1}$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t). \tag{8.2}$$

These equations constitute what is known as the *internal description* of a linear system. Together, they are often called the *state equations* of a system because the vector $\mathbf{x}(t)$ consists of variables that together describe the (internal) state of the system. It is the dynamics of these variables that is of the most interest. This is because the *state vector*, $\mathbf{x}(t)$, serves to summarize the effects of all past input. Thus, all future output depends only on the current value of the state vector and the future input. In a sense, then, the state vector is a bridge between the past input and the future output.

The remaining variables in these equations are understood as follows (see figure 8.1): $\mathbf{u}(t)$ is the input or control vector, $\mathbf{y}(t)$ is the output vector, $\mathbf{A}$ is the dynamics matrix, $\mathbf{B}$ is the input matrix, $\mathbf{C}$ is the output or sensor matrix, and $\mathbf{D}$ is the feedthrough matrix. The dimensions of each of the vectors can be different, but these dimensions determine the corresponding matrix dimensions. Thus, there can be multiple inputs, outputs, and state variables, whose interactions are determined by the elements of the matrices.

The system depicted in figure 8.1 is called *time-invariant* because none of the matrices are functions of time. This means that the parameters controlling the dynamics of the system are taken not to change over the period of interest. This is often a useful simplifying assumption, but it is clearly unrealistic in describing neurobiological systems. We discuss both time-invariant and time-varying systems (see section 8.2). One assumption that we do not relax is that the systems we are interested in are *lumped* systems. A lumped system is one whose state variables are finite. A system whose state has infinitely many variables

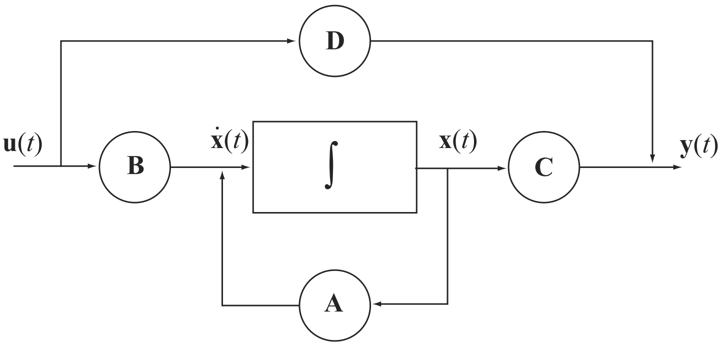


Figure 8.1
A generic block diagram for a time-invariant linear system showing the elements of equations (8.1) and (8.2). In this diagram $\mathbf{x}(t)$ is the state vector, $\mathbf{u}(t)$ is the input or control vector, $\mathbf{y}(t)$ is the output vector, $\mathbf{A}$ is the dynamics matrix, $\mathbf{B}$ is the input matrix, $\mathbf{C}$ is the output or sensor matrix, and $\mathbf{D}$ is the feedthrough matrix.

is called a *distributed* system. Of course, any distributed system can be arbitrarily well approximated by a sufficiently large lumped system. The examples we deal with are all reasonably small lumped systems.

8.1.2 A control theoretic description of neural populations

Given the ubiquity of nonlinear transformations in neurobiological systems, it is far from clear that such linear analyses will be of use. This, in fact, is not a problem unique to neurobiological systems. Because there is no general theory for analyzing nonlinear systems, the study of such systems is generally based on linear systems theory. We adopt this same strategy. So, while we consider both linear and nonlinear neurobiological systems, we always base our analyses on linear control theory.

The means by which we integrate control theory and our previous analyses of representation and transformation is captured by the third principle of neural engineering: neural dynamics can be characterized by taking neural representations to be control theoretic state variables. In other words, elements of the state vector, $\mathbf{x}$, (i.e., the state variables) *just are* the representations we find in neural populations. And, the relations between these variables (as defined by the system matrices) *are* the transformations implemented by these populations. However, we cannot take the control theoretic descriptions of the relations between variables at face value. Rather, we need to relate the intrinsic dynamics of neural populations to the standard control theoretic characterization. This allows for a 'translation' between standard control theory and a neurally relevant system description.

In figure 8.1, there are four matrices and a transfer function. In order for this to be a dynamic system, the transfer function (integration in this figure) needs to be a function of time. Given how we have so far characterized neural function, there are two possibilities for transfer functions in a neural population; that is, there are two parts of the system that are inherently dynamic. One is the synaptic filter, $h(t)$, and the other is the soma voltage, $V(t)$. Surprisingly, we can show that the synaptic filter generally dominates the dynamics of the cellular response as a whole (see appendix F.1).

As mentioned earlier, a good approximation to synaptic dynamics is given by (see section 4.4.2)[2]

$$h(t) = \frac{1}{\tau} e^{-t/\tau}, \tag{8.3}$$

where τ is the synaptic time constant. This function describes the postsynaptic current (PSC) produced by the arrival of a spike and, of course, it is the linear decoder we discussed extensively in chapter 4. So, we need to characterize the dynamics of neural systems with respect to this decoder.

We can begin by drawing a 'neural' control diagram (see figure 8.2), like that in figure 8.1. This diagram is missing the feedthrough and output matrices from figure 8.1. However, in the case of a neural system, where each such subsystem describes a single population, both the feedthrough matrix and the output matrix can be taken to be incorporated into the input matrix of a subsequent population. Thus, these two matrices need not be explicitly considered here. Note also that the original integration is replaced by $h(s)$, the Laplace transform of the synaptic dynamics, $h(t)$, in equation (8.3).

It is very convenient to use the Laplace transform when characterizing dynamics with differential equations. Essentially, this transform provides a means of writing and manipulating differential equations algebraically. Much like the Fourier transform, the independent variable for the Laplace transform, s, can be thought of as frequency. The following equation defines the Laplace transform:

$$\begin{aligned} \mathcal{L}(f(t)) &= \int_0^\infty e^{-st} f(t) dt \\ &= f(s). \end{aligned}$$

Here, $f(t)$ is in the time domain and $f(s)$ is in the frequency domain.

Taking the Laplace transform of the time-invariant internal description, equations (8.1) and (8.2), gives

2 This is a significant simplification, though a good approximation to true PSCs. In particular, it ignores the rapid (yet finite) rise in the observed PSC. We have used various forms for this filter in numerical experiments and have found that including a finite rise time tends to improve the dynamics of the simulation. However, the increased complexity of analyzing more faithful models serves to obscure the insights that can be gained by (at least initially) considering a simpler model. Nevertheless, similar techniques can be employed to analyze other synaptic models.

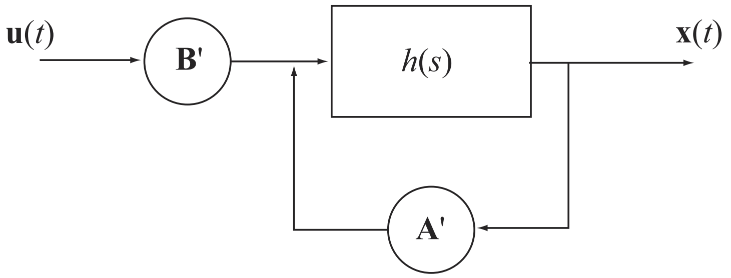


Figure 8.2
A generic neural population as a linear system. Note that the feedthrough matrix and output matrix are not included as part of the figure because both can be incorporated into the input matrix of the subsequent population. $\mathbf{A}'$ and $\mathbf{B}'$ are the *neural* dynamics and *neural* input matrices respectively. The transfer function, $h(s)$, is the Laplace transform of the synaptic dynamics.

$$\begin{aligned} s\mathbf{x}(s) &= \mathbf{A}\mathbf{x}(s) + \mathbf{B}\mathbf{u}(s) \qquad (8.4) \\ \mathbf{y}(s) &= \mathbf{C}\mathbf{x}(s) + \mathbf{D}\mathbf{u}(s). \qquad (8.5) \end{aligned}$$

Given figure 8.2, we can see that equation (8.5) will always be $\mathbf{y}(s) = \mathbf{x}(s)$ for a neural subsystem, so we focus on equation (8.4). The system described by figure 8.2 can be written as

$$\mathbf{x}(t) = h(t) * [\mathbf{A}'\mathbf{x}(t) + \mathbf{B}'\mathbf{u}(t)]$$

or, in the frequency domain,

$$\mathbf{x}(s) = h(s)\left[\mathbf{A}'\mathbf{x}(s) + \mathbf{B}'\mathbf{u}(s)\right]. \qquad (8.6)$$

The Laplace transform, $h(s)$, of $h(t)$ as given by (8.3) is

$$h(s) = \frac{1}{1 + s\tau}.$$

Therefore,

$$\begin{aligned} \mathbf{x}(s) &= \frac{1}{1+s\tau}\left[\mathbf{A}'\mathbf{x}(s) + \mathbf{B}'\mathbf{u}(s)\right] \\ &= \frac{\tau^{-1}}{\tau^{-1}+s}\left[\mathbf{A}'\mathbf{x}(s) + \mathbf{B}'\mathbf{u}(s)\right]. \end{aligned}$$

So,

$$\begin{aligned} (\tau^{-1} + s)\mathbf{x}(s) &= \tau^{-1}\left[\mathbf{A}'\mathbf{x}(s) + \mathbf{B}'\mathbf{u}(s)\right] \\ s\mathbf{x}(s) &= \tau^{-1}\left[\mathbf{A}' - \mathbf{I}\right]\mathbf{x}(s) + \tau^{-1}\mathbf{B}'\mathbf{u}(s). \qquad (8.7) \end{aligned}$$

Equating the right-hand sides of (8.4) and (8.7) defines the relation between the dynamics and input matrices for the standard control system and the neural control system. Namely,

$$\mathbf{A}' = \tau\mathbf{A} + \mathbf{I} \tag{8.8}$$

$$\mathbf{B}' = \tau\mathbf{B}, \tag{8.9}$$

where **I** is the appropriately dimensioned identity matrix.

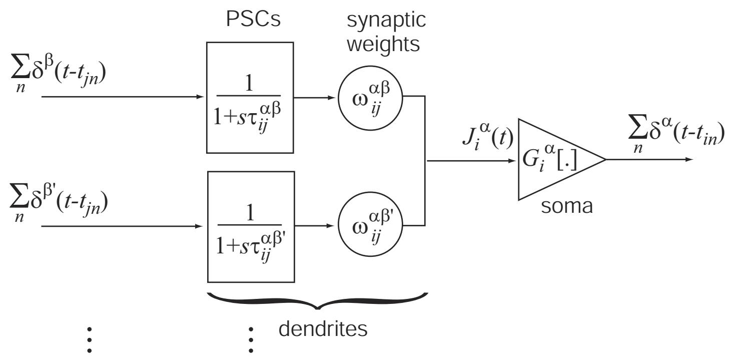
As simple as these equations are, given a control system in the standard form of figure 8.1, we can use them to determine the equivalent neural system in the standard form of figure 8.2. We present a detailed example of this in section 8.2, where we re-examine the neural integrator previously presented in sections 2.3 and 5.3.

8.1.3 Revisiting levels of analysis

Before examining specific examples that use this relation, let us consider how the preceding analysis can help articulate the theory as it has been presented so far. Recall that in chapter 2 we introduced the distinction between basic representations (i.e., neural activities) and higher-level representations (i.e., encoded physical properties). In this section we examine how higher-level and basic representations can be related using the control theoretic approach just discussed. As well, we show how to move easily between descriptions employing these different levels of representation. As a result, we characterize a generic neural subsystem that can be combined to construct large, complex models with various degrees of neural realism (possibly within the same model).

To understand the subsequent discussion, a few comments on notation are in order. First, we adopt the conventions from linear control theory that, in systems diagrams: 1) boxes denote transfer functions (which define the dynamics of the system); 2) circles denote matrices that multiply their inputs; and 3) intersecting lines indicate addition. We also introduce a convention to use a rightward pointing triangle to indicate a function of the input that may be nonlinear. As well, we use Greek superscripts to index *populations*, and Roman subscripts to index *neurons* within a population.

Based on the discussion in the previous section, we can draw a standard subsystem at the basic level (see figure 8.3). Here we have a system that is consistent with typical neuroscientific descriptions of neuron function. Spike trains, $\sum_n \delta^\beta(t - t_{jn})$, from various preceding populations, β, arrive at the synaptic cleft and, by the action of neurotransmitters, induce a current change, $\omega_{ij}^{\alpha\beta} h(t)$, in the postsynaptic dendrites of neurons in population α. These resulting postsynaptic currents (PSCs) are filtered versions of the presynaptic spike train, where the filter is well-approximated by $1/1 + s\tau_{ij}^{\alpha\beta}$. Either at the time of filtering or due to various dendritic processes, these PSCs can have varying degrees of effect on the changes in the somatic current, $J_i^\alpha(t)$. The synaptic weights, $\omega_{ij}^{\alpha\beta}$, model this effect. There



PSCs
synaptic weights
$\sum_n \delta^{\beta}(t-t_{jn})$
$\frac{1}{1+s\tau_{ij}^{\alpha\beta}}$
$\omega_{ij}^{\alpha\beta}$
$\sum_n \delta^{\beta'}(t-t_{jn})$
$\frac{1}{1+s\tau_{ij}^{\alpha\beta'}}$
$\omega_{ij}^{\alpha\beta'}$
$J_i^{\alpha}(t)$
$G_i^{\alpha}[.]$
soma
$\sum_n \delta^{\alpha}(t-t_{in})$
dendrites

Figure 8.3
The basic-level analysis as a system diagram. Spike trains from presynaptic neurons are shown on the left and the output spike train is shown on the right. The latter is the result of weighted dendritic PSCs that are passed through a spiking nonlinearity in the soma. See text for further explanation.

is then a highly nonlinear process, $G_i^\alpha[\cdot]$, which can be modeled in any number of ways (see section 4.5), that results in a series of output spikes, $\sum_n \delta^\alpha(t - t_{in})$. These spikes travel to subsequent neural populations. Of course, subsequent populations can be described using this same diagram, since the inputs and outputs are both neural spike trains. Thus we can build up basic-level descriptions of complex neural systems using combinations of this subsystem.

The main problem with adopting *solely* this level of description is that there is enormous flexibility in choosing weight matrices. This is where the ability to relate high-level transformations to weight matrices eventually becomes important. But suppose, for the time being, that we did not know how to relate weight matrices to higher-level descriptions. We still might have a good idea of what the high-level transformations are in a neural system (recall the vestibular system example in section 6.5). In such a case we would be in a good position to generate a control theoretic system description, since we would know both the dynamics and the representations in the system. Figure 8.4 shows the result of such a characterization.

At this level of description, we define the input to the subsystem as some (possibly nonlinear) function of the variables we take to be received by the neural system. This input is filtered by the population filters, $h^{\alpha\beta}(t)$, and may also be transformed in some manner by $\mathbf{M}^{\alpha\beta}$. Notably, $\mathbf{M}^{\alpha\beta}$ can be either the input matrix or a dynamics matrix, depending on the origin of the signal, $\mathbf{x}^\beta(t)$ (i.e., it will be the dynamics matrix for $\alpha = \beta$ and an input matrix otherwise). The result is the variable, $\mathbf{x}^\alpha(t)$, that is represented by the current neural population. This signal is then sent to subsequent populations that can be characterized using this same high-level description.

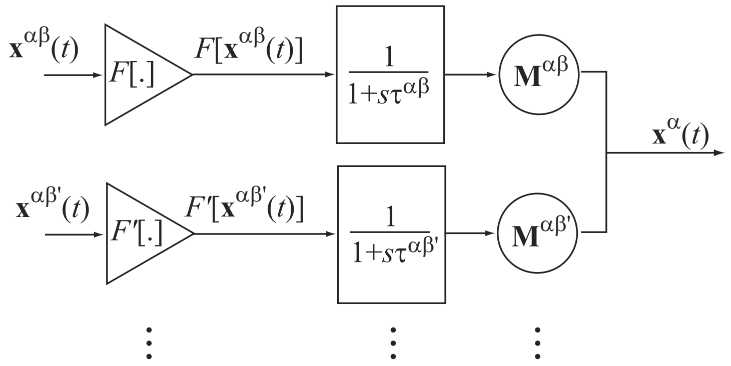

$\mathbf{x}^{\alpha\beta}(t)$
$F[.]$
$F[\mathbf{x}^{\alpha\beta}(t)]$
$\frac{1}{1+s\tau^{\alpha\beta}}$
$\mathbf{M}^{\alpha\beta}$
$\mathbf{x}^{\alpha}(t)$
$\mathbf{x}^{\alpha\beta'}(t)$
$F'[.]$
$F'[\mathbf{x}^{\alpha\beta'}(t)]$
$\frac{1}{1+s\tau^{\alpha\beta'}}$
$\mathbf{M}^{\alpha\beta'}$

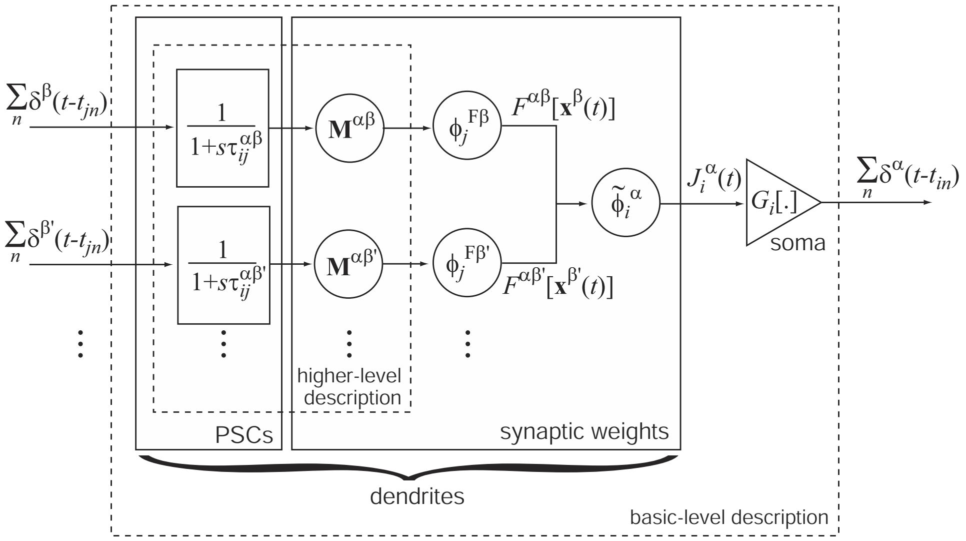
Figure 8.4
The higher-level analysis as a system diagram. The input from previous populations is decoded, passed through the population filter, and then transformed by the relevant control matrices. Note that we can introduce the $\mathbf{M}^{\alpha\beta}$ matrices and the $h^{\alpha\beta}(s)$ filters in any order because both are linear transformations. We have chosen to write the matrix second, in order to be consistent with the basic-level diagram above. See text for further explanation.

Given our previous characterizations of both representation and transformation, we have the tools to reconcile these two descriptions (see figure 8.5). Essentially, this figure combines figures 8.3 and 8.4 and provides more detail regarding the origin of the synaptic weights. In figure 8.5, the solid-line boxes highlight the dendritic components, which have been separated into postsynaptic filtering by the PSCs, and the synaptic weights. The weights themselves are determined by three matrices: 1) the decoding matrix whose elements are the decoders, $\phi_j^{F\beta}$, for some (possibly nonlinear) function, F, of the signal, $\mathbf{x}^{\beta}(t)$, that comes from a preceding population, β; 2) the encoding matrix whose elements are the encoders, $\tilde{\phi}_i^{\alpha}$, for this population; and 3) the higher-level transformation matrix, $\mathbf{M}^{\alpha\beta}$, that defines the transformations relating populations α and β. The most significant change between this and the preceding two figures is that the nonlinear functions, F, are decoded after being filtered by the PSCs. Because the decodings and filtering are all linear, they can be written in any order.

As a brief review, it is useful to relate this figure to the equations we have presented so far. First, consider the expression for the spike generating mechanism for neurons, i, in the population α:

$$\sum_k \delta^{\alpha}(t - t_{ik}) = G_i\left[J_i^{\alpha}(t)\right].$$

As usual, we take the somatic current, $J_i^{\alpha}(t)$, to be determined by a dot product of the



$\sum_n \delta^{\beta}(t\text{-}t_{jn})$
$\sum_n \delta^{\beta'}(t\text{-}t_{jn})$
$\frac{1}{1+s\tau_{ij}^{\alpha\beta}}$
$\frac{1}{1+s\tau_{ij}^{\alpha\beta'}}$
$\mathbf{M}^{\alpha\beta}$
$\mathbf{M}^{\alpha\beta'}$
$\phi_j^{F\beta}$
$\phi_j^{F\beta'}$
$F^{\alpha\beta}[\mathbf{x}^{\beta}(t)]$
$F^{\alpha\beta'}[\mathbf{x}^{\beta'}(t)]$
$\tilde{\phi}_i^{\alpha}$
$J_i^{\alpha}(t)$
$G_i[.]$
soma
$\sum_n \delta^{\alpha}(t\text{-}t_{in})$
higher-level description
PSCs
synaptic weights
dendrites
basic-level description

Figure 8.5
A generic neural population systems diagram. This figure is a combination of the higher-level and basic-level system diagrams (denoted by dotted lines). The solid-line boxes highlight the dendritic elements. The synaptic weights are shown decomposed into the relevant matrices. See text for further discussion.

neuron's encoding vector with the variable being represented by the population:

$$J_i^{\alpha}(t) = \alpha_i \left\langle \tilde{\phi}_i^{\alpha} \mathbf{x}^{\alpha}(t) \right\rangle_m + J_i^{\alpha\, bias}. \tag{8.10}$$

Now, to determine an expression for $\mathbf{x}^{\alpha}(t)$, we can look to equation (8.6) and see that we can generalize that expression to be

$$\mathbf{x}^{\alpha}(s) = \sum_{\beta} h^{\alpha\beta}(s)\mathbf{M}^{\alpha\beta}\mathbf{x}^{\beta}(s).$$

So,

$$\mathbf{x}^{\alpha}(t) = \sum_{\beta} h^{\alpha\beta}(t) * \mathbf{M}^{\alpha\beta}\mathbf{x}^{\beta}(t),$$

or, because we need not constrain ourselves to linear systems, we can write more generally

$$\mathbf{x}^{\alpha}(t) = \sum_{\beta} h^{\alpha\beta}(t) * \mathbf{M}^{\alpha\beta}F^{\alpha\beta}[\mathbf{x}^{\beta}(t)].$$

Thus, $\mathbf{M}^{\alpha\beta}$ is the input matrix relating the output of population β to the state of population α. As mentioned earlier, the dynamics matrix, $\mathbf{A}'$, in (8.6) is simply the special case, $\mathbf{M}^{\alpha\alpha}$.

Given the work of the previous two chapters, we can compute (nonlinear) functions, F, of the preceding population's input as follows:

$$\begin{aligned} F^{\alpha\beta}[\mathbf{x}^{\beta}(t)] &= \sum_j \boldsymbol{\phi}_j^{\alpha F\beta} \sum_{i,n} h_{ij}^{\alpha\beta}(t) * \delta(t - t_{ijn}^{\beta}) \\ &= \sum_{i,j,n} \boldsymbol{\phi}_j^{\alpha F\beta} h_{ij}^{\alpha\beta}(t - t_{ijn}^{\beta}), \end{aligned}$$

where t_{ijn}^{β} indicates the nth spike time from population β to the current population that goes from neuron j in β to i in α. Similarly, $h_{ij}^{\alpha\beta}(t)$ indicates the PSC in the dendrite of neuron i of α to which neuron j in β projects. The decoders, $\boldsymbol{\phi}_j^{\alpha F\beta}$ are the optimal linear decoders for the function, $F^{\alpha\beta}$, of the output of population β that projects to α. We can now substitute our expression for $\mathbf{x}^{\alpha}(t)$ into (8.10):

$$\begin{aligned} J_i^{\alpha}(t) &= \alpha_i \left\langle \tilde{\boldsymbol{\phi}}_i^{\alpha} \sum_{\beta} h^{\alpha\beta}(t) * \mathbf{M}^{\alpha\beta} F^{\alpha\beta}[\mathbf{x}^{\beta}(t)] \right\rangle_m + J_i^{\alpha\, bias} &(8.11) \\ &= \alpha_i \left\langle \tilde{\boldsymbol{\phi}}_i^{\alpha} \sum_{j,n,\beta} h^{\alpha\beta}(t) * \mathbf{M}^{\alpha\beta} \boldsymbol{\phi}_j^{\alpha F\beta} h_{ij}^{\alpha\beta}(t - t_{ijn}^{\beta}) \right\rangle_m + J_i^{\alpha\, bias}. &(8.12) \end{aligned}$$

As usual, we can simplify this expression by constructing the connections weights:

$$\omega_{ij}^{\alpha\beta} = \left\langle \tilde{\boldsymbol{\phi}}_i^{\alpha} \mathbf{M}^{\alpha\beta} \boldsymbol{\phi}_j^{\alpha F\beta} \right\rangle_m .$$

Note, however, that in (8.12) we now have *two* linear filters, $h^{\alpha\beta}(t)$ and $h_{ij}^{\alpha\beta}(t)$, even though they are of the same form. Conveniently, they can be combined to give a third filter, $g_{ij}^{\alpha\beta}(t) = h^{\alpha\beta}(t) * h_{ij}^{\alpha\beta}(t)$.[3] This gives a compact expression for the soma current:

$$J_i^{\alpha}(t) = \alpha_i \sum_{j,n,\beta} \omega_{ij}^{\alpha\beta} g_{ij}^{\alpha\beta}(t - t_{ijn}^{\beta}) + J_i^{\alpha\, bias}.$$

Fortunately, we can simplify this expression while preserving the expected behavior and greatly reducing the computational load. To begin, we note that the convolution of the

3 In fact, we could avoid there being two linear filters by taking $\mathbf{x}(t)$ to be the weighted neuronal spike trains themselves. Doing so means that $F[\mathbf{x}(t)] = \sum_{jn} \boldsymbol{\phi}_j^F \delta(t - t_{jn})$, so that only one $h(t)$ would be present in (8.12). This definition of $\mathbf{x}(t)$ is not obviously consistent with the definition of neural representation presented in chapter 4. However, given that real neural spikes are not infinitely thin delta functions, these two ways of understanding $\mathbf{x}(t)$ approach one another for large populations. Most importantly of all, both approaches work very well for constructing large-scale models.

two filters, $h(t)$, results in a filter, $g(t)$, that looks much like the original $h(t)$. The main difference is that there is a larger attenuation of very high frequency information by $g(t)$ (i.e., it does not have an infinitely steep onset, as $h(t)$ does). Nevertheless, we can use the original $h(t)$ in place of the $g(t)$ and get similar behavior. In fact, the $h(t)$ filter generally serves to make the system *less* stable over time, so designing stable systems using this filter is more difficult than using the $g(t)$ filter. As well, this substitution guarantees that the previous analyses we have done using $h(t)$ as the filter will apply to any systems we model.

In addition to this substitution, we can gain significant computational savings by ignoring, at least for the time being, the complexity introduced to the system model by the subscripts i and j on the filter. In fact, when we run simulations at the higher-level, we have no choice but to ignore these subscripts since we can only define one $h(t)$ per population. In other words we must assume that the filters $h_{ij}^{\alpha\beta}(t)$ between all neurons in β and all neurons in α are the same. For simplicity, we adopt this assumption at the basic level as well. We do, however, allow that the filters on the projections from various β to α may be different.

Despite this simplification in practice, the equations and the systems diagrams are general. That is, they are equally suited to complex and simple applications. As a result of these simplifications, the expression for the soma current becomes

$$J_i^{\alpha}(t) = \alpha_i \sum_{j,n,\beta} \omega_{ij}^{\alpha\beta} h(t - t_{ijn}^{\beta}) + J_i^{\alpha\, bias}.$$

Although the somewhat Baroque notation may serve to obscure how to apply the theory, it should make it clear how flexible the theory can be.

8.1.4 Three principles of neural engineering quantified

In chapter 1, we mentioned that in order for the principles of neural engineering to underwrite a theory of neurobiology, they must be quantitatively expressed. We are now in a position to do just that. As we have already shown the relation between scalars, vectors, and functions, we only present these principles as they relate to vectors.

Principle 1

Neural representations are defined by the combination of nonlinear encoding and weighted linear decoding.

Neural encoding is defined by

$$\sum_n \delta(t - t_{in}) = G_i \left[\alpha_i \left\langle \tilde{\boldsymbol{\phi}}_i \mathbf{x}(t) \right\rangle_m + J_i^{bias} \right].$$

Neural decoding is defined by

$$\hat{\mathbf{x}}(t) = \sum_i a_i(\mathbf{x}(t))\boldsymbol{\phi}_i^{\mathbf{x}},$$

where

$$\begin{aligned} a_i(\mathbf{x}(t)) &= \sum_n h_i(t) * \delta(t - t_{in}) \\ &= \sum_n h_i(t - t_{in}). \end{aligned}$$

In both cases, i indexes neurons in population and n indexes spikes transmitted from one population to another.

Principle 2

Transformations of neural representations are functions of the variable that is represented by the population. Transformations are determined using an alternately weighted linear decoding.

Assuming the encoding in principle 1, we can estimate a function of $\mathbf{x}(t)$ as

$$\hat{f}(\mathbf{x}(t)) = \sum_i a_i(\mathbf{x}(t))\boldsymbol{\phi}_i^{f},$$

where, $a_i(\mathbf{x}(t))$ is defined as before. The only difference between this decoding and the representational decoding are the decoders themselves, $\boldsymbol{\phi}_i^{f}$.

Principle 3

Neural dynamics are characterized by considering neural representations as control theoretic state variables. Thus, the dynamics of neurobiological systems can be analyzed using control theory.

Allowing $\mathbf{x}(t)$ to be a state variable and $\mathbf{u}(t)$ to be the input, we have the following general expression for the encoding:

$$\sum_n \delta(t - t_{in}) = G_i\left[\alpha_i \left\langle \tilde{\boldsymbol{\phi}}_i \left(h_i(t) * [\mathbf{A}'\mathbf{x}(t) + \mathbf{B}'\mathbf{u}(t)]\right)\right\rangle_m + J_i^{bias}\right].$$

To get a better understanding of how these principles can be used together, let us consider a simple example in some detail.

8.2 An example: Controlling eye position

In sections 2.3 and 5.3 we discussed the relevant anatomy and physiology of the areas of the brainstem implicated in controlling eye position. Here, we adopt the same system description and design specification and focus on the addition of dynamics to this network.

As mentioned in 2.3, this area of the brain is often called the 'neural integrator'. The reason is simple: it acts to integrate a velocity command, $u(t)$, and produce a displacement command, $x(t)$, that is sent to the motor neurons controlling eye position, i.e., $x(t) = \int u(t)dt$. The evidence that the nuclei prepositus hypoglossi (NPH) contains a neural integrator for horizontal eye position is impressively consistent (see Fukushima et al. 1992 for a review).

However, this system does not act as a perfect integrator. In a set of psychophysical experiments, Hess et al. (1985) show that the drift of horizontal eye position in the dark in humans, has a time constant of between 20 s and ~100 s, with a mean of 70 s. They suggest that drift could be understood as being comprised of two components: gaze-dependent drift, which was centripetal (towards the center); and 'constant drift' which was unidirectional. Together, these two components result either in centripetal drift or unidirectional drift that progressively slows as eye position moves across center. Notably, the eye traces in these experiments show many small corrective saccades, so it is likely that systems other than just the NPH integrator are responsible for this behavior. Nevertheless, these experiments suggest that the integrator is quite good, being able to help sustain an eye displacement for long periods of time. Even in the much simpler eye position integrator found in the goldfish (~40 neurons), time constants of about 10 s have been observed (Seung et al. 2000). Work in this system has shown that the integrating behavior cannot be entirely explained by molecular mechanisms (Askay et al. 2001). So, network *structure* is central to understanding this system's behavior.

While generating networks that perform reasonably good integration using spiking neurons is not novel (see, e.g., Seung and Sompolinsky 1993), it provides an excellent, simple example that relies on all three principles we have discussed. As well, large parts of the ensuing discussion do not focus on horizontal eye position integration in particular, but rather consider the more general problem of constructing integrators. This is partly motivated by the fact that building integrators has long been recognized as a difficult, yet central, problem in analog computing. It is also motivated by the importance of stability, noise reduction, and other computational properties of integrators that are essential for complex behavior in neural systems (see section 6.5 for further discussion). For this reason, we present a characterization of integration that applies quite broadly. Thus, success on this simple example both validates the approach, and sets the stage for the more complex models we subsequently consider (see section 8.3).

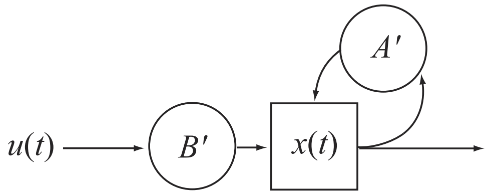


Figure 8.6
Higher-level block diagram for the neural integrator. For clarity, we use the state variable, $x(t)$, in place of the transfer function $h(t)$ to designate the population.

8.2.1 Implementation

The basic system we are interested in implementing is shown in figure 8.6. The network consists of a recurrently connected population, like that shown in figure 8.2, receiving a velocity command, $u(t)$. It has long been known that this kind of feedback is central to implementing an integrator in an analog circuit. It has also long been known that integration is a difficult problem to solve using analog devices because of the high demands on precision. So, despite the apparent simplicity of the integrator, a close inspection of this circuit reveals a number of interesting properties of neural systems modeled in this way. As mentioned, these properties are relevant for modeling many more complex, and interesting circuits.

We can express the dynamics of an integrator in standard control theoretic form as

$$\dot{x} = Ax(t) + Bu(t) \tag{8.13}$$

$$x(s) = \frac{1}{s}\left[Ax(s) + Bu(s)\right], \tag{8.14}$$

where $A = 0$ and $B = 1$. Previously, we derived equations (8.8) and (8.9), that show the relation between standard control theoretic description and one which is amenable to neural dynamics. Using these equations, we find the equivalent neural control matrices in figure 8.6, A' and B', to be

$$\begin{aligned} B' &= \tau \\ A' &= 1, \end{aligned}$$

where τ is the time constant of the PSC of the population representing $x(t)$.

In order to use this high-level description of the dynamics in a model, we need to integrate it with the neural representation we have defined for this system. This is a specific example of the general derivation provided in section 8.1.3. First, consider the time domain description of the system in figure 8.6:

$$x(t) = h(t) * [A'x(t) + B'u(t)]\,. \tag{8.15}$$

Writing this convolution explicitly gives

$$x(t) = \int_{-\infty}^{t} h(t-t')\,[A'x(t') + B'u(t')]\,dt'. \tag{8.16}$$

Substituting the activity population representation into (8.16) for $x(t')$ gives

$$x(t) = \int_{-\infty}^{t} h(t-t')\left[A'\sum_i a_i(t')\phi_i^x + B'u(t')\right]dt', \tag{8.17}$$

which means that the spiking population representation is

$$x(t) = \int_{-\infty}^{t} h(t-t')\left[A'\sum_{i,n} h(t'-t_{in})\phi_i^x + B'u(t')\right]dt'. \tag{8.18}$$

So, depending on our preferred degree of simulation realism, we can use any of equations (8.16), (8.17), or (8.18) as expressions for the dynamics of $x(t)$. The first equation models the dynamics of the high-level variable, the second models the dynamics over rate neurons and the third models the dynamics using spiking neurons. Let us consider generating a spiking network, but using (8.17) as it makes the derivation simpler. As well, we demonstrate how to mixing levels in one model by leaving the input $u(t)$ as a high-level variable.

We proceed as follows:

$$a_j(t) = G_j\left[\alpha_j\left\langle x(t)\tilde{\phi}_j\right\rangle + J_j^{bias}\right].$$

Substituting (8.17) for $x(t)$, and leaving the higher-level representation of the input, $u(t)$, gives

$$a_j(t) = G_j\left[\alpha_j\left\langle\int_{-\infty}^{t} h(t-t')\left[A'\sum_i a_i(t')\phi_i^x + B'u(t')\right]dt'\tilde{\phi}_j\right\rangle + J_j^{bias}\right].$$

Using the convolution operator, we have

$$a_j(t) \quad = \quad G_j\left[\alpha_j\left\langle h(t) * \tilde{\phi}_j\left[A'\sum_i a_i(t)\phi_i^x + B'u(t)\right]\right\rangle + J_j^{bias}\right] \tag{8.19}$$

$$= G_j \left[h(t) * \left[\sum_i \omega_{ji} a_i(t) + B' \tilde{\phi}_j u(t) \right] + J_j^{bias} \right], \quad (8.20)$$

where $\omega_{ji} = \alpha_j A' \phi_i^x \tilde{\phi}_j$.

As shown in section 5.1, we can now substitute our expressions for the decoded spike trains (e.g., $\sum_n h(t - t_n)$) into (8.20) and let G_j be the spiking LIF nonlinearity to give the spiking implementation description. This shows precisely how various levels of the generic subsystem can be simulated (see figure 8.5). Choosing a particular kind of simulation (high-level, activities, or spikes) determines both how realistic the simulation is, and what computational demands will be imposed by the simulation.

Although in this section we are concerned with the neural integrator, the preceding procedure for combining dynamics and the various levels of description is general. Let us now turn to considering more specific issues that arise for implementing an integrator. Recall that in order to have a perfect integrator, we need $A' = 1$. While we can set A' exactly to some desired constant, we know that we cannot expect our estimate of $x(t)$ (i.e., $\hat{x}(t)$) to be perfect (even if there is no noise). Thus the product $\hat{x}(t)A'$ will not be exactly equal to $x(t)A'$. We can account for this error by supposing that, in fact, $\hat{x}(t)$ is perfect and A' is subject to error. This allows us to determine what affect changing the element that we can control, A', has on our simulation. Of course, if A' is not exactly 1, then in the standard control diagram, A is not exactly 0. So, it is useful to examine exactly what happens when A is not exactly 0.

Consider, again, equation (8.4):

$$\begin{aligned} x(s) &= \frac{1}{s}(Ax(s) + Bu(s)) \\ &= \frac{A}{s}x(s) + \frac{B}{s}u(s). \end{aligned}$$

From this equation we can see that if A is slightly greater than zero, the dynamics become unstable because some small fraction of $x(t)$ is added to the current value of $x(t)$ at every time step. As well, if A is slightly less than zero, the system becomes 'leaky' since some small fraction of $x(t)$ is subtracted from $x(t)$ at every time step. In other words, A acts like a (negative) rate constant for the system. We call this the *effective rate constant*, and its inverse the *effective time constant*, $\tau_{\mathit{eff}} = \frac{1}{r_{\mathit{eff}}}$, of the system:

$$A = -\frac{1}{\tau_{\mathit{eff}}}.$$

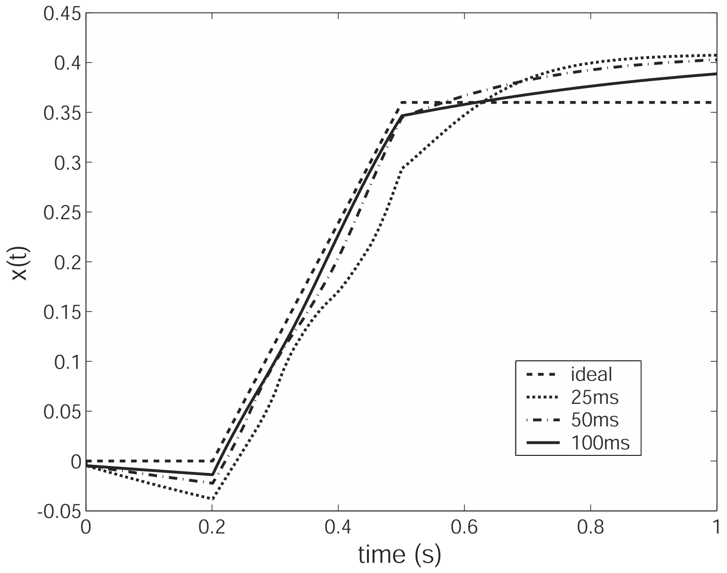


Figure 8.7
Integration of a pulse with a recurrent network of 20 LIF neurons. Plots for $\tau = 25,\ 50,$ and 100 ms compared to the ideal integrator. The average $\tau_{eff} = 0.7, 1.2, 2.3,$ and ∞ seconds respectively.

Since we know the equivalent value of A in the neural circuit, we know that

$$\begin{aligned} -\frac{1}{\tau_{eff}} &= \frac{A' - 1}{\tau} \\ \tau_{eff} &= \frac{\tau}{1 - A'}. \end{aligned}$$

This shows two things: 1) how the synaptic time constant, τ, of constituent neurons affects the integration abilities of the circuit of which they are a part; and 2) that the goodness of the $x(t)$ representation affects the integration abilities of the circuit. In the first case, we can see that a longer τ makes for a longer effective time constant (note that a perfect integrator has an infinite effective time constant). Thus neurons with longer synaptic time constants (e.g., neurons with NMDA receptors) are more suited to integration (see figure 8.7). In the second case, we can see that as A' nears unity, the effective time constant becomes infinite. Thus, setting $A' = 1$ means that all of the integration error is a result of representation error. So, the better our representation of $x(t)$ (and thus the closer $A'\hat{x}(t)$ matches $A'x(t)$), the better job we can do of implementing stable dynamics (exemplified in the extreme by perfect integration).

Unfortunately, this also means that our analysis of the dynamics of the neural integrator is not as simple. This is because, the error in $\hat{x}(t)$ varies as a function of x. We know that our neural population more accurately represents $x(t)$ for some values, and is only 'perfect' at a few points (see section 7.2). A more detailed way of showing how well a neural population represents some signal is to compare the estimated value, $\hat{x}(t)$, with the actual value, $x(t)$, for each possible value in the range of $x(t)$. This kind of comparison is shown in figure 8.8a. That graph shows precisely how the error fluctuates as a function of x.

Notably, graphs like that in figure 8.8a tell us quite a bit about the dynamics of the system. As the inset shows, we can 'trace' the progress of repeated encodings and decodings of the represented variable x. Repeated encoding and decoding, of course, is exactly what happens in a feedback network. So, following the movement of successive encodings tells us how a feedback network drifts over time. As well, the magnitude of the difference between the ideal curve and our representation gives us a sense of how quickly the representation in the network drifts. Figure 8.8b displays the same error information, but in more detail. However, tracing dynamics on this graph is less intuitive because the axes are scaled differently. Nevertheless, we know that whenever the error is positive, the network drifts in a positive direction at a speed proportional to the error (and similarly in the negative direction). Thus, every other zero-crossing on this graph (as on that in figure 8.8a), represents a *dynamically stable point*. This is because positive error to the left of the point and negative error to the right of the point both cause the system to drift towards that point.

The identification of these dynamically stable points suggests a more appropriate measure of the system's overall stability. In particular, it is useful to identify the system's *drift velocity*, $\frac{\Delta E}{\tau}$.[4] In fact, figure 8.8b gives a good sense of what the drift velocity looks like over the range of x, since the drift velocity is essentially the derivative of this curve.

We can use this measure to determine how various network properties and neuron properties affect it. We already know from figure 8.7 that as τ increases, drift velocity decreases (note that *slower* drifts are better for an integrator). There are two other important factors that affect drift velocity. The first should be quite obvious: the number of neurons matters a great deal. Since the representation of $x(t)$ improves as we increase the number of neurons (i.e., the error decreases), we expect repeated encoding and decoding to disturb the representation in a larger population much less than in a smaller population. This is shown in figure 8.9a. The second factor, which is perhaps less obvious, is the RC time constant, τ^{RC}, of the neurons. As shown in figure 8.9b, longer RC time constants make for

4 Note that we call this measure a *velocity* because it is very much like $\frac{dE}{dt}$, since τ determines the effective time step for the system. The drift velocity measures much the same property as τ_{eff} but relies on representational error rather than the value of A'. Of course, it is the product of A' and the error that determine the dynamics.

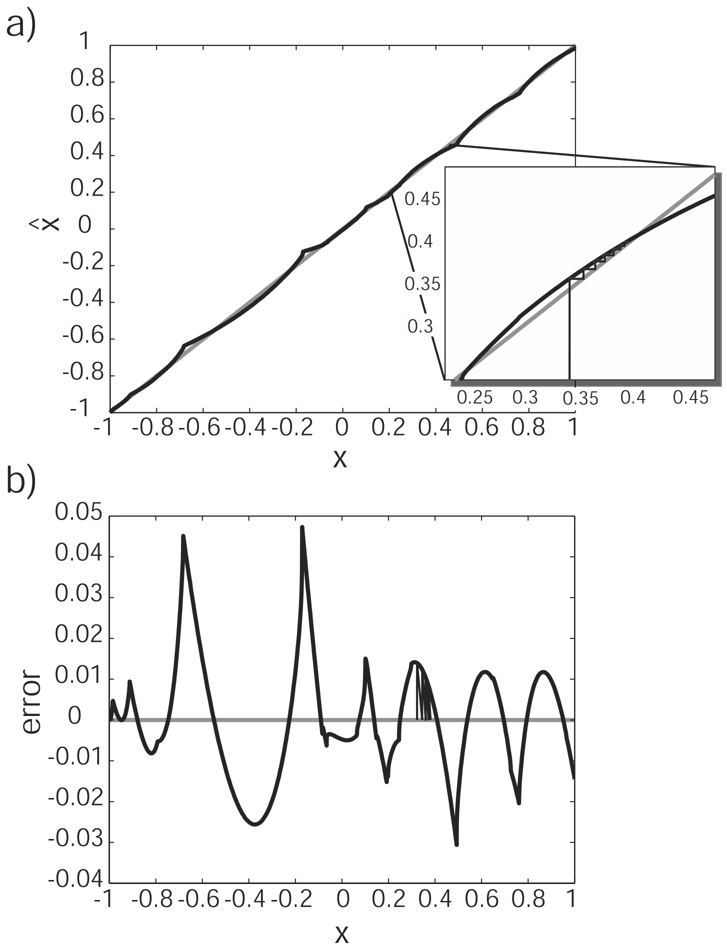


Figure 8.8
Representational linearity of the network used in figure 8.7. a) The linearity over the range of x. The grey line indicates perfect linearity. The expanded view shows the dynamics of the example in figure 8.7. Each 90° angle indicates the behavior after one additional time step. b) The difference between the estimate and actual value of x at every point, again the dynamics are shown over the same range.

better integrators. Notably, this is not because a longer RC time constant gives the neuron a longer 'memory' as might first be supposed (i.e., it does not act like τ). Rather, longer RC time constants result in *more linear* response functions, especially near the neuron's threshold (see, e.g., figure 4.3). In other words, the notoriously steep onset of the LIF neuron becomes less steep as τ^{RC} increases. This has the effect of reducing the error when

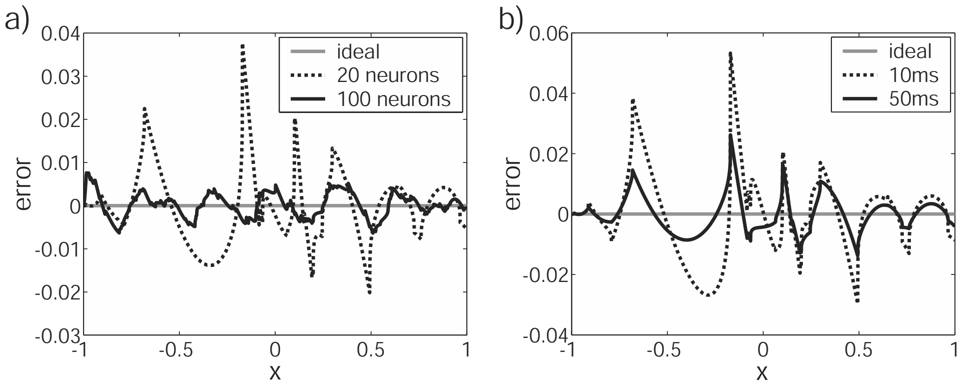


Figure 8.9
Effects of a) population size and b) RC time constant on integrator performance. In all networks, $\tau = 30$ ms. In a) $\tau^{RC} = 20$ ms and average drift velocities are 3.84 for $N = 20$ and 1.82 for $N = 100$. In b) $N = 20$ and average drift velocities are 5.7 for $\tau^{RC} = 10$ ms and 2.7 for $\tau^{RC} = 50$ ms.

fitting the resulting response functions to a straight line (which is precisely what we are doing when we find the decoders). Both of these examples show that reducing error, be it by increasing the number of neurons, or changing τ^{RC}, always improves the integrating ability of the circuit. It is interesting to note that the response functions of neurons found in neural integrators of the goldfish are extremely linear (Seung et al. 2000). We suspect the reason for this is that linear neuron responses greatly reduce representational error.

We now have a good sense of how to control the performance of the basic integrator. However, these ways of controlling network performance are based on static properties of the network. But, we can also use this simple example to introduce dynamic control. In the case of the integrator, the most natural parameter to control dynamically is A' (in equation (8.15)) because it directly determines the dynamic characteristics of the network.

In order to use a population of neurons to control this parameter, we can define a representation for it and then include that representation in equations like (8.19) above. Let

$$A'(t) = \sum_l b_l(t)\phi_l^{A'},$$

and substitute this into (8.19) to give

$$a_j(t) = G_j\left[\alpha_j\left(h(t) * \tilde{\phi}_j\left[\sum_l b_l(t)\phi_l^{A'}\sum_i a_i(t)\phi_i^x + B'u(t)\right]\right) + J_j^{bias}\right] \quad (8.21)$$

We know that to perform a multiplication between the recurrent signal, $x(t)$, and the control signal, $A'(t)$, we can introduce a new population whose dimensionality is the sum of those being multiplied (see section 6.3.1). So let

$$\mathbf{p}(t) = \sum_m c_m(t)\boldsymbol{\phi}^{\mathbf{p}}_{\boldsymbol{m}}$$

be such a population. So, we can write (8.21) as

$$a_j(t) = G_j\left[\alpha_j\left(h(t) * \tilde{\phi}_j\left[\sum_m c_m(t)\boldsymbol{\phi}^{\mathbf{p}}_{\boldsymbol{m}} + B'u(t)\right]\right) + J_j^{bias}\right], \tag{8.22}$$

where

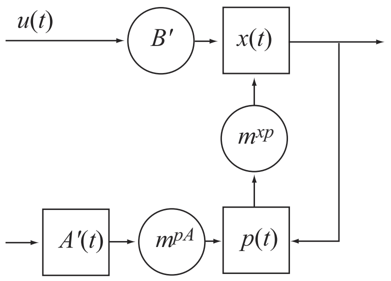
$$\begin{aligned} c_m(t) &= G_m\left[\alpha_m\left(x(t)\tilde{\phi}_m^{c_1} + A'(t)\tilde{\phi}_m^{c_2}\right) + J_m^{bias}\right] & (8.23)\\ &= G_m\left[\alpha_m\left(\sum_i a_i(t)\phi_i^x\tilde{\phi}_m^{c_1} + \sum_l b_l(t)\phi_l^{A'}\tilde{\phi}_m^{c_2}\right) + J_m^{bias}\right]. & (8.24)\end{aligned}$$

Figure 8.10a depicts the system described by these equations. Note that because $A'(t)$ is a function of time, equations (8.22) and (8.24) describe a linear *time-varying* system. Or, if we take $A'(t)$ to be an input to the system (rather than a time-varying dynamics matrix), then these equations describe a nonlinear system. This example thus shows how we can use this framework to implement these broader classes of control systems. Figure 8.10b shows the effects of changing $A'(t)$ on the neural integrator. Notice that when $A'(t)$ is reduced from 1 to .5, the circuit begins to act like a low-pass filter, rather than an integrator.

Although simple, this example is important for showing one important way to control the dynamics of a neural system. We can, in general, manipulate the parameters in the relevant $\mathbf{M}^{\alpha\beta}$ matrix using a control signal that is encoded by another population. For instance, we have also constructed a circuit in which we control the input matrix, B'. This is another means of constructing a time-varying or nonlinear system. In this case, changing the value of B' simply scales the input signal, $u(t)$. Although controlling elements of any $\mathbf{M}^{\alpha\beta}$ generally introduces nonlinearities into the model, as in this example, it often provides an intuitive way of controlling the dynamics of a given system (see section 8.5 for another example).

8.2.2 Discussion

As mentioned in the introduction, both our analyses and previous ones have shown how to construct networks that perform reasonably good integration using spiking neurons (see, e.g., Seung and Sompolinsky 1993). These analyses have helped determine how to impose *network properties* (i.e., choose synaptic weights, and numbers of neurons) that result in the ability to integrate an input signal. However, the fact that this model is part of a general a)



b)

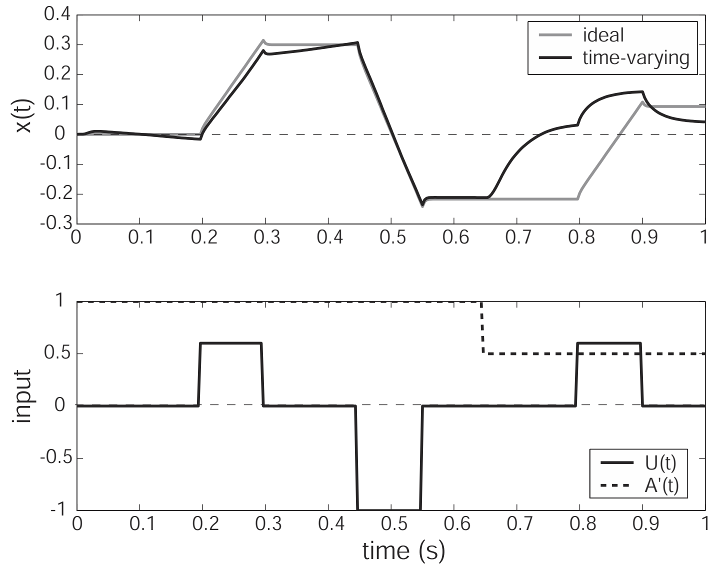


Figure 8.10
Time-varying neural integrator. In a) we have drawn the equivalent control circuit. Note that we need to multiply the time constant, τ_I, by $u(t)$ to determine the appropriate input to the integrator. This nonlinear operation necessitates an extra population, $p(t)$, as in the otolith example. All of the $m^{\alpha\beta}$ values are 1. In b) we show the output of the integrator, $x(t)$, the integration time constant, $\tau_I(t)$, and the input signal, $u(t)$. For this simulation, $N = 1000$, $\tau_{syn} = 10$ms, and $\tau_{RC} = 20$ms in all three populations.

framework has additional benefits. For instance, in showing how *single cell properties*, such as the synaptic time constant, τ, and RC time constant, τ^{RC}, affect integration as well, we have integrated the network and cellular levels of analysis. This is important because it is clear that both single cell and network properties matter to overall system dynamics. As well, having constructed the model as part of a general framework makes it

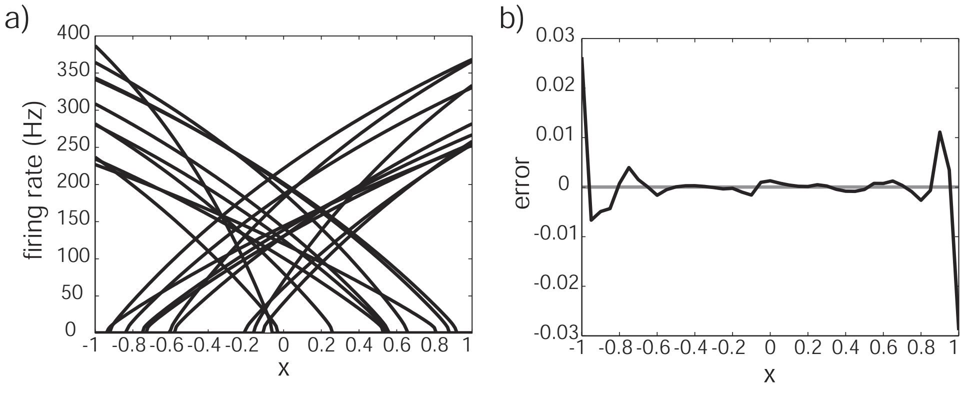


Figure 8.11
Neural integrator performance with thresholds distributed as in the real integrator. a) Distribution of a small network with most neurons active near zero. b) The error plot showing less error and therefore better dynamics near zero (compare to figure 8.8 for an even distribution).

easy to generalize such examples to examine more complex control systems; e.g., a time-varying system. As a consequence, we can apply what we learn when modeling systems like the eye-position integrator to more complex models that have similar connectivity (e.g., the vestibular model, see section 6.5), or related dynamics (e.g., the lamprey model, see section 8.5).

More specifically, we can gain insights about the eye-position integrator itself from examining this model. For instance, the example circuits we presented earlier assume populations of neurons with evenly distributed neuron thresholds. However, as mentioned in section 2.3.1, the eye-position integrator has thresholds distributed such that they are concentrated on the contralateral side. From what we have said about the effect of representation on integration, this suggests that, for the same number of neurons, there is a preferential encoding of eye position when the eyes are straight ahead. This is because more neurons are active at this position (zero), giving a better estimate of values near zero (see figure 8.11b). This means that the dynamics should be more stable near zero than for eye positions near either extreme, as is found to be the case in the real system (Hess et al. 1985).

The fact that the thresholds of integrator neurons, in both the goldfish and human systems, tend to be concentrated contralaterally raises a second point; the importance of the opponency (i.e., on/off) configuration. Suppose, as some models do (Seung et al. 2000), that the integrator consists only of tuning curves with positive slopes. In this case, for values near the extreme left of the range, there are very few neurons firing at very low firing rates. Given the large fluctuations in dendritic currents due to neural spiking, which are especially

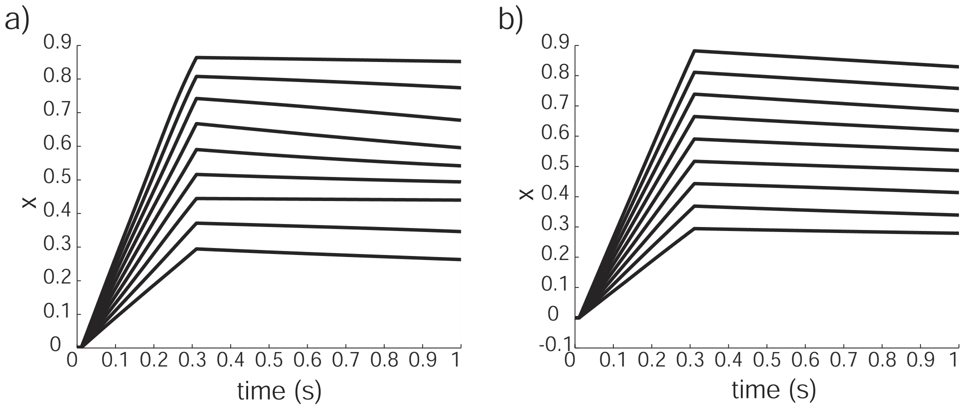


Figure 8.12
Networks approximating the neural integrator in a) humans and b) goldfish. The figures show integrator behavior at a variety of eye positions. In this model, we have tuned the gain slightly lower than unity in order to have centripetal drift at all eye positions. The average time constants are a) 20.1 s (N=1000) and b) 10.3 s (N=30).

prominent at low spike rates, we would expect any estimate of the encoded eye position to be extremely poor. Consequently, it would be very difficult to have stable dynamics in this neighborhood given these fluctuations. However, if we add a second population of neurons with negatively sloped tuning curves, then most of them will be firing at high rates at these same eye positions. These neurons will thus counteract the difficulties involved in encoding eye position with only a few active neurons.

Finally, the general integrator analysis we have done in the previous section deviates from what is seen in the biological eye-position integrators. That is, the fixed points that we (and others) have seen in such models have not been directly observed in the biological system. Rather, in biological integrators, there tends to be *centripetal* drift or unilateral drift, not a combination of centripetal and non-centripetal drift. However, because we have formulated our model in the context of control theory, we know precisely how to incorporate centripetal-only drift into the model. Slightly reducing the feedback gain, A', from unity has the desired effect. As shown in figure 8.12, this slight variation results in networks that perform comparably to the biological systems. Here, simulations of the human integrator and goldfish integrator are compared. In both cases, we use a population of neurons of approximately the same size as found in the respective systems and we find that the resulting time constants of these networks also fall within the range found for each system (Hess et al. 1985; Seung et al. 2000).[5]

5 Note that for the human integrator we use the LIF model, whereas for the goldfish model we use a rectified linear model which more closely mirrors the tuning properties of neurons found in this system (Seung et al. 2000).

8.3 An example: Working memory

8.3.1 Introduction

As for the neural integrator example, we have already presented the information relevant to specifying the system description and design specification for a model of lateral intraparietal (LIP) working memory (see 3.4). So, in this section we are concerned solely with the dynamics of LIP working memory. We have chosen to present this model because, although the transformation involved is familiar (namely, integration), this example includes the transformation of *function* representations.

Recall that LIP neurons have been shown to have sustained activity during a delay period between stimulus presentation and a 'go' signal. This sustained activity is typically modeled as a sustained Gaussian 'bump' of neural activity across a population (Zhang 1996; Laing and Chow 2001; Hansel and Sompolinsky 1998; Camperi and Wang 1998; Kishimoto and Amari 1979). This behavior is analogous to the neural integrator: where the integrator 'remembers' scalar values over some desired range, area LIP 'remembers' some desired set of Gaussian-like functions. However, as discussed in chapter 3, things are more complicated in LIP for two reasons: 1) parietal systems can store multiple targets at the same time (Platt and Glimcher 1997); and 2) there is evidence for significant effects of non-spatial parameters on the strength (i.e., height) of such representations (Sereno and Maunsell 1998; Romo et al. 1999).

8.3.2 Implementation

Because there are a number of distinct issues that arise during implementation, we have divided this section into two sub-sections. In the first, we show how to describe the dynamics of this system in terms of the vector of coefficients of the function space (i.e., using a vector representation). Once we have done this, it is straightforward to apply our previous discussion of the relation between vector and function representations. In the second section, we present simulation results based on the model we have derived.

8.3.2.1 Dynamics of the vector representation

Recall from section 3.3 that we can define an equivalent vector representation for this system with respect to the orthonormal basis $\Phi_m(\nu)$:

$$x(\nu; \mathbf{A}) = \sum_m A_m \Phi_m(\nu). \tag{8.25}$$

Thus, we can write the encoding and decoding rules for the amplitudes, $\mathbf{A}$, as

$$a_i(\mathbf{A}) = G_i\left[\alpha_i \left\langle \mathbf{A}\tilde{\mathbf{q}}_i\right\rangle_m + J_i^{bias}\right] \tag{8.26}$$

$$\hat{\mathbf{A}} = \sum_i a_i(\mathbf{A})\mathbf{q}_i. \tag{8.27}$$

In order to convert these $\mathbf{A}$ vectors back into their equivalent functions, we can decode them using $\Phi_m(\nu)$ as in (8.25).

As mentioned in section 8.3.1, the dynamics of this system should be very familiar. In fact, we can rely on the control diagram in figure 8.6 for the neural integrator to define the dynamics for this system as well. Thus, the vector version of (8.15) defines the dynamics, and $\mathbf{A}'$ and $\mathbf{B}'$ are naturally generalized to be $\mathbf{I}$ and $\tau\mathbf{I}$ respectively. We can now proceed very much as we did for the integrator. For expository purposes, however, we perform the analysis here directly for spikes. Thus, we begin by writing our spike-based encoding of the coefficients, $\mathbf{A}(t)$, as

$$\hat{\mathbf{A}}(t) = \sum_{i,n} h(t - t_{in})\mathbf{q}_i$$

(the decoding stays the same). Recall from (8.18) that we can incorporate the higher-level dynamics into a basic-level representation. Thus, for vector dynamics we write:

$$\mathbf{A}(t) = \int_{-\infty}^{t} h(t - t')\left[\mathbf{A}'\sum_{i,n}\delta(t' - t_{in})\mathbf{q}_i + \mathbf{B}'\mathbf{u}(t')\right]dt'. \tag{8.28}$$

Substituting this expression into (8.26) gives

$$\begin{aligned} a_i(\mathbf{A}(t)) &= G_i\left[\alpha_i\left\langle h(t) * \tilde{\mathbf{q}}_i\left[\mathbf{A}'\sum_{j,n}\delta(t - t_{jn})\mathbf{q}_j + \mathbf{B}'\mathbf{u}(t)\right]\right\rangle_m + J_i^{bias}\right] \\ &= G_i\left[\sum_{j,n}\omega_{ij}h(t - t_{jn}) + \alpha_i h(t) * \mathbf{B}'\mathbf{u}(t) + J_i^{bias}\right], \end{aligned}$$

where $\omega_{ij} = \alpha_i\left\langle \mathbf{A}'\tilde{\mathbf{q}}_i\mathbf{q}_j^x\right\rangle_m$.

8.3.2.2 Simulation results

Figures 8.13 and 8.14 show simulation results from this model. All results are from the same network of 1000 spiking LIF neurons with synaptic time constants of 5 ms. The tuning curves of this network are like those found in LIP, as shown previously in figure

3.2. As expected, the network is able to stably encode multiple Gaussian bumps of various heights.

Figure 8.13 shows the basic case of encoding a single bump at some location. This behavior reproduces that of past models. Figure 8.13b displays the time course of the estimates of the amplitudes, $\mathbf{A}(t)$, that encode the function, $x(\nu; \mathbf{A}(t))$, which is shown in 8.13a. The dotted lines in figure 8.13b depict the ideal behavior. The spiking model is much noisier than this ideal, but has fixed points that very closely approximate those of the ideal. As can be seen from 8.13a, the approximation is so close that the function $x(\nu; t)$ remains very stable during the course of the simulation. Figure 8.13c shows spike rasters for 200 of the 1000 neurons used in the model. As expected, there is a general increase in the firing rate of neurons nearer the middle of the encoded bump.

Figure 8.14 shows how this model extends the results of past models by demonstrating that the network can also encode and 'remember' multiple bumps of different heights. This model extends past results in two distinct ways: 1) it encodes bumps at the same position but of different heights; and 2) it encodes multiple bumps (of various combinations of heights). Figure 8.14 combines these results by showing the performance of the model encoding two bumps of different heights, where the leftmost bump is at the same position as (but a different height from) that shown in figure 8.13a. Again, figure 8.13b shows that the encoded amplitudes, $\mathbf{A}$, are quite stable during the run.

Notably, increasing the synaptic time constant of the model will lengthen the time constant of the network (as discussed in detail in the case of the neural integrator; see section 8.2.1). Because these results are from a network with a 5 ms, AMPA-like synaptic time constant, and some receptors (e.g., NMDA) have time constants about 20 times longer, the stability shown here can be significantly improved by changing the relevant cellular properties .[6]

We believe, given these results, that this model contributes to our understanding of how working memory systems, like LIP can produce the behavior they do. But, there is an important sense in which the problem we have been solving is different from that usually posed, and addressed by all past models. Namely, the bumps we have been discussing are not *the same bumps* that modelers usually worry about. Past models have been interested in bumps of *neural activation*. Here, we have modeled bumps as *encoded functions*. At first glance, these seem to be very different things. Neural activations can be measured directly from neural systems, but encoded functions cannot: they have to be *decoded* in order to give rise to neural activations. Nevertheless, this model displays definite 'bumps' of neural activity as well (see figure 8.15). Both a single bump of activation for the single encoded bump and a double bump of activation for the double encoded bumps are evident.

6 Also recall that including more neurons would lengthen the network time constant.

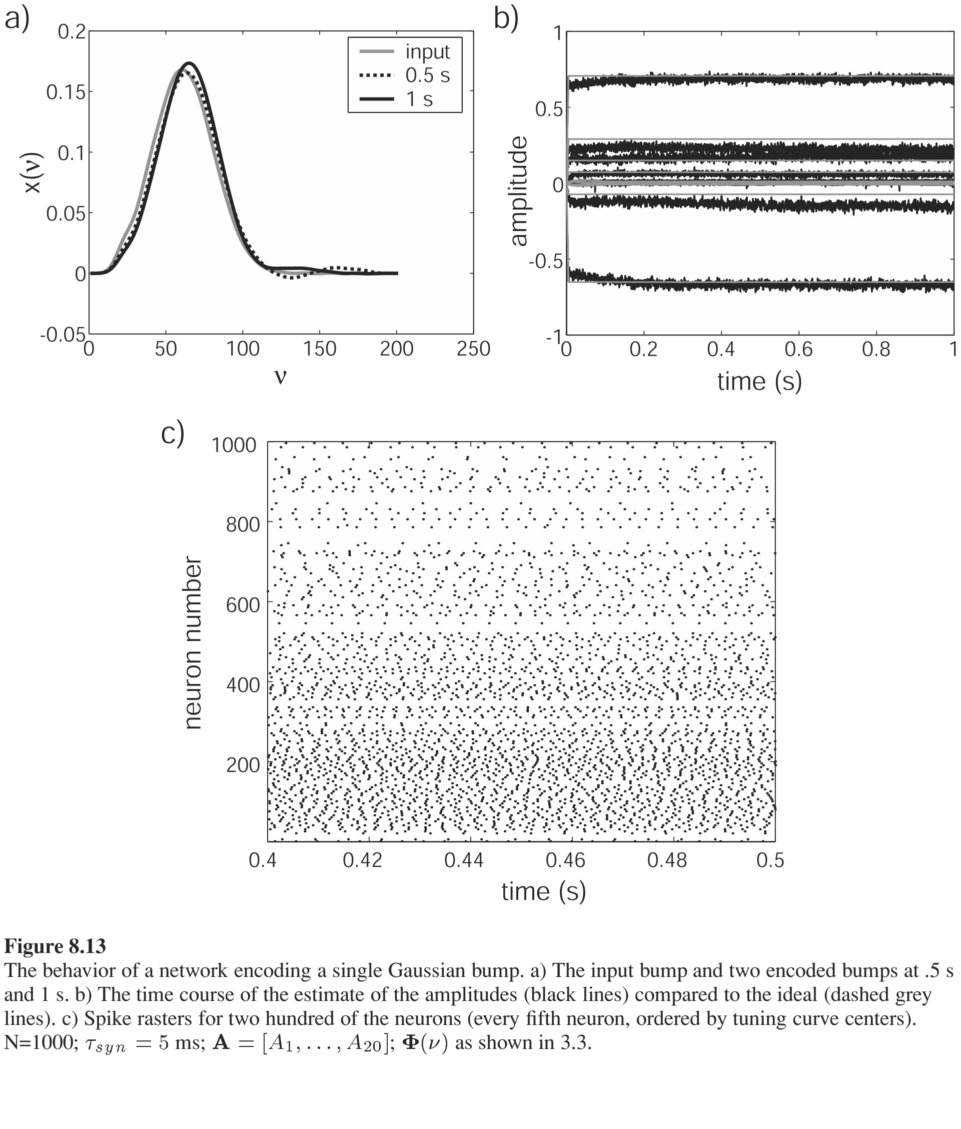


Figure 8.13
The behavior of a network encoding a single Gaussian bump. a) The input bump and two encoded bumps at .5 s and 1 s. b) The time course of the estimate of the amplitudes (black lines) compared to the ideal (dashed grey lines). c) Spike rasters for two hundred of the neurons (every fifth neuron, ordered by tuning curve centers). N=1000; $\tau_{syn} = 5$ ms; $\mathbf{A} = [A_1, \ldots, A_{20}]$; $\mathbf{\Phi}(\nu)$ as shown in 3.3.

The fact that these bumps of activation are approximate, rather than perfectly smooth, is consistent with the available evidence. And, in fact, it is possible that this less perfect kind of bump can help account for some of the variability found in the neural data. In other words, perfectly smooth activity bumps have not been observed in neural systems, and that fact might be *explained* by this kind of model.

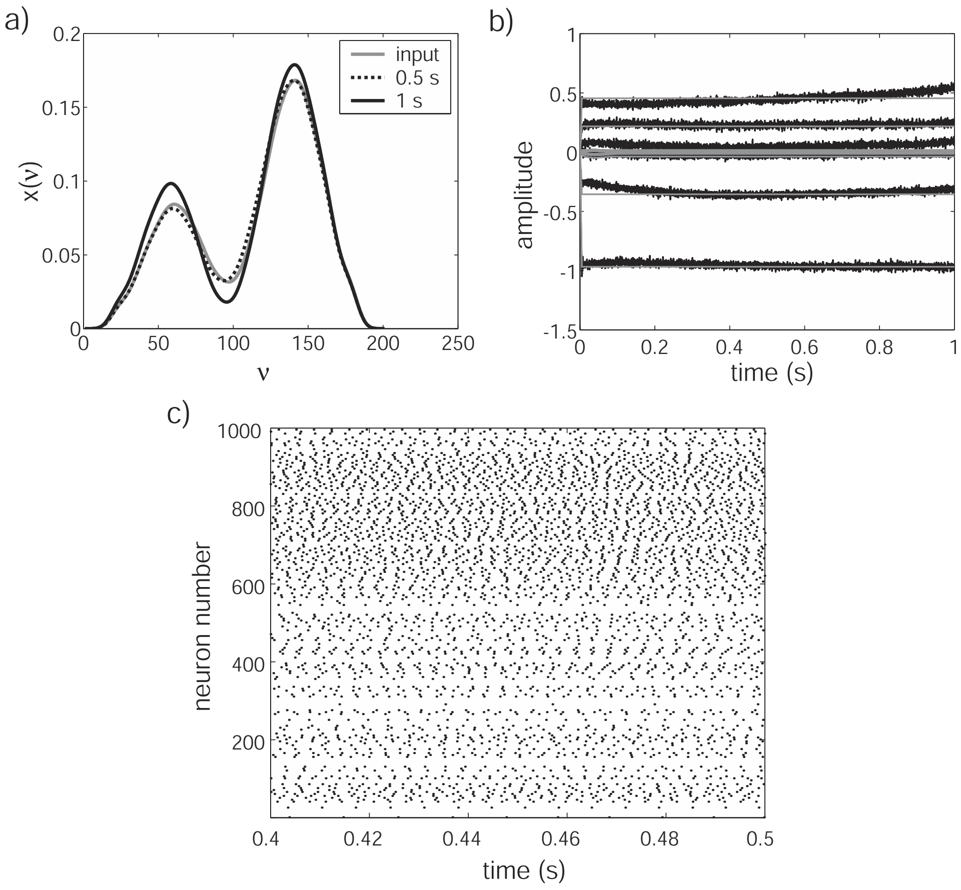


Figure 8.14
The behavior of a network encoding a double Gaussian bump. All figures and network properties are the same as those as in figure 8.13. Note that the leftmost bump is in the same position, but half the height of that in figure 8.13a.

8.3.3 Discussion

Our main purpose in presenting this example was to show how transformations of functional representations can be captured by this framework. However, it also incorporates novel contributions to current efforts directed at modeling areas like LIP (e.g., Laing and Chow 2001; Camperi and Wang 1998; see Ermentrout 1998b for a review). In particular, this network describes additional experimentally observed behavior (i.e., storing multiple

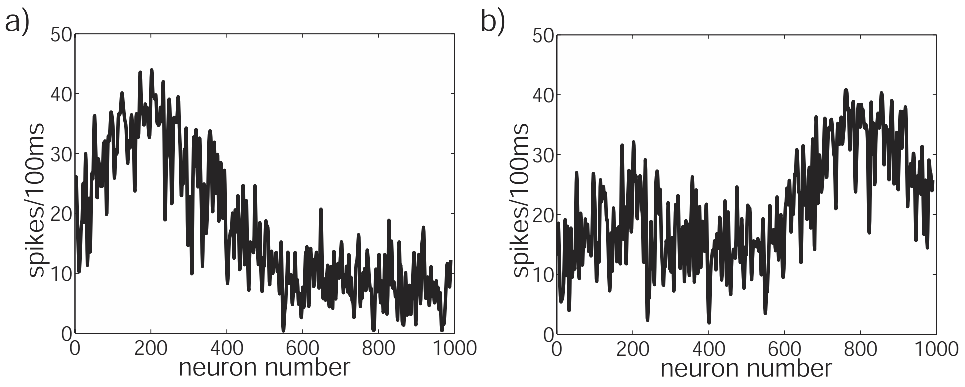


Figure 8.15
Stationary bumps of neural activity. These figures show raw spike counts over the 100 ms window half way through the simulation. They have also been slightly smoothed by a running spatial average. a) Activity bump for the simulation in figure 8.13. b) Activity bumps for the simulation in figure 8.14. In both cases, bumps of neural activity corresponding to the decoded bumps in figures 8.13a and 8.14a are clearly visible.

spatial locations at the same time, and storing parameterized bumps). As well, we have used a spiking model, rather than a less realistic (though more common) rate model (see e.g., Hansel and Sompolinsky 1998; Kishimoto and Amari 1979). Finally, although we have not presented an example here, we have used this approach to generate a similar model that stably encodes two-dimensional bumps (Eliasmith and Anderson 2001).

Interestingly, such models lead to an experimental prediction. Currently, memory experiments generally record from neurons with stimuli at the center of their receptive fields. In these experiments, researchers always observe a *decrease* in firing rate during the memory delay relative to the initial encoding of the stimulus. Notably, the standard stimuli in these kinds of experiments is a small point of light. Such stimuli are significantly narrower than tuning curves in the neurons thought to be encoding these bumps. Similarly, given an input that is significantly narrower than the tuning curves of neurons in this model, the firing rate of neurons whose receptive fields are centered on the stimuli show a decrease in firing rate (see figure 8.16). However, the networks also include neurons whose firing rates *increase* as well. Specifically, these are neurons for which the stimuli is on the *edge* of its receptive field. We suspect, then, that there are neurons whose firing rates increase during the memory delay period.

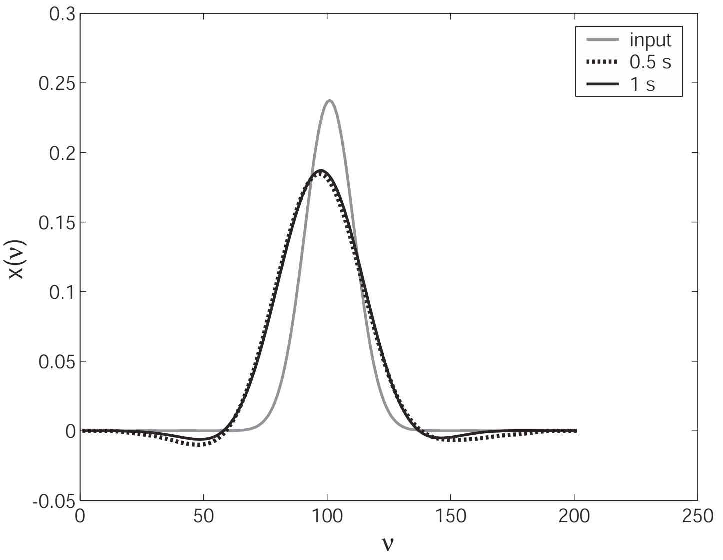


Figure 8.16
Narrow bump evolution for the network used previously (see figures 8.13 and 8.14). We can see from this figure that, for a narrow stimulus, neurons whose receptive field center is near the center of the bump have a decrease in firing rate. However, it is also clear that neurons whose receptive field center is near the edge of the bump have an increase in firing rate.

8.4 Attractor networks

8.4.1 Introduction

It should be quite obvious that the preceding bump network and the neural integrator are two of a kind. Both are designed to dynamically 'hold' or 'maintain' a single point in some abstract space, over an extended period of time. In fact, it is noticing this high-level similarity, and being able to characterize the behavior of neurobiological systems at that high-level, that paves the way for constructing a bump network whose dynamics and representations give rise to a novel model. However, we think that these two networks only scratch the surface. That is, we think they are only two examples from a very broad class of networks that is fundamental to understanding many neurobiological systems. In this section, we describe that class of networks and discuss how they can be understood using this framework. Understanding neurobiological systems in terms of such networks is not a novel approach in itself (see, e.g., Amit 1989; Goodridge and Touretzky 2000; Seung 1996), but a general method for integrating neurobiological plausibility and the complex dynamics exhibited by such networks is.

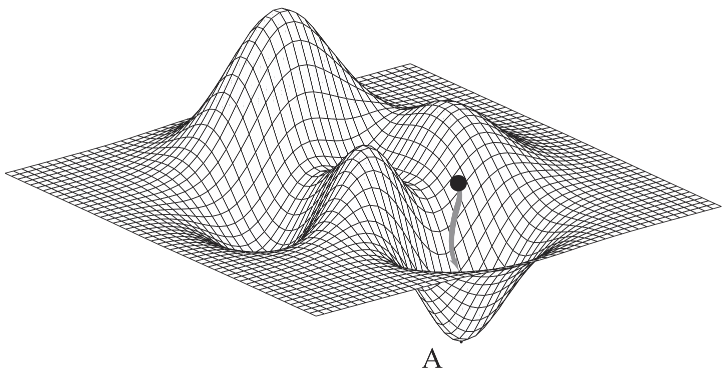


Figure 8.17
An arbitrary state space shoing a point attractor at A. The black dot represents the current state of the system and the grey arrow indicates is future trajectory towards A.

In the terminology of dynamic systems theory, this class of networks is called *attractor networks*. This is because the state space of the network (e.g., possible eye positions in the neural integrator, or possibly encoded functions in the bump network) can be characterized as containing a series of dynamically ‘attractive’ points. In the simplest case, a point, A, is said to be *attractive* when a system at any point in the neighborhood of A approaches A over time. The standard analogy is to imagine that the system is a ball on a hilly surface (the state space). When there is a large dent in the surface, the ball will tend towards the bottom of that dent, or basin. The point at the bottom of the basin is thus a *point attractor* (see figure 8.17).

There are a number of natural generalizations of a single point attractor. First, there can be more than one attractor in the same state space. If we arrange these in particular ways, we can get other interesting kinds of attractors. For example, if we arrange the points in a line in the state space, we have a *line attractor* in which each point along the line is a stable point. Thus, the system stays at whichever point it first reaches that lies on the line (see figure 8.18a). This, in fact, describes the behavior of the neural integrator; once the integrator reaches some eye position, for any possible eye position, it stays there (Seung 1996). As we have been at pains to point out, however, the line attractor in the neural integrator model is not ideal, but rather approximate;[7] that is why it drifts over time. This means that the model has many, though not infinite, points aligned along a line in neural state space, with minimal curvature between them (see figure 8.18b).

7 Note that, in the face of noise, a true line attractor and a sufficiently good approximation are indistinguishable.

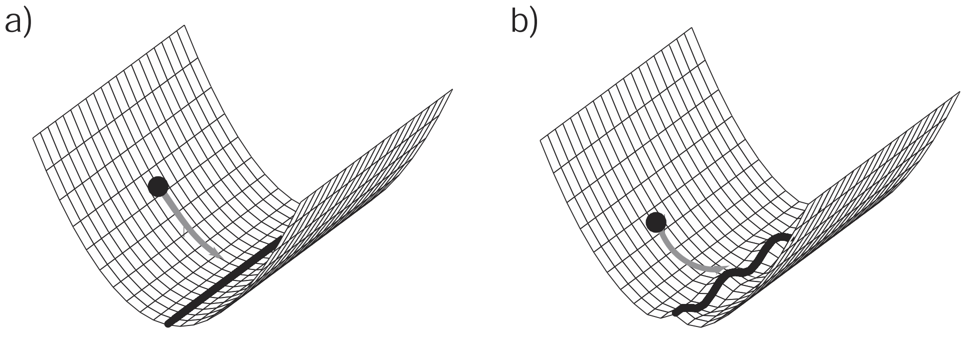


Figure 8.18
a) A line attractor. b) An approximate line attractor. Note that the degree of curvature is proportional to how long it will take the system to traverse the graph. By analogy, a ball rolls more slowly down a shallow incline than a steep one.

To construct another kind of attractor network, we can arrange the fixed points in a circle, giving what is known as a *ring attractor*. A ring attractor is just like a line attractor whose end points have been joined. This kind of attractor is a natural way to describe systems that can encode and hold positions over a repeating axis (e.g., directional heading in hippocampus; see Zhang 1996). Just as with a line attractor, each point in the ring is a stable point that the system does not move from unless perturbed (see figure 8.19).

In general, the terminology and formalism of attractor networks is very useful for both qualitative and quantitative descriptions of systems that evolve over time. More specifically, there are good reasons to adopt this terminology for *neural* systems. This is largely because ‘persistent activity’ in neurons has long been observed in the brain. Persistent activity is sustained firing that is generally started by some external stimulation, but then continues after the stimulation itself ceases. Hebb (1949) first proposed that persistent activity could be identified with short-term memory. However, it was not until 40 years later that Amit (1989), following work on attractors in artificial neural networks (Hopfield 1982), suggested that such persistent neural activity could be identified with dynamical attractors of a recurrent biological network. If Amit’s suggestion is right, then attractor dynamics are ubiquitous in neural systems because 1) local recurrence is a common feature of cortical circuits (Douglas et al. 1995) and 2) persistent neural activity has been recorded from a large variety of both cortical and subcortical areas. Cortical areas with persistent activity include motor, premotor, posterior parietal, prefrontal, frontal, hippocampal, and inferotemporal cortex. Subcortical areas include the basal ganglia, midbrain, superior colliculus, and brainstem. Furthermore, persistent activity has been observed in mammals as well as various non-mammalian vertebrates. Let us briefly consider two specific examples in more detail.

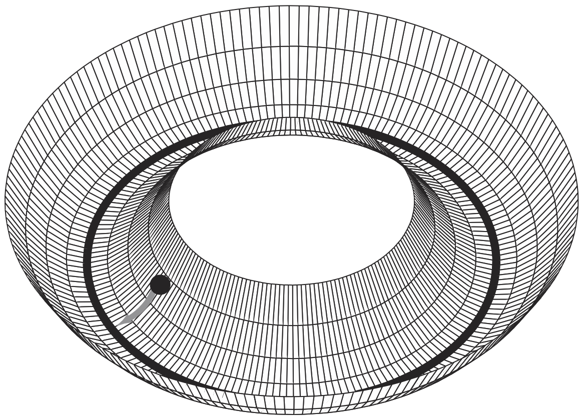


Figure 8.19
A ring attractor. This is like the line attractor, with the endpoints joined. It is a natural model for the head direction system in the hippocampus (Zhang 1996).

In the case of the neural integrator, there is excellent evidence that recurrent properties of a somewhat localized network are largely responsible for the observed persistent activity (Askay et al. 2001; Askay et al. 2000; Seung et al. 2000; Seung 1996). For example, Askay et al. (2001) have shown that persistent activity in the *in vivo* integrator is not sensitive to stimulation of individual neurons that are part of the integrator network. Thus, sustained firing seems to be mainly a property of the organization of the entire network that is not derivative of the properties of individual cells, such as plateau potentials. As well, stimulation studies have shown that perturbing eye position can be affected by stimulating areas thought to be part of the recurrent attractor network (Yokota et al. 1992), but not by stimulating downstream, oculomotor neurons (Robinson 1968).[8] Thus, the neural integrator is a good candidate for a neurobiological system that can be described as a recurrent, line attractor network: it is stable against certain kinds of perturbations; it can stably store many different eye positions; and it has this property on the basis of network-level organization.

There is also evidence that the ring attractor is a good description of neural dynamics in certain cortical and sub-cortical areas. Since the early 1980s, there has been an ongoing experimental and theoretical study of what is known as the ‘head-direction system’ in mammalian hippocampus and nearby thalamic areas (Ranck Jr. 1984; Taube et al. 1990a; Taube et al. 1990b; Rolls and O’Mara 1995; Chen et al. 1994; Touretzky and Redish 1996; Goodridge and Touretzky 2000). By far the majority of these studies have been based

8 For a good review of the relevant evidence, see Moschovakis (1997).

on navigation in the rat (although see Rolls and O'Mara 1995). In the rat, these cells act like a neural gyroscope, always indicating the current head direction, even in the dark (McNaughton et al. 1991). Many computational neuroscientists have found the affinity between the observed neural dynamics and those of the ring attractor compelling. To the best of our knowledge, all current models take part of the head direction system to have ring attractor dynamics (see Zhang 1996; Goodridge and Touretzky 2000; Touretzky and Redish 1996; Skaggs et al. 1995).

Given the compelling success and biological plausbility of attractor models of both the neural integrator and the head-direction system, it is clear that attractor dynamics are useful for characterizing *some* neurobiological systems.[9] However, by far the majority of the work on attractors in neurobiology has focused on simple line and ring attractors. In the next few sections, we argue that attractor dynamics can be used to understand a far wider variety of neurobiological systems. As well, we show how the approach we are presenting can support this kind of generalization. More specifically, we generalize attractors dynamics in two ways: 1) by generalizing representation and 2) by generalizing dynamics.

8.4.2 Generalizing representation

Both the head-direction model and horizontal neural integrator model operate on scalar representations, x. In the head-direction ring attractor, x, ranges between 0 and 360 degrees, and in the horizontal neural integrator, it ranges between about ± 50 degrees. Of course, both head direction and eye position are better described as (at least) two-dimensional vectors that account for vertical motion as well. In the neural integrator, it is well known that vertical eye position is encoded in much the same manner, though somewhat independently (Moschovakis 1997). This suggests that a reasonable model of the more complete, 2-dimensional integrator can be built out of two systems modeled after the horizontal integrator alone. However, there are instances when such higher-dimensional integrators are not simply a combination of many, *independent* one-dimensional integrators. A good example of this is the bump network in section 8.3. In this network, we represent the functions as high-dimensional vectors, where each neuron had a different, non-axial preferred direction vector in this space. So, in this case, the dimensions are not independently encoded.

These examples show how changing the dimensionality of the representations can change the kind of attractor being implemented. For instance, the 2-dimensional integrator implements a *plane* attractor, rather than a line attractor (see figure 8.20). Given the fact that we can write the dynamics of these simple sorts of attractors as $\dot{\mathbf{x}} = \mathbf{0}$, and we know how to represent $\mathbf{x}$ for any arbitrary dimension, we can straightforwardly build models

9 As a result, the Cold Spring Harbor Symposium on Persistent Neural Activity, organized by D. Tank and S. Seung, in October 2000 drew a large number of experimental and theoretical neuroscientists interested in pursuing this approach.

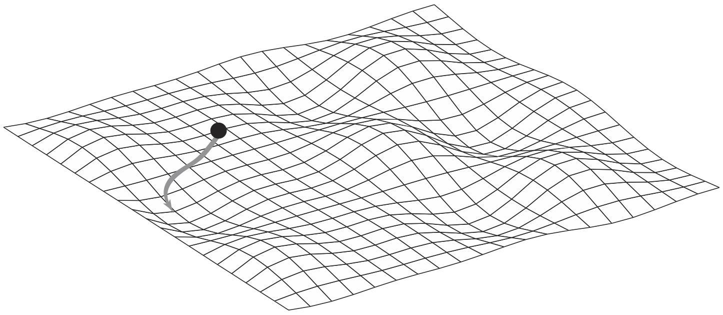


Figure 8.20
An approximate plane attractor. This is a natural model for a more comprehensive head-direction or neural integrator system.

of ‘hyperplanes’ of any dimension. This could very well be useful for building models of either motor or perceptual systems that operate on time scales much longer than those of individual neurons (as in the case of the neural integrator and the short-term memory systems). Given how common persistent activity is across neurobiological systems that operate in very different state spaces, such models could be quite common.

Moving from scalars to vectors only generalizes over the first two stages of the representational hierarchy we identified earlier (see section 3.1). Next in the hierarchy are functions. Of course, the bump model implements an attractor network using functional representations. But, as we have shown, the problem of implementing attractor networks for functions is no more difficult than for vectors. In our bump model, we used a 20-dimensional vector space in which we constructed an arbitrarily-shaped attractor. That is, we did not have a hyperplane attractor because only *some* of those 20-dimensional vectors—those that were the amplitudes for Gaussian-like functions—were made stable. So, through the techniques used in our models we have shown how to construct arbitrary attractors in arbitrarily high dimensional vector spaces (including function spaces). It is the generality of our characterization of both representations and transformations that makes this possible. The utility of this characterization is demonstrated by the ability of the resulting networks to model behaviors that have not otherwise been captured.

Furthermore, we have explored ways to help us better understand the nature of neural attractors. As noted earlier, our talk of attractor networks is shorthand for constructing neural *approximations* to these attractors. Since understanding the approximation that a representation has to what it represents (i.e., $x(t) - \hat{x}(t)$, and the effects of noise), has been central to our framework, it should not be surprising that the framework can prove useful in characterizing approximate attractors. Figure 8.8 is one way of showing, in a dynamically relevant way, how good the approximation is. In the discussion surrounding that figure, we noted that every other crossing point would be dynamically stable. Since, in section 7.4, we introduced tools to help us determine how many crossing points there would be in such a network (i.e., the 'representational capacity'), we can determine how many stable points there are in a given attractor network.

In general, the work we did in chapter 7 on the goodness of neural representation is informative about the dynamics of attractor networks. For instance, given that the central conclusion of those discussions was that heterogeneity was important to good representation, we should suspect that heterogeneity is important for stable dynamics as well. This is shown to be the case in figure 8.21. In this figure, the heterogeneous population performs better than either the population comprised of identical neurons or of shifted neurons. The population of identical neurons always has only one fixed point (zero), except in the case of four neurons, where there are fixed points at each of the extremum.[10] For this population, although the RMS error is continuously improving, the dynamic properties are consistently poor, and stay so for any number of neurons in the population. In contrast, both the heterogeneous and shifted populations continue to improve their approximation to a line attractor as the number of neurons is increased. However, in all cases the heterogeneous population performs as well or better than the shifted population because there are more degrees of freedom for the optimization to take advantage of when determining the decoding weights. Specifically, the gain of different neurons is different in the heterogeneous case, but they are all the same in the shifted case. So, the heterogeneous population tiles the input space more completely and can thus be used to construct a better approximation to a line, which results in a better approximation to a line attractor (i.e., improved drift velocity).

8.4.3 Generalizing dynamics

As previously noted, there are a wide variety of attractors. So far, we have discussed point, line, and ring attractors. However, this list is by no means exhaustive. In fact, each of these attractors has very simple dynamics, i.e., $\dot{\mathbf{x}} = 0$. We can think of this as being a very simplified version of the state equations discussed in section 8.1:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u},$$

10 This occurs because reducing the RMS error as the number of neurons increases means balancing the significant nonlinearities near the neuron thresholds with the contributions near the maximum firing rates of the contralateral neurons. When there are only a few neurons (2 or 4), this cannot be done very effectively so the nonlinearities dominate, resulting in large RMS error and poor estimates near the extremes. As the number of neurons increases, RMS goes down as a better balance is achieved. The temporary, 'extra' fixed points at the extremum are due to the progressive improvement in the estimate during the 'settling' of this balancing. They are, in effect, accidental.

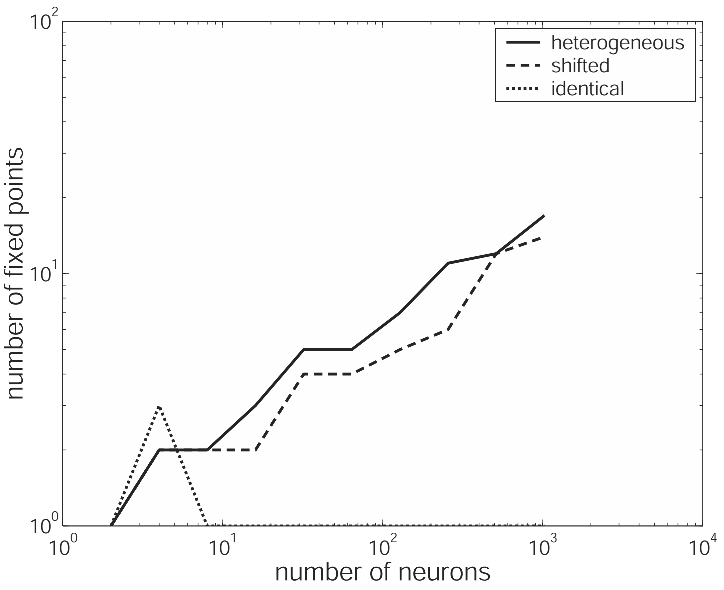


Figure 8.21
The number of stable fixed points as a function of the number of neurons for the nearly identical, shifted, and heterogeneous populations of neurons. The heterogeneous population has the most stable dynamics over a range of population sizes.

where $\mathbf{A}$ and $\mathbf{B}$ are both zero. Of course, we could allow $\mathbf{A}$ to be equal to some constant. If we did, then our system would no longer have stable dynamics, it would either increase or decrease exponentially, depending on the sign of $\mathbf{A}$. However, there are other kinds of stable dynamics that can be achieved if we pick $\mathbf{A}$ properly. For example we could choose

$$\mathbf{A} = \begin{bmatrix} 0 & \omega \\ -\omega & 0 \end{bmatrix}.$$

In this case, we will have implemented a simple harmonic oscillator at frequency ω (the amplitude of the oscillations depends on the initial conditions).

This introduces another, more complex class of attractors: *cyclic* (or *periodic*) *attractors*. Unlike the attractors we have discussed so far, the 'ball' analogy fails us in understanding cyclic attractors. This is because we would have to imagine a ball that sat in a circular trough and always went downhill, but never descended. However, this picture does get the idea across that a cyclic attractor is, rather than a single state, a sequence of states that the system tends to repeat over time (see figure 8.22). That is, regardless of starting position, the system tends to settle into a repetitive, dynamic pattern.

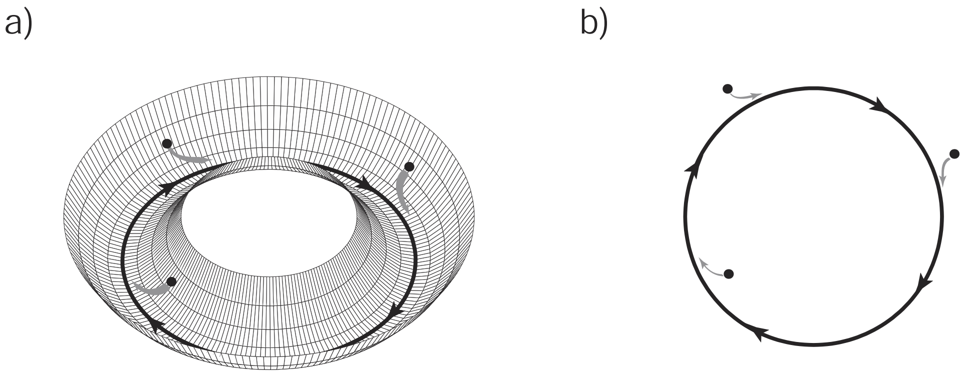


Figure 8.22
A cyclic attractor. The thick line shows a cyclic attractor and the grey lines indicate possible trajectories of the system with this attractor in the state space. Figure a) includes the state space topology as with previous attractors. Figure b) is a top down view, which is a more standard way to diagram this kind of dynamics.

In fact, the simple harmonic oscillator does not describe a single cyclic attractor. Rather, it describes a plane of infinitely many concentric cyclic attractors. So, just as the line attractor is a set of infinitely many point attractors, so the oscillator attractor is a set of infinitely many cyclic attractors. As well, just as we can control the stable point of the line attractor by adjusting $u(t)$, so we can control the cycle (i.e., amplitude) at which the oscillator attractor operates by adjusting $\mathbf{u}(t)$. One notable difference between the oscillator and the line attractor is that the oscillator also includes the variable ω which controls the speed with which the system traverses the attractor. There is no analogous variable (since the attractor pattern is static) in the line or ring attractors.

So, what use is such a network for understanding neurobiology? Just as the line attractor maps on to the stable behavior of eye positions, so the oscillator attractor maps naturally on to many repetitive behaviors, like swimming, walking, and chewing. These kinds of motions have the same qualitative dynamics as cyclic attractors. In section 8.5, we show that cyclic attractors provide a good way of understanding swimming behavior in the lamprey eel.

8.4.4 Discussion

In the preceding discussion, we gave examples of particular neurobiological systems that can be modeled as attractor networks. However, this kind of mapping of one kind of attractor dynamics onto ‘whole’, independent systems is purely expository. For instance, the otolith network discussed in section 6.5 clearly incorporates an integrator, but is more than just an integrator. So, attractor networks can often be considered just one part of a larger neurobiological system performing a more complex function. Still, analyzing these networks on their own helps us to both understand and model more complex systems.

We suspect that attractor networks are, in fact, very likely to be part of more complex systems because they have a number of important computational properties. Given our past discussion, the *stability* of these networks is their most apparent property. Dynamic stability is important for a number of reasons. First, it provides a kind of short-term memory that can aid in the processing of complex problems. Second, it allows the system to react to environmental changes on multiple time scales. In particular, stability permits systems to act on longer time scales than it might otherwise, which is important for certain kinds of behaviors such as prediction, navigation, hunting, etc. Third, stability can make the system more robust; i.e., more resistant to undesirable perturbations. For instance, such networks are good at noise reduction (see, e.g., Pouget et al. 1998). Because these networks are constructed so as to have only a certain set of stable states, random perturbations to nearby states quickly dissipate to a stable state. If, for example, we give the bump network a 'noisy bump' as an input signal, in a few time steps it settles to the nearest (i.e., most similar) bump that lies along the trajectory joining stable bump points. This is because the state space has been constructed so that the gradients are lowest near the stable points. Thus the steeper gradients far away from the stable points ensure a quick 'relaxation' to more stable states. As well, if a stable bump is perturbed by randomly changing the firing rates of a few neurons, the effects are minimal because such perturbations move the bump to a nearby, but less stable point in the state space, which quickly relaxes back to the original state. These behaviors make attractor networks effective for filtering out noise in any particular band because we can define the set of functions that are to be part of the attractor network in such a way as to exclude any functions we deem undesirable (e.g., those with high frequencies). As well, the same mechanisms that support noise reduction make attractor networks suitable for categorization tasks, as has been emphasized by Hopfield (1982).

To conclude, we note that we have not *completely* generalized attractor dynamics. For example, we did not discuss either the torus or chaotic attractors. There is no reason that this framework could not be used to examine these classes of attractors as well. With the introduction of these new kinds of attractors comes different possible computational strategies. A number of researchers have examined the computational role that these more complex kinds of dynamics might play in neurobiological systems (see, e.g., Skarda and Freeman 1987; King 1991). Furthermore, we have not discussed the possibility of the attractors themselves being dynamic. That is, the possibility of constructing attractors 'on-the-fly' to perform some function. For instance, if we want to robustly reach towards some target, one way of accomplishing this task is to tailor the limb state space to have a point attractor at exactly that point. Again, integration of our approach with control theory should help to construct plausible neurobiological models with these kinds of properties.

8.5 An example: Lamprey locomotion

8.5.1 Introduction

To conclude this chapter, we present a model that draws on and extends many of the central themes we have presented to this point, including control theory, function representation, and attractor dynamics. More than simply being an example of the application of these ideas, however, we think that modeling lamprey locomotion makes an important methodological point as well.

Lamprey swimming has been subject to intense experimental and theoretical scrutiny for well over two decades (Cohen et al. 1982; Grillner et al. 1991). This is because the isolated lamprey spinal cord makes an excellent experimental platform, and because the exhibited behavior is amenable to current theoretical tools. Specifically, the isolated cord, even relatively small portions, exhibits oscillatory bursting activity when bathed in excitatory amino acids. This 'fictive swimming' resembles the activity seen in the whole swimming lamprey, so this preparation is ideal for performing tests that are directly relevant to whole animal behavior. Furthermore, the activity of each short portion of spinal cord is itself well-modeled by a simple biphasic oscillator. The entire cord can thus be modeled as a chain of biphasic oscillators (Cohen et al. 1982; Kopell and Ermentrout 1988). Because there are mathematical tools available for analyzing this kind of system, the theoretical analysis has been useful for guiding experiments and making predictions, undoubtedly providing a good example of the successful interaction between theory and experiment (Marder et al. 1997).

In addition, finding isolated groups of neurons that independently exhibit such rhythmic patterns fits well with the now ubiquitous idea that central pattern generators (CPGs) underly much of motor behavior (Selverston 1980). A CPG is a group of neurons that can produce rhythmic patterns without sensory input, just as seen in the lamprey. So, the spinal cord is typically thought of as a series of local CPGs connected in series, and analyzed as such (Grillner et al. 1991; Lansner et al. 1997; Kopell and Ermentrout 1988). Given this way of understanding lamprey motor behavior, the standard methodology is what is often called 'bottom-up'. That is, small functional units (i.e., local CPGs) are identified, their properties characterized in great detail, and then they are connected together in various ways to emulate some more complex behavior (i.e., lamprey swimming). As successful as this approach has been, we think that it has some central limitations that our framework can help address. In particular, because our framework provides a means of characterizing a system at many levels at once, it can help integrate 'top-down' data with this kind of 'bottom-up' approach. In other words, we do not want to offer a replacement for the CPG methodology, but an extension that will help us constrain models from the 'top' and the 'bottom'.

The limitations of the purely bottom-up approach have not gone unnoticed by those building these models. For example, Marder et al. (1997) note that networks comprised of chains of coupled oscillators in no way guarantee oscillations and often produce synchrony: a deadly result for an organism. As well, Wannier et al. (1998) have noted that it is difficult to control the direction and frequency of the oscillations arising from such networks. Because we have a means of choosing both the representation and the higher-level control structures in our model, both of these kinds of difficulties can be avoided from the start, as we discuss below. Furthermore, the kinds of simplifications that modelers make in order to map a chain of biphasic oscillators onto the lamprey nervous system are different that those we tend to make. For example, such models generally assume that the coupling between neighboring oscillators is both highly localized and discrete (Wilson 1999b, p. 212). However, this is known not to be the case for the lamprey spinal cord. As well, the weights used to drive the network are 'hand-picked' to produce the desired behavior. Using our approach, connection weights are determined analytically and approximate very well the pattern of connectivity observed in the lamprey.

Before presenting the example, it is essential to emphasize that we take our approach to be *complimentary* to the typical CPG approach. That is, both top-down and bottom-up analyses of such systems are essential for a complete understanding of the system. Our point is that having a framework that allows both kinds of data to inform the construction of such models should be preferred over a 'one-sided' approach.

8.5.2 System description

The lamprey family consists of about 17 species in North America. These eel-like fish range in size from approximately 10–60 cm when full grown. They sport a long single or double dorsal fin that joins a caudal fin at the tail (Becker 1983). Phylogenetically ancient, lamprey are often characterized as the simplest vertebrate, suggesting that they are both experimentally accessible and relevant for understanding more complex organisms. As well, their main mode of movement, an undulatory swimming, is fairly simple to characterize mathematically. In the remainder of this section, we highlight some of the central constraints on a good model of lamprey locomotion before identifying the relevant representation and considering how to provide a mathematical description of the system's behavior.

Some of the central features of lamprey neurophysiology and behavior are (see Wilson 1999b; Lansner et al. 1997; Grillner et al. 1991; Marder et al. 1997):

1. the spinal cord is continuous, but made up of about 100 segments;
2. connectivity is mostly local, but spans several segments;
3. connectivity is asymmetric;

4. individual neurons code muscle tension over small regions;

5. during swimming, neural firing on one side of the lamprey oscillates in counter-phase with a similar population on the other side (i.e., biphasic oscillation is observed);

6. the length of the lamprey is equal to about 1 period of the swimming wave;

7. the inter-segmental phase lag is independent of swimming frequency (i.e., 6. is true regardless of swimming speed); and

8. the lamprey can swim between about .25 and 10 Hz forwards, and can swim backwards.

These constraints are all met by our model. Some of these constraints give us important hints regarding what kinds of representations and dynamics are in this system. For instance, neurons in the cord seem to only code muscle tensions over small regions, so we can assume a local Gaussian-like tuning curve for neurons in the cord:

$$a_i(z,t) = G_i\left[\alpha_i \left\langle \tilde{\phi}_i(z) T(z,t) \right\rangle_z + \beta_i\right],$$

where $T(z,t)$ is the tension in the muscles along the length, z, at time t. The encoding functions, $\tilde{\phi}_i(z)$, should be spaced approximately equally, and close together along the length of the lamprey. Furthermore, we need to ensure that the frequency of swimming, ω, can be controlled and that there is always one period of swimming wave evident in the lamprey's body position. The remaining constraints will be important during the implementation stage.

We now need to describe how the represented variable, $T(z,t)$, relates to the observed lamprey behavior. As mentioned, the lamprey produces a traveling wave along its length when swimming. An equation describing this kind of wave is (see figure 8.23)

$$x(z,t) = A\sin(\omega t - kz). \tag{8.29}$$

We can solve this equation directly and compare it to the motion of the swimming lamprey. Doing so shows that it is a good description of lamprey locomotion. This, then, provides a reasonable high-level characterization of the observed behavior of the system, one which we can use to constrain our neural-level model.

In order to relate this high-level description to the firing of neurons in the spinal cord, we need to describe how this network controls the muscles along the length of the lamprey so as to result in an undulatory motion. To do this, we can invoke Newton's second law of motion: $\mathbf{F} = m\mathbf{a}$. In the case of the lamprey, we know the displacement over time (and thus the acceleration) of the system, and can use this information to determine what forces are needed to give rise to the observed displacement. Because the lamprey swims in a liquid, the forces that arise are due to the resistance of the liquid to the movement of the lamprey's body. As a result, fluid dynamics can help us get an expression for the forces that arise given the observed displacement.

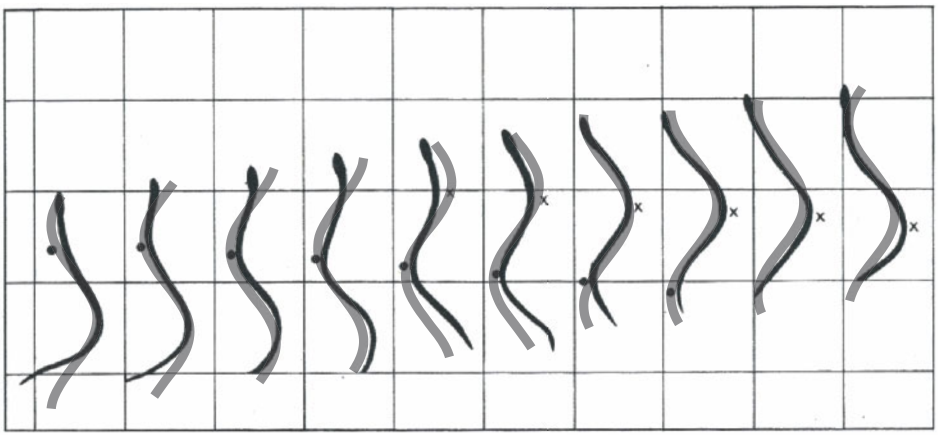


Figure 8.23
Lamprey swimming motion and direct solution (grey line) of equation (8.29). The lamprey was photographed every 0.09 s during normal swimming. The equation provides a good approximation to the lamprey swimming, although the amplitude of the lamprey wave varies from head to tail (being smaller at the head), unlike the equation solution. (Adapted from Gray (1933) © The Company of Biologists Ltd., reproduced with permission)

In general, the normal force resulting from moving an object through a liquid is a function of velocity. The simplest such function is

$$\begin{aligned} F^n(z,t) &= \eta \dot{x} \\ &= \eta A \omega \cos(\omega t - kz), \end{aligned}$$

where η is the viscosity of the liquid (water in this case), and $F^n(z,t)$ is the normal force at position z at time t. We use this expression even though it truly only holds in the regime of low Reynold's numbers (i.e., for 'creeping' or 'Stokes' flow). While this expression does not capture the exact force that would be experienced by a lamprey, it serves to keep the problem simple.[11]

11 A much better expression for the normal force is $F^n = C_d \eta \dot{x}^2$ where C_d is the coefficient of drag (this is a function of the Reynold's number). In the regime of active normal velocities ($\dot{x} > 1/3$ m/s), C_d is approximately constant. Following the procedure outlined below, this results in an expression for the tensions of $T(z,t) = \frac{\kappa}{4}\left[\cos(2\omega t)(z - \frac{1}{2}) - \frac{1}{2}k\cos(2\omega t)\sin(2kz)\right]$ which makes the remaining derivation more difficult, though still tenable. However, even this expression assumes a static flow velocity. To generate an even more accurate model, oscillatory flow should be used to calculate the normal force (Sumer and Fredsøe 1997).

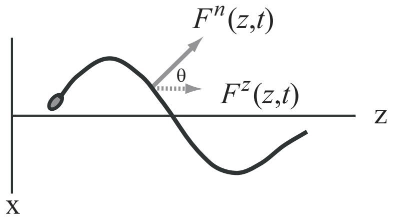


Figure 8.24
Components of the force generated by lamprey swimming. $F^z(z,t)$ is the propelling force in the z direction and $F^n(z,t)$ is the force normal to the lamprey's body at z.

To determine the force contributing to the forward motion of the lamprey, we can find the component of the normal force acting along the z axis (see figure 8.24):[12]

$$\begin{aligned} F^z(z,t) &= \cos\theta \cdot F^n(z,t) \\ &\approx F^n(z,t)\frac{\partial x(z,t)}{\partial z} \\ &= -\eta A^2 \omega k \cos^2(\omega t - kz). \end{aligned}$$

We now need to to relate these forces to the contractions of muscles along the lamprey's body. To do so, we use the simple model of lamprey mechanics shown in figure 8.25. As shown in appendix F.2.1, we can solve for the tensions needed to perform the requisite swimming. This results in the expression

$$T(z,t) = \kappa(\sin(\omega t - kz) - \sin(\omega t)), \tag{8.30}$$

where $\kappa = \frac{\gamma\eta\omega A}{k}$, $k = \frac{2\pi}{L}$, $A = 1$ is the wave amplitude, $\eta = 1$ is the normalized viscosity coefficient, $\gamma = 1$ is the ratio of intersegmental and vertebrae length, $L = 1$ is the length of the lamprey, and ω is the swimming frequency.

We have now mathematically defined the transformations and representations relevant for understanding the system we are modeling. Notice that this equation defines a whole set of possible swimming behaviors that are parameterized by ω. In other words, this expression describes lamprey locomotion at any speed (forwards or backwards), which can be controlled by varying ω.

8.5.3 Design specification

As usual, we must now specify the noise, range and precision relevant to these representations. Again, because we are interested in the representation of functions, this step is

12 This approximation holds since $\cos\theta = \sin(90 - \theta) \approx \frac{\Delta x}{\Delta z}$, where θ is the angle between the force vectors.

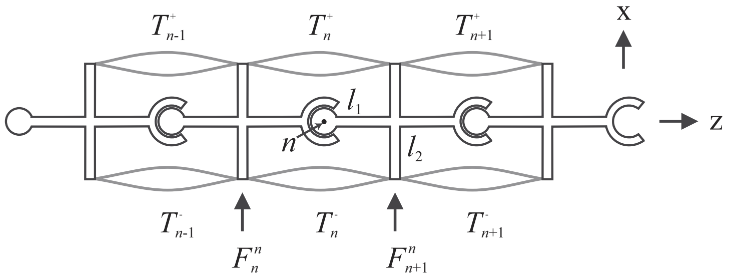


Figure 8.25
A simple model of lamprey mechanics used to relate muscle tension, $T(z,t)$, to the forces, $F(z,t)$, generated during swimming. Here the body has been discretized into n segments, where the distance between the segments is $2l_1$ and the width of the segment is $2l_2$.

broken into two parts; identifying the function space, and specifying constraints on the neurons themselves.

To begin, we define a representation of the dynamic pattern of tensions in terms of the coefficients $A_n(t)$ and the orthonormal spatial harmonic functions $\Phi_n(z)$:

$$T(z,t;\mathbf{A}) \quad = \kappa\left(A_0 + \sum_{n=1}^{N} A_{2n-1}(t)\sin(2\pi nz) + A_{2n}(t)\cos(2\pi nz)\right). \quad (8.31)$$

We can find the appropriate coefficients by forcing the difference between $\hat{T}(z,t)$ and $T(z,t)$ to be zero (see appendix F.2.2). As a result, we find that $A_0(t) = -\sin(\omega t)$, $A_1(t) = -\cos(\omega t)$, $A_2(t) = \sin(\omega t)$, and for $n > 2$, $A_n(t) = 0$. This is a representation in a higher-level orthogonal space of the ensemble of functions we want to be able to generate. We can specify the domain of the functions, $T(z,t;\mathbf{A})$ because we know that the lamprey is of length L so $z \in (0,L)$ and clearly $t \geq 0$. Finally, the frequency parameter defining the $\mathbf{A}$ space can range between about -2 Hz (backwards) and 10 Hz (a fast swim). This completes the design specification in the orthonormal space.

As usual, we take the neuronal noise to have a variance of 0.1. However, there is no good evidence as to how precise the tension in each segment along z is encoded. So, we simply make a (presumably conservative) assumption of 200 neurons per segment. We have now specified the range and precision of our encoding for both the higher level and basic representations.

8.5.4 Implementation

In order to implement the dynamics of lamprey swimming using the representation defined above, we need to make our dynamical description amenable to this representation. Employing the standard control theory form discussed in section 8.1, we can write the high-level state-space equation as

$$\dot{\mathbf{A}} = \mathbf{M}_D\mathbf{A} + \mathbf{M}_I\mathbf{U}, \tag{8.32}$$

where $\mathbf{A}$ are the amplitudes in our orthonormal space, $\mathbf{M}_D$ is the dynamics matrix, $\mathbf{M}_I$ is the input matrix, and $\mathbf{U}$ is the input, in this case a startup signal.

However, we are clearly interested not only in implementing these dynamics, but in implementing them using a *neural* representation. As a result of using such a representation, there are many unforeseen sources of error (from noise, spikes, etc.). The dynamics we define in the orthogonal space should be robust to such errors. To ensure that this is the case, we can enforce the constraint that, over time, the error goes towards zero despite any perturbations. To do so, let us write the error

$$E = \left\langle [T(z,t) - \hat{T}(z,t)]^2 \right\rangle_{t,z}, \tag{8.33}$$

where $T(z,t)$ is defined by (8.30) and $\hat{T}(z,t)$ is defined by (8.31). Solving (8.33) results in the expression (see appendix F.2.2)

$$\begin{aligned} E \;=\; & \left\langle \frac{1}{2}(A_1 + \cos(\omega t))^2 + \frac{1}{2}(\sin(\omega t) + A_0)^2 \right. \\ & \left. + \left(\frac{A_0 + A_2}{2}\right)^2 + \left(\frac{A_0 - A_2}{2} + \sin(\omega t)\right)^2 + \sum_{n=3} A_n^2(t)_t \right\rangle_t . \end{aligned} \tag{8.34}$$

To ensure that this error goes to zero over time, each of the terms in this expression must also go to zero. The third term suggests that we need to damp the sum of A_0 and A_2 over time to keep the error small. Together with the fourth term, it also suggests that we must enforce the constraint $A_0 = -A_2 = -\sin(\omega t)$. As well, we can see that the first and second terms together identify an oscillator since

$$\begin{aligned} A_1 &= -\cos(\omega t) \\ A_0 &= -\sin(\omega t). \end{aligned}$$

As can be shown (see appendix F.2.3), these dynamics can be written in standard control theory form as

$$\begin{aligned} \dot{A}_0 &= -\omega A_1 \\ \dot{A}_1 &= \omega A_0 \end{aligned}$$

or,

$$\dot{\mathbf{A}} = \mathbf{M}\mathbf{A},$$

where

$$\mathbf{M} = \begin{bmatrix} 0 & \omega \\ -\omega & 0 \end{bmatrix}.$$

This, of course, is the matrix that defines a cyclic attractor (see section 8.4.3). So, not surprisingly, the lamprey can be thought of as implementing cyclic attractor dynamics. Viewing the lamprey's behavior in this way demonstrates otherwise obscure similarities that this system has to LIP and the neural integrator. It is because we have a framework that can integrate 'top-down' considerations that such connections become evident. These connections can prove useful because they can give us insight into what kinds of perturbations, representational constraints, etc. are important for characterizing a novel system.

In the specific case of the lamprey model, the oscillator matrix alone will not ensure that the constraints we previously identified are enforced. So, we need to explicitly include the term

$$\frac{d(\frac{A_0+A_2}{2})}{dt} = 0.$$

We can now include a row and column of zeros in $\mathbf{M}$ to enforce this constraint. However, our matrix is now in a different coordinate system than our original variables (one which includes half the sum of A_0 and A_2 as an axis). Thus we must perform a coordinate transformation back to the original space. Doing so gives (see appendix F.2 section F.2.4)

$$\begin{aligned} \mathbf{M_A} &= \mathbf{R}^{-1}\mathbf{M}\mathbf{R} \\ &= \begin{bmatrix} 0 & \omega & 0 \\ -\omega & 0 & 0 \\ 0 & -\omega & 0 \end{bmatrix}, \end{aligned} \qquad (8.35)$$

where $\mathbf{R}$ is the coordinate transformation matrix. Thus, $\mathbf{M_A}$ is the matrix that implements the desired dynamics and enforces the constraints in our original space.

However, this matrix does not ensure robustness to errors in representation of the components $\mathbf{A}$. In particular, according to this matrix,

$$\frac{\dot{A}_0 + \dot{A}_2}{2} = \frac{-\omega\cos(\omega t) + \omega\cos(\omega t)}{2},$$

which evaluates to zero only if our representations of A_0 and A_2 are perfect. If they are not perfect, then this derivative is non-zero and a rapidly increasing error is introduced. Thus, we need to explicitly damp this term so it goes to zero even in the face of small errors in our representation. To do this, we construct a damping matrix, $\mathbf{M}_{damp}$, with damping terms, $-\alpha_0 < 0$:

$$\mathbf{M}_{damp} = \begin{bmatrix} -\alpha_0 & 0 & -\alpha_0 \\ 0 & 0 & 0 \\ -\alpha_0 & 0 & -\alpha_0 \end{bmatrix}.$$

Similarly, the final term in (8.34) tells us that we must damp the higher-order terms in the orthogonal representation. Intuitively, this is because the neural representation is quite local along the length of the lamprey, thereby introducing high frequency components not included in the orthogonal representation. As a result, we need to ensure that these high frequency components are damped out of the dynamics. We can do this by introducing a second damping term, $-\alpha < 0$ and applying it to any higher frequency amplitudes that may be introduced. We can determine how many higher frequency amplitudes to damp out by noting the width of the tuning curves of neurons found along the length of the lamprey. We need only damp those amplitudes that might be introduced by tuning curves of this width.

We can combine these damping terms in a single matrix, increasing the dimensionality of the original matrix and including the $-\alpha$ term along the diagonal. This matrix then damps all sources of error introduced by our representation:

$$\mathbf{M}_{damp} = \begin{bmatrix} -\alpha_0 & 0 & -\alpha_0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -\alpha_0 & 0 & -\alpha_0 & 0 & 0 \\ 0 & 0 & 0 & -\alpha & 0 \\ 0 & 0 & 0 & 0 & \ddots \end{bmatrix}.$$

We have now effectively defined the dynamics matrix, $\mathbf{M}_D$, so let us turn to the input matrix, $\mathbf{M}_I$. This matrix is essential for starting the swimming motion, and could also be used to increase or decrease the amplitude of the swimming wave. For convenience, we have chosen startup dynamics defined by the startup matrix $\mathbf{M}_I$:

$$\mathbf{M}_I = \begin{bmatrix} \frac{1}{2} & 0 & -\frac{1}{2} \\ 0 & 1 & 0 \\ -\frac{1}{2} & 0 & \frac{1}{2} \end{bmatrix}.$$

Substituting this matrix into (8.32) and allowing $\mathbf{U} = \mathbf{A}$ shows that the resulting equation is $\dot{\mathbf{A}} = \mathbf{A}$, giving an exponential startup while satisfying the constraints imposed by (8.34).

Putting these matrices together, we can create a set of dynamical equations that result in the desired behavior:

$$\dot{\mathbf{A}} = \mathbf{M}_D\mathbf{A} + \mathbf{M}_I\mathbf{U} = [\mathbf{M}_{\mathbf{A}} + \mathbf{M}_{damp}]\mathbf{A} + \mathbf{M}_I\mathbf{U}(t - t_s), \tag{8.36}$$

where the function $\mathbf{U}(t - t_s)$ is a step function that goes to $\mathbf{A}$ at $t = 0$ and goes to zero at $t = t_s$, controlling the application of the startup matrix. This completes a high-level control theoretic characterization of the system. Note that this system is guaranteed to be stable, to oscillate as expected, and its frequency can be controlled through manipulation of ω. Changing ω can be incorporated by allowing $\mathbf{M}_D$ to be time-varying, as in the neural integrator (see section 8.2.1).

In previous examples, we often embed this kind of high-level characterization directly into a neural representation. However, in this case, let us introduce a level of representation that is intermediate between this high-level description and the neural-level description. In other words, we use two levels in the representational hierarchy defined by table 3.2. In particular, we can define the function representation in terms of a scalar representation. This is natural to do in this case for two reasons. First, the lamprey itself is composed of a series of about 100 segments, so considering each of those segments as representing the local tension reflects the physiology. Second, because running the model may be highly computationally demanding at the level of single spiking neurons, it is easier to explore some of the model's behaviors at an intermediate, and less computationally demanding, level. A simple choice for this intermediate level representation is to use one Gaussian encoding function for each segment. This is especially useful because it is so closely related to the eventual neural level representation. We can then proceed as usual:

$$T(t, z) = \sum_i a_i(t)\phi_i(z), \tag{8.37}$$

where we can find the decoding functions by minimizing the error as before. The $a_i(t)$ is a scalar that can be thought of as the 'segment activity', defined by

$$a_i(t) = \left\langle \tilde{\phi}_i(z)T(t, z)\right\rangle_z . \tag{8.38}$$

Together, (8.37) and (8.38) form an overcomplete representation of $T(z, t)$.

To use this representation, while preserving the higher-level dynamics, we can construct a projection operator between the higher-order orthogonal space and this space. This operator effectively allows us to move between these spaces (see appendix F.2 section F.2.5):

$$\Theta = \left[\tilde{\phi}\mathbf{\Phi}\right]^{-1}.$$

Next, we take advantage of Θ to transform the dynamical equations for the Fourier amplitudes, $\mathbf{A}$, given by (8.36), to dynamical equations in the intermediate space, $\mathbf{a}$ (see appendix F.2 section F.2.5):

$$\dot{\mathbf{a}} = \mathbf{m}_D\mathbf{a} + \mathbf{m}_I\mathbf{a}, \tag{8.39}$$

where

$$\begin{aligned}\mathbf{m}_D &= \Theta^{-1}\mathbf{M}_D\Theta \\ \mathbf{m}_I &= \Theta^{-1}\mathbf{M}_I\Theta.\end{aligned}$$

We can now simulate the lamprey's swimming in the intermediate space using this equation. Not surprisingly, the lamprey swims as expected.

Now, however, we would like to include neurons in our model. Because we have defined the intermediate representation encoding functions as a series of Gaussians (one for each segment) along the length of the lamprey, we can choose which of those segments we wish to represent using neurons. In other words, we can simulate some of the segments at the intermediate level and some of the segments at the neural level. Being able to mix degrees of neurobiological realism in a single simulation can prove extremely useful, especially for computationally intensive simulations.

In the case of the lamprey, we allow the neurons in a segment to represent what we earlier called the segment activity (see figure 8.26). In other words, we construct populations of LIF neurons to represent $a_i(t)$ in equation (8.37) for some set of segments, i. This essentially assumes that each neuron in a population has the same encoding function for tension, $\tilde{\phi}_i$, although their tuning curves will be quite different.[13] More explicitly, we can write

$$\begin{aligned}a_i(t) &= \sum_j b_j(t)\phi_j \\ b_j(t) &= G_j\left[\alpha_j a_i(t)\tilde{\phi}_j + J^{bias}\right],\end{aligned}$$

13 Alternatively, we could suppose that all neurons have different encoding functions, spaced randomly using an even distribution between the previous segment, and the next segment. This results in tuning curves more like those seen in the lamprey and behaves the same as the model discussed here. We assume all encoding functions are the same for simplicity.

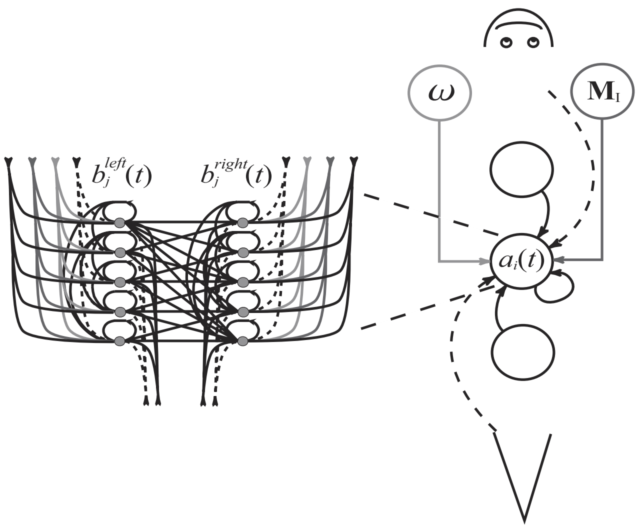


Figure 8.26
Model connectivity and the various levels of representation. Shown are the high-level control signals, ω and $\mathbf{M}_I$, segment level representations, $a_i(t)$, and individual neuron representation, $b_j(t)$. The $b_j(t)$ represent the $a_i(t)$ with respect to their basis, the $a_i(t)$ represent $T(z,t)$ with respect to their basis and ω parameterizes $T(z,t)$ at the highest level of description.

where $\tilde{\phi}_j = \pm 1$ are the encoding weights for the segment activity. If we replace these encoders with $\tilde{\phi}_i$ for the segment, we then find the tuning curves for the tension. Thus, we can relate all three levels of representation quite directly using this characterization.

Again the simulation behaves as expected. Now, however, we can look at the firing profiles of particular neurons to see if they match what is known from the neurophysiology. As figure 8.27 shows, the behavior over both the population and at the level of individual neurons matches what is seen in lamprey experiments.

8.5.5 Discussion

This model meets each of the constraints listed in section 8.5.2, although we have not discussed all of them in detail. However, the main purpose of our model is not simply to meet such constraints, but to do so while employing a top-down methodology. Clearly, some of the considerations employed while developing this model are justified by 'bottom-up' data (e.g., the choice of Gaussians as encoders for the intermediate representation). As a result, this model shows how we can be sensitive to both kinds of considerations concurrently. Having a principled means of integrating bottom-up and top-down constraints is essential for building complex models and testing hypotheses about neural function.

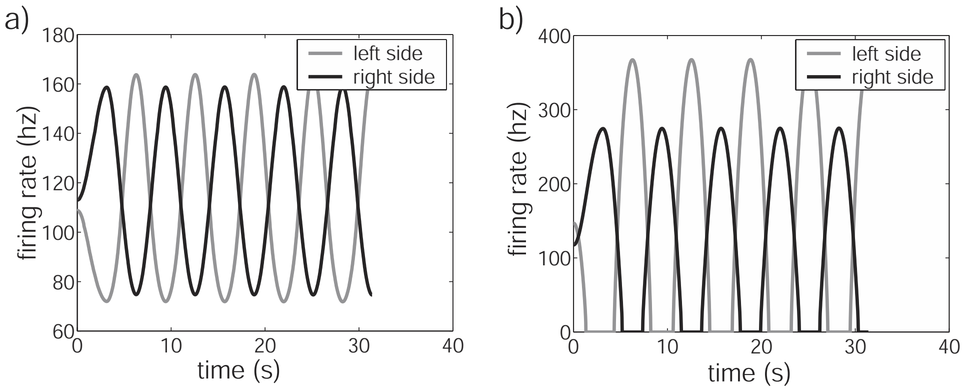


Figure 8.27
Neuron firing rates during startup and swimming. a) Mean firing rate as a function of time for the LIF populations encoding left and right tensions in the middle segment. b) Examples of single neuron example firing rates for LIF neurons taken from the same populations as in a). Note that this kind of oscillatory firing reflects that seen in the lamprey and in biphasic oscillator models.

While the bottom-up approach has resulted in more detailed models than the one presented here (see e.g., Ekekberg and Grillner 1999; Wadden et al. 1997), their intrinsic strengths and weaknesses are different. For instance, in bottom-up models, controlling the frequency of swimming is difficult and seldom accounts for the full range of lamprey swimming frequencies. As well, the control signal for backwards swimming is quite different from that for forward swimming (i.e., a bias signal must be switched from the head to the tail of the simulation). Our top-down approach, in contrast, makes control of both direction and frequency a simple matter; the sign of ω determines the direction and the magnitude of ω determines the frequency. As discussed in detail for the neural integrator model, elements of the dynamics matrix, including ω, can be controlled by an independent signal. Thus, forward and backward swimming are simply a result of a switch in sign of the control variable ω (i.e., positive and negative respectively) which does not result in a radically different kind of signal. Furthermore, there is significant concern regarding the stability of oscillator models (Marder et al. 1997). But, in the case of the above model, we have guaranteed stability by placing constraints on the allowable representation at the higher level (i.e., damping higher-order terms in the Fourier representation). Being able to include such high-level constraints in a principled manner becomes more important the more complex our models become.

This model also demonstrates a number of other techniques that become important as the complexity of our models increase. For instance, we showed a means of using the representational hierarchy to build progressively more detailed models of lamprey locomotion. Employing intermediate representations have a number of benefits. First, they result in significant computational savings, while highlighting novel properties of the lower-level network (e.g., interactions and coupling between neighboring segments). Second, intermediate representations make the connection between the higher-level characterization and the basic-level network more explicit, simplifying some derivations. And finally, well-chosen intermediate representations permit simulation of different parts of the network at different levels of detail simultaneously (e.g., segment and neural levels).

Beyond these methodological and practical points lies a deeper, by now familiar, theoretical message. This is that the methodology, techniques, and theory employed in building this simulation are general. Thus, some of the more abstract properties of this model, like the fact that it implements a cyclic attractor, not only become evident, but suggest a broader view of neurobiological systems. The payoff in terms of specific models like the lamprey, is that our understanding of controlling systems of this *kind* can now be applied directly to *this* neurobiological system. The real lamprey, of course, has a wide variety of swimming behavior that we have not addressed. However, using this framework, high-level analyses of the control of cyclic attractors can be applied for generating and testing hypotheses about how the lamprey controls its swimming to produce such behavior. The payoff in terms of understanding neurobiological systems more generally is that what we learn about lamprey behavior may be carried over to other, less familiar systems.

8.6 SUMMARY

At the beginning of the chapter, we set out to integrate what we had previously learned about neural transformation and representation with a general characterization of dynamics. Because control theory is well-established, very general, and appropriate for understanding both linear and nonlinear dynamic systems, we used it as a general means of introducing dynamics into the models we had developed to this point. To do so, we related the standard state space description of linear systems to one that accounted for intrinsic neural dynamics. This allowed us to revisit and precisely quantify the principles of neural engineering we had laid out in chapter 1, and to identify a general form for a generic neural subsystem.

The remainder of the chapter was largely taken up by different applications of this characterization. We began by revisiting the neural integrator. We demonstrated how to understand this system as a simple control structure, and how to extend the basic system to allow for the control of various matrix elements. This showed that our characterization of 'neural' control theory can be used to describe time-varying and nonlinear systems. We then turned to the working memory system in LIP. There we showed how to understand the dynamics of function representation, and how work on the integrator naturally generalized to this kind of model. We demonstrated that our approach could help derive a novel model able to account for phenomena not yet addressed by computational models.

We then discussed how the kinds of dynamics seen in both the integrator and the working memory system suggested a more general means of characterizing dynamics via the tools of dynamic systems theory. We noted that recent interest in attractor networks has been significant, but that there is not currently an approach for unifying the various kinds of attractor networks. We suggested that generalizations of both the representations and dynamics seen in contemporary models are naturally understood in terms of our framework. As well, such generalizations are a powerful way to characterize the behavior of many neural systems and subsystems.

Finally, we presented a fairly complex model that both exhibited a kind of attractor network we had not yet considered, and highlighted some strengths of this approach. In particular, the lamprey model demonstrated how relying on multiple levels of the representational hierarchy in a single model can be a useful technique. And, more importantly, this model showed that the framework can help us integrate both top-down and bottom-up information about a system. Being able to rely on a high-level characterization of a system's behavior when building a model can introduce important constraints that might be missed, or difficult to satisfy, by focusing on only low-level data.

In sum, the breadth of neurobiologically plausible models considered in this chapter provides good support to our central claim that the three principles of engineering are useful, data-driven, quantitative and can help unify our understanding of neurobiological systems (see section 1.6).

9 Statistical inference and learning

In each of the previous chapters, we have presented both theoretical discussions and their applications to specific neurobiological systems. In this chapter, in contrast, we present theoretical discussions and their applications to simple, ‘toy’ problems. While we think it is preferable to include specific biological examples, we also believe that it is valuable to explore some more theoretical ideas that are likely to be of use in understanding systems we have not yet modeled in detail. As a result, this chapter is the most speculative in the book, but it is nevertheless important because the issues it addresses are, we argue below, central to a broad understanding of neurobiological function. In other words, we think that any approach purporting to be a theory of neurobiological systems must have something to say about both statistical inference and learning. Both of these topics address how neural systems incorporate their particular environmental history to usefully inform future responses—something neurobiological systems are supremely adept at doing.

9.1 Statistical inference and neurobiological systems

Over two thousand years ago, Aristotle devised the first logical system in his treatise called the *Organon* (literally, ‘instrument’). His purpose was to identify *the* theoretical instrument that we can use to know things about the world. He suggested that language and thought, on the one hand, and reality, on the other, were isomorphic. Therefore, he took the rules of his logic to govern both of those domains. He did such a convincing job of determining these rules, that it took nearly two thousand years until significant alternatives were proposed. Currently, there are many alternatives to Aristotle’s original formulation, but perhaps the approach which deviates the most is the one that rejects language-like structures as fundamental; namely, probability theory. Rather than consider the *truth and falsity of sentences*, probability theory considers the *likelihood of events or states* of the world.[1] Because we are interested in neurobiological systems *in general*—nearly all of which do not have significant linguistic abilities, but all of which must reason about states of the physical world—it is reasonable to suspect that probability theory is the more appropriate tool for understanding such systems.

Unlike the certainty which Aristotle assumed, natural systems most often confront partial, noisy, uncertain information that they must use to make decisions; failure to use this information, no matter how degraded, may mean certain death. Again, probability

1 Of course, probability theory has been applied to language-like structures. For instance, when probabilities are considered to be degrees of belief in propositions. However, this is neither historically nor theoretically the foundational case.

theory and statistical inference naturally deal with cases in which information is partial. Furthermore, probability theory describes how to incorporate multiple uncertain sources of information to get a more certain result. And, similarly, it describes how to update a current 'take' on the world given novel information; this, of course, is learning. In general, then, probability theory is the best available quantitative tool for describing the kinds of reasoning evident in neurobiological systems.

A number of researchers have developed sophisticated probabilistic formalisms for modeling cognitive function (e.g., in computer vision; see Yuille and Clark 1993; Hallinan et al. 1999), and for modeling neurobiological function as statistical inference (Amit and Geman 1999; Knill and Richards 1996). One of the more general approaches to statistical modeling is known as *pattern theory* (Grenander 1981; Mumford 1996). The main purpose of this approach is to generate general models of complex, real-world patterns—like those encountered by biological systems.

Pattern theory incorporates another general approach, Bayesian inference, for describing pattern analysis and recognition. In our terminology, Bayesian inference defines transformations that are useful for working with the complex representations defined in pattern theory. In fact, Bayesian inference has been studied extensively in its application to reasoning about random scalar variables (Pearl 1988).[2] And, there is significant literature relating this general theory to particular algorithms and approaches in neural networks (see Jordan and Sejnowski 2001 for a review).

The preceding considerations all speak to the need for characterizing high-level statistical transformations. However, there are also basic-level considerations that suggest an interesting correspondence between neural architecture and the mechanisms needed to perform statistical inference. To see this correspondence, we must characterize these representations and transformations explicitly.

To begin, we know that all the information about a relation between any two random variables, $\mathbf{x}$ and $\mathbf{y}$, is captured by their joint distribution, $\rho(\mathbf{x}, \mathbf{y})$.[3] Probability theory tells us how to write this joint distribution in terms of a relation between the individual distributions, $\rho(\mathbf{x})$ and $\rho(\mathbf{y})$:[4]

$$\begin{aligned} \rho(\mathbf{x}, \mathbf{y}) &= \rho(\mathbf{y}|\mathbf{x})\rho(\mathbf{x}) && (9.1) \\ &= \rho(\mathbf{x}|\mathbf{y})\rho(\mathbf{y}). && (9.2) \end{aligned}$$

2 Such variables are the values of particular attributes of the underlying generators of patterns in pattern theory.

3 The function $\rho(\cdot)$ is referred to as a probability density function (PDF). It can be thought of as relating the values of a continuous random variable to the likelihood each value has of occurring. A PDF is a continuous version of the more familiar normalized histogram. The PDF of a random variable, x, describes the probability that x lies between any two values; i.e., $P(a \leq x \leq b) = \int_a^b \rho(x)\, dx$.

4 Note that Bayes' rule consists of equating the two right hand sides of the following equations and solving for either $\rho(\mathbf{x})$ or $\rho(\mathbf{y})$ as needed.

Furthermore, probability theory tells us how to determine the probability density function (PDF) for either parameter alone by 'marginalizing' (i.e., integrating) the joint distribution:

$$\rho(\mathbf{y}) = \int \rho(\mathbf{x}, \mathbf{y})\, d\mathbf{x} \tag{9.3}$$

$$\rho(\mathbf{x}) = \int \rho(\mathbf{x}, \mathbf{y})\, d\mathbf{y}. \tag{9.4}$$

In a neurobiological context, each of these PDFs would be represented in a neural population. In the context of vision, for instance, we can think of $\rho(\mathbf{x})$ as representing how likely each image is in the environment. Of course, this estimate can only be usefully made in the context of some evidence, or data, provided by the environment. This data could be the image falling on the retina that is corrupted by noise. We would then construct the conditional PDF, $\rho(\mathbf{x}|\mathbf{d})$, that corresponds to the probability density function for the true image given the measured image, $\mathbf{d}$.

In analyzing images, it is generally useful to extract certain properties of the image that are not directly captured by the image falling on the retina (i.e., $\mathbf{d}$), such as objects in the visual field, optic flow fields, etc. To extract these properties, various transformations must be performed. Letting the variable $\mathbf{y}$ correspond to such properties, and generalizing (9.1), the probability density function for the variable $\mathbf{y}$ can be determined as a weighted average of the conditional $\rho(\mathbf{y}|\mathbf{x})$:

$$\rho(\mathbf{y}|\mathbf{d}) = \int \rho(\mathbf{y}|\mathbf{x})\rho(\mathbf{x}|\mathbf{d})\, d\mathbf{x}. \tag{9.5}$$

Equation (9.5) tells us, for instance, how to determine the probability of the existence of all possible objects in the field of view given the input image, $\mathbf{d}$. While this particular example is clearly oversimplified, and, as we show, probably impossible to implement, it serves to demonstrate the potential power of this kind of formulation. Notably, equation (9.5) is simply a linear transform, or projection of $\rho(\mathbf{x}|\mathbf{d})$ into the new space $\rho(\mathbf{y})$. Nevertheless, it includes the transformations we have talked about to this point, i.e., $\mathbf{y} = f(\mathbf{x})$, as a subset. This can be seen by forming the conditional[5]

$$\rho(\mathbf{y}|\mathbf{x}) = \delta(\mathbf{y} - f(\mathbf{x})), \tag{9.6}$$

and solving (9.5).

Importantly, (9.5) frees us from the assumption of Gaussian statistics. For example, it allows multi-modal distributions in $\rho(\mathbf{x})$ and $\rho(\mathbf{y})$ (see section 9.2). As well, the conditional connecting the two spaces, $\rho(\mathbf{y}|\mathbf{x})$, can also be multi-modal, allowing unimodal

5 A slightly blurred version, $\rho(\mathbf{y}|\mathbf{x}) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(\mathbf{y}-f(\mathbf{x})^2)/2\sigma^2}$, is nearly as simple and begins to take advantage of the extra power of this statistical formulation.

inputs to support multiple hypotheses. As a result, being able to implement transformations defined by (9.5) results in computationally powerful systems. This is demonstrated by the many successes of artificial neural network (ANN) models, which can implement precisely these transformations. It should come as no surprise that real neurobiological networks are also ideally suited to implementing equation (9.5).

To see how, notice that $\rho(\mathbf{x}|\mathbf{d})$ is simply a function parameterized by the variables $\mathbf{d}$ (just as $x(\nu;\mathbf{A})$ is parameterized by $\mathbf{A}$; see chapter 3). Hence we can create ensembles of neurons to encode $\rho(\mathbf{x}|\mathbf{d})$ and $\rho(\mathbf{y}|\mathbf{d})$ as

$$\begin{aligned} a_i(\mathbf{d}) &= G_i\left[\alpha_i\left\langle\tilde{\phi}_i(\mathbf{x})\rho(\mathbf{x}|\mathbf{d})\right\rangle_{\mathbf{x}} + J_i^{bias}\right] \\ b_j(\mathbf{d}) &= G_j\left[\alpha_j\left\langle\tilde{\phi}_j(\mathbf{y})\rho(\mathbf{y}|\mathbf{d})\right\rangle_{\mathbf{y}} + J_j^{bias}\right], \end{aligned} \tag{9.7}$$

with the corresponding decoding rules

$$\begin{aligned} \hat{\rho}(\mathbf{x}|\mathbf{d}) &= \sum_i \phi_i(\mathbf{x})a_i(\mathbf{d}) \\ \hat{\rho}(\mathbf{y}|\mathbf{d}) &= \sum_j \phi_j(\mathbf{y})b_j(\mathbf{d}). \end{aligned} \tag{9.8}$$

Using equation (9.5) to define the transformation between these two function ensembles, we find that the neuronal variables are related by

$$\begin{aligned} b_j(\mathbf{d}) &= G_j\left[\alpha_j\left\langle\tilde{\phi}_j(\mathbf{y})\rho(\mathbf{x}|\mathbf{y})\rho(\mathbf{x}|\mathbf{d})\right\rangle_{\mathbf{x},\mathbf{y}} + J_j^{bias}\right] \\ &= G_j\left[\alpha_j\left\langle\tilde{\phi}_j(\mathbf{y})\rho(\mathbf{x}|\mathbf{y})\sum_i \phi_i(\mathbf{x})a_i(\mathbf{d})\right\rangle_{\mathbf{x},\mathbf{y}} + J_j^{bias}\right] \\ &= G_j\left[\sum_i \omega_{ji}a_i(\mathbf{d}) + J_j^{bias}\right], \end{aligned} \tag{9.9}$$

where

$$\omega_{ji} = \alpha_j\left\langle\tilde{\phi}_j(\mathbf{y})\rho(\mathbf{x}|\mathbf{y})\phi_i(\mathbf{x})\right\rangle_{\mathbf{x},\mathbf{y}}. \tag{9.10}$$

Within this framework, equations (9.9) and (9.10) tell us how to implement simple feed-forward statistical inference in a neurobiologically plausible network. In particular, the connection weights between neurons in these networks can be understood as the projection of the encoding functions of the output neurons, $\tilde{\phi}_j(\mathbf{y})$, on the conditional, $\rho(\mathbf{y}|\mathbf{x})$, weighted by the decoding function, $\phi_i(\mathbf{x})$, of each input neuron.

Examining these equations more closely reveals a number of important consequences. First, statistical inference in high-dimensional spaces requires estimating high-dimensional integrals. That is, information from many sources must be 'added up'. Because of the high degree of convergence on each neuron in typical neural networks, such networks are ideally suited for performing these kinds of transformations. Given the previous considerations regarding the importance of such inference, in some ways this is not at all surprising.

Second, the form of these equations is identical to those for a simple feed-forward ANN. Furthermore, given our past analysis of neurobiological representation, we can see that there is no particular advantage gained by introducing the nonlinearity $G_i\left[\cdot\right]$ (especially on the higher-level variables). The nonlinearity is simply due to the nature of the representation found in neurobiologically plausible networks. Perhaps the nonlinearity is more the result of an implementational constraint on low-power, low-precision physical devices like neurons.

Third, this formulation of statistical inference using PDFs represented by a set of basis functions is more general than one that assumes Gaussian statistics. This is because the variables in this more general formulation (i.e., a_i) are all treated in the same manner. In contrast, when using Gaussian statistics, the mean and covariance matrices are modified by different sets of rules.

And finally, (9.5) can be understood by taking the conditional $\rho(\mathbf{y}|\mathbf{x})$ to be a 'look-up table' (LUT), that specifies the value of $\mathbf{y}$ for every value of $\mathbf{x}$. When viewed in this way, the limitations of (9.5) become clear—every possible consequence given the input $\mathbf{x}$ must be pre-computed (i.e., embedded in ω_{ji}). This is why we referred to our previous example in the visual system as oversimplified; that kind of simple feed-forward formulation of the problem would require astronomically high amounts of resources to represent $\rho(\mathbf{y}|\mathbf{d})$. More importantly, the conditional, $\rho(\mathbf{y}|\mathbf{x})$, would have to relate every possible image (a high-dimensional space) to every possible object (another very high-dimensional space). As a result, the addition of one new object (i.e., increasing the dimension of the object space by one) requires enough resources to define its relation to every possible image—this problem is commonly called the 'curse of dimensionality.'

One way to begin to address this problem is to recognize that it is often possible to divide such high-dimensional spaces into statistically independent subspaces. Let us suppose that $\mathbf{x}$ can be so divided into $\rho(\mathbf{x})$ and $\rho(\mathbf{z})$. Then, the joint density $\rho(\mathbf{x},\mathbf{z})$ can be written as $\rho(\mathbf{x})\rho(\mathbf{z})$. Now, the transformation for finding the PDF for $\mathbf{y}$ given those for $\mathbf{x}$ and $\mathbf{z}$ becomes

$$\rho(\mathbf{y}|\mathbf{d}_\mathbf{x},\mathbf{d}_\mathbf{z}) = \int \rho(\mathbf{y}|\mathbf{x},\mathbf{z})\rho(\mathbf{x}|\mathbf{d}_\mathbf{x})\rho(\mathbf{z}|\mathbf{d}_\mathbf{z})\,d\mathbf{x}\,d\mathbf{z}. \tag{9.11}$$

An implementation of (9.11) in a neural network has the form

$$b_j(\mathbf{y}) = G_j\left[\sum_{ik}\omega_{jik}a_i(\mathbf{d_x})c_k(\mathbf{d_z}) + J_j^{bias}\right], \tag{9.12}$$

where

$$\omega_{jik} = \left\langle \tilde{\phi}_j(\mathbf{y})\rho(\mathbf{y}|\mathbf{x},\mathbf{z})\phi_i(\mathbf{x})\phi_k(\mathbf{z})\right\rangle_{\mathbf{x},\mathbf{y},\mathbf{z}}. \tag{9.13}$$

Equation (9.11) defines a much richer class of inference than that supported by (9.5). In fact, there are a number of possible interpretations of (9.11) in a neurobiological context. Most generally, we can think of the variables $\mathbf{z}$ dynamically changing the connection weights between $\mathbf{y}$ and $\mathbf{x}$ such that the inference between the latter two spaces is carried out in the *context* defined by the $\mathbf{z}$ space. But, notice that (9.5) does not make a distinction between which space is providing the context and which is being modulated. This is what gives rise to different interpretations. Thus $\mathbf{x}$ and $\mathbf{z}$ could represent evidence from two-different modalities such as vision and audition, in which case the circuit is still feed-forward. Alternatively, $\mathbf{z}$ could represent variables in a higher order ('top-down') model and $\mathbf{x}$ could provide the feed-forward ('bottom-up') evidence. A more specific possibility is that the $\mathbf{z}$ variables determine the location where covert attention is directed, and $\mathbf{x}$ is a visual image, resulting in an attention-like, context dependent processing of the evidence—in this case the transformation defines a *shifter circuit* (Anderson and Van Essen 1987; Olshausen et al. 1993).

The powerful set of transformations that results from the generalization of equation (9.5) to (9.11) does not come for free, however. Most noticeably, (9.12) requires multiplicative (i.e., nonlinear) interactions between the activities of the $\mathbf{z}$ and $\mathbf{x}$ populations. Of course, these could be implemented using intermediate neurons as described in section 6.3. However, it would be much more efficient to put the nonlinear interactions into the dendritic trees of neurons. The ubiquitous need for performing this kind of context dependent statistical inference is one more reason that we might expect to find such nonlinearities. Others have made similar suggestions. Kawato (1995), while modeling the cerebellum, has similarly argued that such nonlinearities are essential. As a result, Schweighofer et al. (1998) have implemented models in which the dendritic trees of Purkinje cells are composed of approximately 50 subunits that act as simple feed-forward neurons described by (9.9).

So, both at higher levels and the neural level, neurobiological systems have an affinity—and a need—for processing statistical information. In the remainder of this section we show how the theory of Bayesian inference can be incorporated into our framework by considering some simple examples. Although this discussion focuses on single random variables, the same techniques are appropriate for more complex representations, as usual. That is, although the examples provided are simple, they demonstrate how all of the essential features of pattern theory and Bayesian inference can be incorporated into neurobiological simulations.

9.2 An example: Interpreting ambiguous input

Because the natural environment is often filled with irrelevant or partial information, there is good reason to expect that phenomena like object recognition do not depend solely on information extracted from sensory signals. Rather, many processes must depend critically on the system's assumptions about the structure of the environment. In other words, 'top-down' information is essential for the successful performance of many tasks. In the context of object recognition, such a view directly contrasts with the classical view of recognition being performed in a strictly feed-forward or 'bottom-up' manner (Marr 1982). However, it has become clear, both functionally and anatomically, that top-down effects are common. Functionally, top-down psychological effects in vision have been extensively observed (Gregory 1997).[6] Anatomically, it is clear that there are massive reciprocal feedback projections from later to earlier visual areas in the primate (Van Essen et al. 1992). Cavanagh (1991) has argued that top-down influences are *essential* for successful processing of heavily degraded signals—such signals seem to be the rule rather than the exception in natural settings.

As discussed earlier, statistical inference formalizes just this kind of reasoning, so let us apply it to a 'toy' characterization of a problem faced by animals. Suppose an animal is interested in determining the location of an object in the external world. Since the actual location is a matter of some uncertainty, we can describe the animal's final 'guess' as to where the object is as a probability density function, $\rho(y)$. This guess will be partly based on information provided by the sensory system, $\rho(x|\mathbf{d})$. This PDF can be understood as the sensory system's assignment of the probability that the object is at each location, x, given the noisy measurements, $\mathbf{d}$. Let us suppose for this example that, under certain conditions, this guess leads to a bimodal distribution that equally emphasizes two different locations (see figure 9.1).

So, given only information from the sensory system, the final guess, captured by $\rho(y)$, would be the same as the best guess from the sensory system, since $\rho(x|\mathbf{d})$ incorporates all of the information available about the position of the object. However, if there is also a top-down 'model' of what positions to expect, then using information from that model

6 One of the most striking is our inability to see concave faces as anything but convex from about a meter or more away.

may bias $\rho(y)$ towards one of the two modes in $\rho(x|\mathbf{d})$. We can again consider such top-down information as a PDF, $\rho(z|\mathbf{m})$. This PDF captures the biased *a priori* information about how likely the object is to be at any given location in space, z, before any new data, $\mathbf{d}$, is known. This PDF is determined by the parameters, $\mathbf{m}$, which we can think of as summarizing past experience with object positions. The information in $\rho(z|\mathbf{m})$ can thus be used to disambiguate the information available solely from the sensory system (see figure 9.1a; Barber 1999).

We have now set up a toy problem that includes bottom-up information, $\rho(x|\mathbf{d})$, top-down information, $\rho(z|\mathbf{m})$, and a final guess about the location of some external object, $\rho(y|\mathbf{d}, \mathbf{m})$.In order to determine what final guess should be determined for some particular $\rho(x|\mathbf{d})$ and $\rho(z|\mathbf{m})$, we can look at the relevant joint distribution, $\rho(x, y, z|\mathbf{d}, \mathbf{m})$, which captures *all* of the relations between $\rho(x|\mathbf{d})$, $\rho(y|\mathbf{d}, \mathbf{m})$, and $\rho(z|\mathbf{m})$.

Given how we have set up the example, we assume that the input data, $\mathbf{d}$, and the internal model parameters, $\mathbf{m}$, are independent, which means that $\rho(x, z|\mathbf{d}, \mathbf{m}) = \rho(x|\mathbf{d})\rho(z|\mathbf{m})$. Therefore, we can express $\rho(y|\mathbf{d}, \mathbf{m})$ just as we did in the more general case in (9.11) as

$$\rho(y|\mathbf{d}, \mathbf{m}) = \iint \rho(y|x, z)\rho(x|\mathbf{d})\rho(z|\mathbf{m})\, dx\, dz. \tag{9.14}$$

From this expression it is clear that we need to know, $\rho(y|x, z)$, the conditional probability that the object is in some location given the top-down and bottom-up information. If there was reason to think that either the top-down or bottom-up information was more reliable, we could use our definition of this PDF to capture that property of the system. In the absence of any such reason, as in this case, we are free to choose this conditional probability. So, we presume that $\rho(y|x, z)$ is

$$\rho(y|x, z) = \frac{1}{\sqrt{2\pi(\alpha(x-z)^2 + \beta)}} e^{-(y-\frac{1}{2}(x+z))^2/\left(\alpha(x-z)^2+\beta\right)}. \tag{9.15}$$

This conditional emphasizes those places in the multi-modal distribution (i.e., $\rho(x|\mathbf{d})$) where the data and model agree, and de-emphasizes those places where they significantly disagree. Notably, this conditional would not work very well with Gaussian statistics, but we have more freedom to choose the conditional when working with general PDFs.

We have now expressed a high-level formulation of the problem of how we should use previous knowledge to interpret ambiguous sensory data. Recall that this is the same as describing context-sensitive reasoning, where the context is captured by the top-down information, or combining independent sensory inputs, $\rho(x|\mathbf{d}_1)$ and $\rho(z|\mathbf{d}_2)$. The network to implement this high-level formulation can be found as previously described for the more general case (see section 9.1).

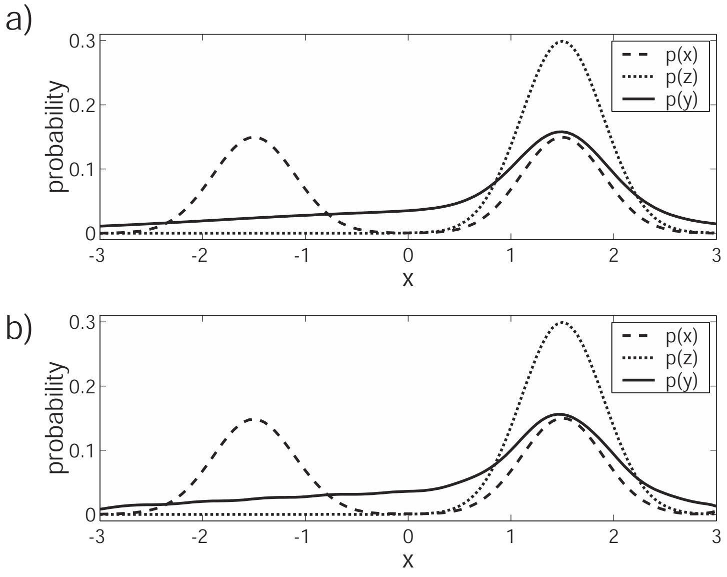


Figure 9.1
Basic statistical inference for position disambiguation. The estimate given the data, $\rho(x|\mathbf{d})$, the estimate given the model, $\rho(z|\mathbf{m})$, and final guess, $\rho(y|\mathbf{d},\mathbf{m})$, for the position of an object are shown in both graphs. a) The solution found by solving equation (9.14) directly. b) The solution found using a network of 200 LIF neurons in each of the three populations. The latter is an imperfect, but very good reproduction of the first.

As shown in figure 9.1, simulating this network shows that it can be used to disambiguate the bottom-up sensory information given a top-down model of where the object is expected to be under these circumstances, as expected.

This model does not include the dynamics we would expect to be involved both in integrating bottom-up and top-down information, and in determining the model parameters, $\mathbf{m}$. We consider this improvement in the next example.

9.3 An example: Parameter estimation

The previous example assumed that there had already been a reasonably good top-down model, parameterized by $\mathbf{m}$, constructed before we attempted to disambiguate the bottom-up information. Of course, the question arises as to where such a model comes from. Ideally, we want a model that is generated based on the statistics of the environment the animal finds itself in. In other words, if the top-down model is to be of any use, it must reflect the structure of the animal's environment.

In order to generate such a model, we can again use statistical inference. In this section, we present a toy parameter estimation problem to show how a neural system can extract the relevant parameters governing a probability distribution of some property in the environment. Suppose that the system receives a stream of values, x_n, each of which can be thought of as a 'measurement' of the underlying statistical process that is giving rise to these values.[7]

In this example we assume that the set of values $x_1, \ldots, x_N$ are generated by a Gaussian process with a mean $\bar{x}$ and variance σ^2 that are fixed for a finite period of time. The purpose of the neural system is to determine estimates of the mean and variance from the measurements x_n signal. Of course, if the mean and variance were truly static, there would be no need for any kind of dynamic statistical inference mechanism. So, we can think of the mean and variance as slowly changing or as being 'reset' occasionally. A more sophisticated system, such as the Kalman filter, would model the actual dynamics of the mean and variance (see section 9.4).

To begin, let us define the conditional probability of obtaining the value of x given $\bar{x}$ and σ as a Gaussian:

$$\rho(x|\bar{x}, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\bar{x})^2/2\sigma^2}. \tag{9.16}$$

We now assume that the mean, $\bar{x}$, and variance, σ^2, themselves are drawn from a broad distribution, $\rho_0(\bar{x}, \sigma)$, whose precise form is not critical. We can express the marginal for x, $\rho(x)$, as

$$\rho(x) = \iint \rho(x|\bar{x}, \sigma)\rho_0(\bar{x}, \sigma)\, d\bar{x}\, d\sigma, \tag{9.17}$$

which will also be very broad. We now suppose that a pair of values $\bar{x}$ and σ have been drawn from $\rho_0(\bar{x}, \sigma)$ and are used to generate a signal, $x_1, \ldots, x_N$, by a random Gaussian process.

Our goal is thus to design a network that represents a PDF over $\bar{x}$ and σ that starts with the prior, $\rho_0(\bar{x}, \sigma)$, and is updated appropriately as the signal arrives. We can define the representation of such a population as

$$\begin{aligned} b_j(x_1, \ldots, x_n) &= G_j \left[\alpha_j \left\langle \tilde{\phi}_j(\bar{x}, \sigma)\rho(\bar{x}, \sigma|x_1, \ldots, x_n) \right\rangle_{\bar{x}, \sigma} \right] \\ \hat{\rho}(\bar{x}, \sigma|x_1, \ldots x_n) &= \sum_j b_j(x_1, \ldots, x_n)\phi_j(\bar{x}, \sigma). \end{aligned}$$

7 This can easily be thought of as the discretization of a continuous signal $x(t)$. The discretization makes the analysis simpler.

To simplify the notation we subsequently use $b_j(x_n)$ and $\rho(\bar{x}, \sigma|x_n)$ to indicate the relevant encoding and decoding.

Because the measurements of the true signal are subject to uncertainty, we take the signal values, x_n, to be the mean of a Gaussian PDF, whose variance, $\sigma^2_{x^n}$, is determined by the amount of noise in the measurement. Since only the mean, x_n, will change, we write this as the conditional, $\rho(x|x_n)$, that is represented by a neural population, where

$$\begin{aligned} a_i(x_n) &= G_i\left[\alpha_i \left\langle \tilde{\phi}_i(x)\rho(x|x_n)\right\rangle_x + J_i^{bias}\right] \\ \hat{\rho}(x|x_n) &= \sum_i \phi_i(x) a_i(x_n) \end{aligned}$$

define the encoding and decoding.

These two representations can be related through an update rule that tells us how the density $\rho(\bar{x}, \sigma|x_n)$ should be modified given the next data point x_{n+1}; namely,

$$\begin{aligned} \rho(\bar{x}, \sigma|x_1, \ldots x_{n+1}) &= \int \rho(\bar{x}, \sigma|x)\rho(x|x_{n+1})\, dx \\ &= \int \frac{\rho(x|\bar{x}, \sigma)}{\rho(x)}\rho(x|x_{n+1})\, dx\, \rho(\bar{x}, \sigma|x_n) \\ &\approx \int \rho(x|\bar{x}, \sigma)\rho(x|x_{n+1})\, dx\, \rho(\bar{x}, \sigma|x_n), \end{aligned} \tag{9.18}$$

where we assume $\rho(x)$ is very broad and hence approximated as a constant whose value can be computed using (9.17). The coupling weights can be found in the usual fashion:

$$\omega_{jil} = \alpha_j \left\langle \tilde{\phi}_j(\bar{x}, \sigma)\rho(x|\bar{x}, \sigma)\phi_i(x)\phi_l(\bar{x}, \sigma)\right\rangle_{x, \bar{x}, \sigma}. \tag{9.19}$$

We can now write the firing rates of the b_j population at time $n+1$ as

$$b_j(x_{n+1}) = G_j\left[\sum_{il} \omega_{jil} a_i(x_{n+1}) b_l(x_n) + J_j^{bias}\right].$$

Note that this same derivation holds regardless of our explicit restrictions on the shape of $\rho(x|x_n)$ or $\rho(\bar{x}, \sigma|x_n)$. That is, the derivation is general enough such that they need not be Gaussians. This is true despite the Gaussian form for the conditional probability which relates the $\rho(x|x_n)$ representation with the $\rho(\bar{x}, \sigma|x_n)$ representation, i.e., (9.16).

Figure 9.2, shows an example time slice of the results of simulating this network using 100 LIF neurons in each population. Though not evident here, as more data is presented to the network, the estimate of the mean and variance, $\rho(\bar{x}, \sigma|x_n)$, becomes narrower. Eventually, the narrowness of the estimate, that is, the certainty of the best guess regarding

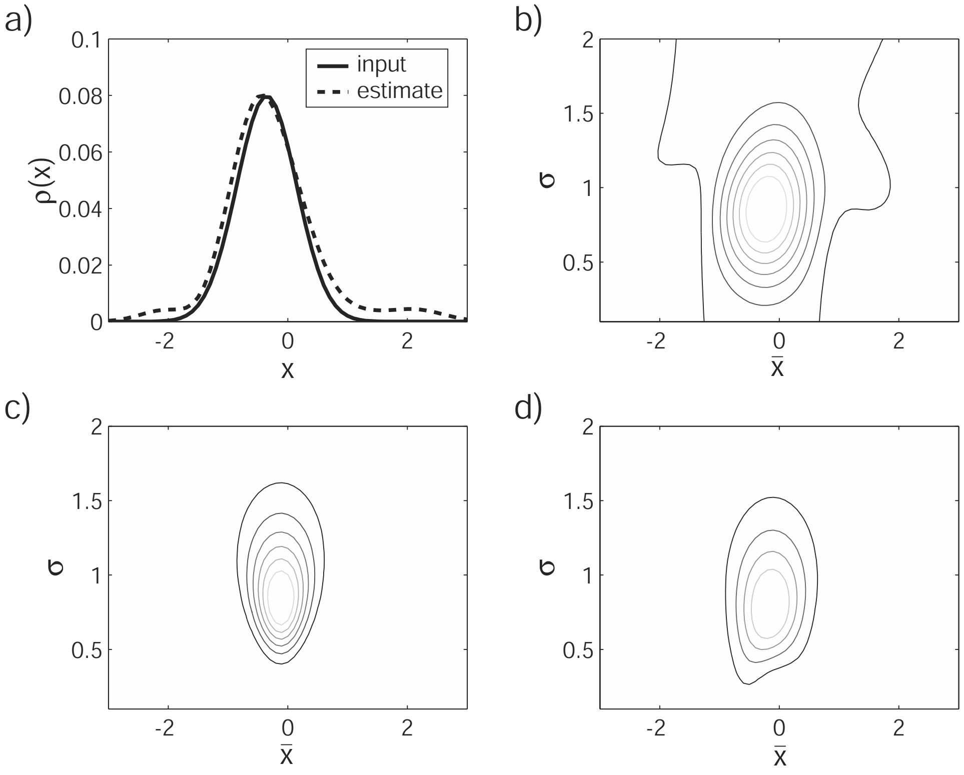


Figure 9.2
Comparison of parameter estimation with and without neural representations. a) The decoded estimate and the original input, $\rho(x|x_n)$. b) The decoded estimate of $\bar{x}$ and σ with 100 LIF neurons in each population, a_i and b_j, shown as a contour plot. c) The estimate of $\bar{x}$ and σ computed using (9.18) directly; d) The estimate of $\bar{x}$ and σ using the representation in b).

the true values of $\bar{x}$ and σ of the underlying process, is limited by the goodness of the neural representation in the b_j population. This again demonstrates the importance of accounting for resource constraints when modeling transformations in neurobiological systems.

There are a number of ways that this model could be made more sophisticated. As mentioned, we assume that the process is stationary (i.e., σ and $\bar{x}$ are not changing over time). There is a well-established theory on how to generate optimal estimates of non-stationary stochastic processes as well (see, e.g., DeWeese and Zador 1998). However, neither the transformations nor the representations are novel, so these more sophisticated models could be implemented using this framework.

However, this is just one of many ways to predict an unknown process under uncertainty. We have taken an essentially Bayesian approach, but such approaches are closely allied to others, like Kalman filtering and information optimization. They all have in common a search for an optimal means of updating a current estimate in light of new information. To make this alliance more explicit, we now turn to an examination of Kalman filtering, an area of study that comes directly from the development of modern control theory.

9.4 An example: Kalman filtering

R. E. Kalman was a central figure in the development of modern control theory. His most influential contribution now bears his name: the Kalman filter (Kalman 1960). The Kalman filter is a least-squares optimal means of estimating the value of unknown state variables in a control system based on a series of noisy and distorted measurements of those state variables; this should be a familiar problem (see section 9.3). The implementation of the Kalman filter is recursive, so only the current measurements and most recent estimate must be stored in order for it to successfully perform. As a result, the filter is constantly updating various filter parameters, i.e., it is dynamic. This kind of filtering is often called *adaptive filtering*.

The flexibility and relative simplicity of the Kalman filter have resulted in its being widely applied by engineers. It has been successfully applied to problems that demand prediction and control of stochastic processes, such as navigation, surveying, tracking, demographic estimation, and image processing (Lev 1997). Many of these problems seem much like those neurobiological systems are interested in solving; i.e., problems that involve determining the underlying state of some process given only noisy measurements of its result. So, perhaps it is not surprising that some recent papers have suggested that parts of the brain act like a Kalman filter. In particular, both visual areas (Rao and Ballard 1997; Rao and Ballard 1999) and the hippocampus (Bousquet et al. 1998), have been compared to, and modeled as, a Kalman filter.

Because the Kalman filter combines ideas from both statistical inference and control theory, and because it has recently been used to make hypotheses about the functioning of various neural systems, it is an appropriate structure to discuss under our framework. Interestingly, using this approach we find that the hypothesis offered by Rao and Ballard (1997) regarding the functional structure of early visual areas is problematic. As a result, we present an alternate hypothesis, but one that is also based on the Kalman filter. In the next section, we briefly discuss two versions of the Kalman filter and how they can be modeled using our approach. We then discuss the implications of each of these versions for understanding early visual areas.

9.4.1 Two versions of the Kalman filter

Essentially, the Kalman filter relies on two independent sets of information to determine the best estimate of some unknown quantity at each moment in time. One set of information, $\mathbf{y}$, comes from the current, noisy measurement of the output of the system with the state variables, $\mathbf{x}$. The second set comes from the current estimate of those state variables, $\hat{\mathbf{x}}$. The unknown quantity to be determined is the value of the state variables at the next instant in time $\hat{\mathbf{x}}(t + \Delta t)$.

The Kalman filter that we consider here is the simple case in which the variance of both the internal model and the measurements is assumed to be static. Usually, the dynamics of these variables are described separately because the internal model will be a poor one at startup (i.e., have large variance), which improves significantly over time. However, we are considering only the steady state case.

We now assume, as is usually done, that the quantities of interest are related by linear systems under Gaussian noise. As a result, the measurements are given by

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \eta_y,$$

where $\mathbf{C}$ describes the constant distortion introduced to the state variables, $\mathbf{x}(t)$, and η_y describes the noise. The evolution of the state variables is given by

$$\mathbf{x}(t + \Delta t) = \mathbf{A}\mathbf{x}(t) + \eta_x,$$

where $\mathbf{A}$ is the internal dynamics matrix (assumed to match that of the system generating $\mathbf{y}(t)$), and η_x describes the noise.[8] This noise term is used to take into account both the Gaussian noise and any differences that might exist between the internal model and the true dynamics of the system.

Given the current estimate of the state variables $\hat{\mathbf{x}}(t)$, and a new set of measurements $\mathbf{y}(t + \Delta t)$, the estimate of the state variables at the next time step is[9]

$$\hat{\mathbf{x}}(t + \Delta t) = \mathbf{P}^{-1}\left[\frac{\mathbf{A}\hat{\mathbf{x}}(t)}{\sigma_x^2} + \frac{\mathbf{C}\mathbf{y}(t + \Delta t)}{\sigma_y^2}\right], \tag{9.20}$$

where

$$\mathbf{P} = \left[\frac{1}{\sigma_x^2} + \frac{\mathbf{C}^2}{\sigma_y^2}\right].$$

8 For simplicity, we have left out the input term, $\mathbf{Bu}(t)$, though it can again be included with minor modification to the remaining derivation.

9 Assume $\rho(\mathbf{x}_{n+1}|\mathbf{y}, \mathbf{x}_n) \approx e^{\left(-\frac{(\mathbf{y}-\mathbf{C}\mathbf{x}_{n+1})^2}{\sigma_y^2} - \frac{(\mathbf{x}_{n+1}-\mathbf{A}\mathbf{x}_n)^2}{\sigma_x^2}\right)}$ where $\mathbf{x}_{n+1}$ indicates $\mathbf{x}(t + \Delta t)$. The maximum likelihood for $\mathbf{x}_{n+1}$ is found by differentiating this equation and setting it to zero, which leads to this result.

The expression in (9.20) is a simple Kalman filter. Note that this is a weighted average of the two estimates of $\mathbf{x}$, where the weight is proportional to their reliability (as determined by the relevant variance). More generally, the terms in this expression consist of the appropriate correlation matrices describing the correlations between each element of the state and measurement vectors. Again, we have simplified from the general case for clarity, but the remaining derivation can proceed without this simplification.

This expression may not be familiar, as the Kalman filter is usually written in the following 'prediction-correction' form

$$\hat{\mathbf{x}}(t+\Delta t) = \mathbf{A}\hat{\mathbf{x}}(t) + \frac{\mathbf{P}^{-1}\mathbf{C}}{\sigma_y^2}\left[\mathbf{y}(t+\Delta t) - \hat{\mathbf{y}}(t+\Delta t)\right], \tag{9.21}$$

where

$$\hat{\mathbf{y}}(t+\Delta t) = \mathbf{C}\mathbf{A}\hat{\mathbf{x}}(t).$$

As some algebraic manipulation shows, these two forms of the Kalman filter are equivalent. Figure 9.3 shows systems diagrams for these two different implementations.

Because our means of translating standard control structures to neural ones depends on a continuous time description, we need to convert (9.20) and (9.21) to the continuous domain. Doing so gives

$$\frac{d\hat{\mathbf{x}}(t)}{dt} = \mathbf{P}^{-1}\left[\frac{\mathbf{A}}{\sigma_x^2}\hat{\mathbf{x}}(t) + \frac{\mathbf{C}}{\sigma_y^2}\mathbf{y}(t)\right], \tag{9.22}$$

and

$$\frac{d\hat{\mathbf{x}}(t)}{dt} = \mathbf{A}\hat{\mathbf{x}}(t) + \frac{\mathbf{P}^{-1}\mathbf{C}}{\sigma_y^2}\Delta\mathbf{y}(t), \tag{9.23}$$

where $\Delta\mathbf{y}(t) = \mathbf{y}(t) - \hat{\mathbf{y}}(t)$.

To construct a neural implementation of these dynamics, we first need to define the representation of the relevant variables. For both versions of the filter we need to represent $\mathbf{x}$ and $\mathbf{y}$:

$$\begin{aligned} a_i(\mathbf{x}) &= G_i\left[\alpha_i\left\langle\tilde{\phi}_i^{\mathbf{x}}\mathbf{x}\right\rangle_n + J_i^{bias}\right] \\ \mathbf{x} &= \sum_i \phi_i^{\mathbf{x}} a_i(\mathbf{x}), \end{aligned}$$

and

$$\begin{aligned} b_j(\mathbf{y}) &= G_j\left[\alpha_j\left\langle\tilde{\phi}_j\mathbf{y}\right\rangle_n + J_j^{bias}\right] \\ \mathbf{y} &= \sum_j \phi_j^{\mathbf{y}} b_j(\mathbf{y}). \end{aligned}$$

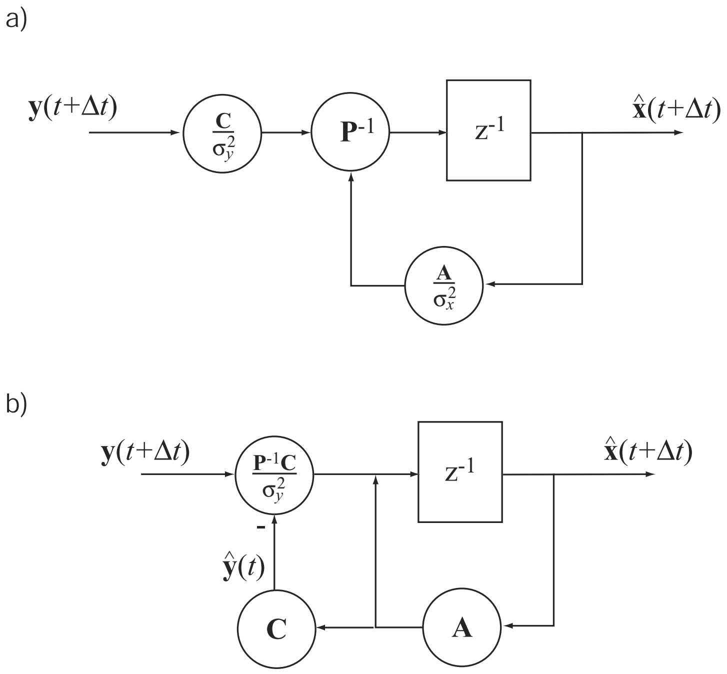


Figure 9.3
Systems diagrams for two ways of implementing a Kalman filter. a) A less familiar form of the Kalman filter (see equation (9.22)). b) The standard prediction-correction form of the Kalman filter (see equation (9.23)). The z^{-1} indicates accumulation, i.e., discrete-time integration.

In addition, for the prediction-correction version we also need to represent $\Delta\mathbf{y}$:

$$\begin{aligned} c_k(\Delta\mathbf{y}) &= G_k\left[\alpha_k\left\langle\tilde{\phi}_k^{\Delta\mathbf{y}}\Delta\mathbf{y}\right\rangle_n + J_k^{bias}\right] \\ \Delta\mathbf{y} &= \sum_k \phi_k^{\Delta\mathbf{y}} c_k(\Delta\mathbf{y}). \end{aligned}$$

Considering first the version, (9.22), we can write the soma currents, $J_i(t)$, of the $a_i(\mathbf{x})$ neurons as

$$\begin{aligned} J_i(t) &= \alpha_i\left\langle\tilde{\phi}_i^{\mathbf{x}}\mathbf{P}^{-1}\left[\frac{\mathbf{A}}{\sigma_x^2}\hat{\mathbf{x}}(t) + \frac{\mathbf{C}}{\sigma_y^2}\mathbf{y}(t)\right]\right\rangle_n + J_i^{bias} \\ &= \sum_{i'}\omega_{ii'}a_{i'}(\mathbf{x}) + \sum_j \omega_{ij}b_j(t) + J_i^{bias}, \end{aligned}$$

where the recurrent coupling weights are $\omega_{ii'} = \left\langle \alpha_i \tilde{\phi}_i^{\mathbf{x}} \mathbf{P}^{-1} \frac{\mathbf{A}}{\sigma_x^2} \phi_{i'}^{\mathbf{x}} \right\rangle_n$, and the feed-forward coupling weights are $\omega_{ij} = \left\langle \alpha_i \tilde{\phi}_i^{\mathbf{x}} \mathbf{P}^{-1} \frac{\mathbf{C}}{\sigma_y^2} \phi_j^{\mathbf{y}} \right\rangle_n$. This circuit is thus very similar in structure to the neural integrator plus an input term.

Considering the prediction-correction version captured by (9.23) instead, we would write the soma currents as

$$\begin{aligned} J_i(t) &= \alpha_i \left\langle \tilde{\phi}_i^{\mathbf{x}} \mathbf{A}\hat{\mathbf{x}}(t) + \frac{\mathbf{P}^{-1}\mathbf{C}}{\sigma_y^2} \Delta\mathbf{y}(t) \right\rangle_n + J_i^{bias} \\ &= \sum_{i'} \omega_{ii'} a_{i'}(\mathbf{x}) + \sum_k \omega_{ik} c_k(t) + J_i^{bias}, \end{aligned}$$

where the recurrent coupling weights are $\omega_{ii'} = \left\langle \alpha_i \tilde{\phi}_i^{\mathbf{x}} \mathbf{A} \phi_{i'}^{\mathbf{x}} \right\rangle_n$, and the feed-forward coupling weights are $\omega_{ij} = \left\langle \alpha_i \tilde{\phi}_i^{\mathbf{x}} \mathbf{P}^{-1} \frac{\mathbf{C}}{\sigma_y^2} \phi_k^{\Delta\mathbf{y}} \right\rangle_n$.

9.4.2 Discussion

The prediction-correction form of the Kalman filter has been proposed as a useful way of understanding visual processing (Rao and Ballard 1997; Rao and Ballard 1999). However, there are both general and vision-specific reasons to suppose that the less familiar form described by (9.22) is more suitable for implementation in neurobiological systems.

Generally speaking, the prediction-correction circuit described by (9.23) and shown in figure 9.3b is more complex than the less familiar circuit, having both more interconnections and demanding more representations. More importantly, the prediction-correction circuit is more likely to be unstable under imperfect representation. To see why, suppose that the dynamics matrix is unity (i.e., $\mathbf{A} = \mathbf{I}$), so the input is static. Under these conditions, the recurrent connections, $\omega_{ii'}$, for the $a_i(\mathbf{x})$ population must implement a higher-order circuit that has positive feedback with unity gain. Such a circuit is on the verge of instability as any slight variation that causes the feedback to go higher than unity will cause an exponential increase in the value of the state variables. In contrast, the feedback in the less familiar version is $\frac{\mathbf{P}^{-1}}{\sigma_x^2}$, which is less than unity and hence stable even under similar perturbations. Put another way, the circuit in figure 9.3a only filters the input signal, while the circuit in figure 9.3b is attempting to integrate the difference between the input and the predicted input. Not only is precise integration notoriously difficult to achieve, but taking the difference will make any error relatively more significant in the computed quantity.

Before turning to specific concerns regarding the application of the prediction-correction Kalman filter to understanding the early visual system, let us first describe how the Kalman filter applies to visual processing. To begin, let us presume that the images

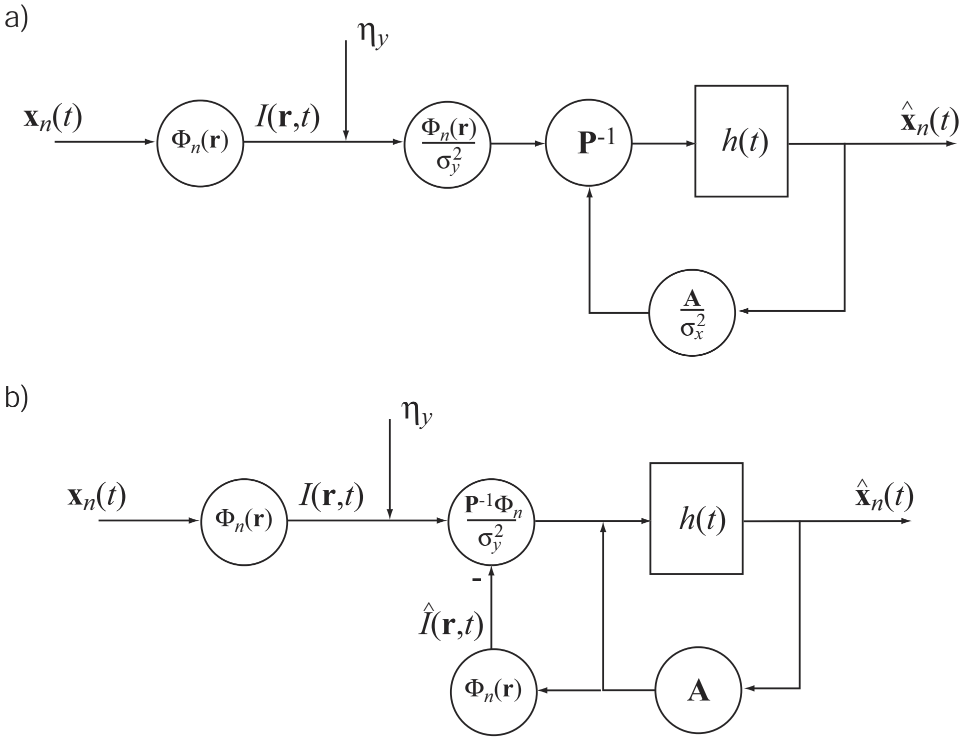


Figure 9.4
Application of the Kalman filter to understanding early visual processing. a) The less familiar version. b) The prediction-correction version. The left side of both diagrams indicates the process whose state variables are being predicted by the filter. The signal η_y is noise. Everything to the right of that signal is presumed to be part of the neurobiological system. The transfer function $h(t)$ incorporates the synaptic dynamics.

falling on the retina, $I(\mathbf{r}, t)$, are generated by some underlying process whose statistics are governed by the well-known inverse frequency (i.e., $1/f$) power spectrum for natural images (Field 1996). We can write the representation of the image as

$$I(\mathbf{r}, t) = \sum_n \mathbf{x}_n(t)\Phi_n(\mathbf{r}),$$

where $\mathbf{r}$ is the retinal coordinate vector and $\Phi_n(\mathbf{r})$ are wavelet-like basis functions (see, e.g., Olshausen and Field 1996). The state variables of this process, $\mathbf{x}_n(t)$, are to be estimated by the Kalman filter. The neurons in the superficial layers of V1 are presumed to be the locus of computation for the estimates of these variables. This description of visual processing is diagrammed for both versions of the Kalman filter in figure 9.4.

This description of visual processing is useful, but more so in the less familiar version than in the prediction-correction version. In addition to the issues of stability discussed earlier, there are two reasons that the less familiar version should be preferred. First, the less familiar version makes it clearer how the context-sensitive reliability of the data, $\mathbf{y}(t)$, and the model, $\hat{\mathbf{x}}(t)$, can be controlled. For instance, if the image is moving rapidly or unpredictably, the evidence generated by the model can quickly become unreliable, so the system should more heavily rely on the data. This would be accounted for by increasing σ_x^2 and decreasing σ_y^2. Given our past discussions, we can be confident that this kind of control can be implemented in a neural system, with good precision. In contrast, the weighting of the model evidence in the standard form is buried in the top-down estimate of the image, $\hat{I}(\mathbf{r}, t)$, making it difficult to vary independently. Furthermore, as mentioned earlier, any errors in this reconstruction will be amplified when taking the difference between it and the actual image data. As a result, it is unclear how to implement an accurate relative weighting of top-down and bottom-up information.

Second, the less familiar form requires minimal feedback that could be accounted for by the recurrent connections within layers 2/3 of the primary visual cortex. In contrast, the standard form requires the predictive feedback from later processing to be fed all the way back to the earliest visual areas, including the thalamic LGN. This latter architecture is proposed by Rao and Ballard (1997) (see especially figure 2 and Rao and Ballard 1999 figure 1a). However, the supposition that a complete reconstruction of the visual image is carried by V1 back-projections to LGN (and back-projections from higher visual areas to V1) is unwarranted. For one, this architecture requires the same channel capacity for the feed-forward and feedback connections just for the image data. Since various control signals and other information are presumably carried along this same channel, this is not a perspicacious use of that resource. As well, ensuring proper timing of the feedback (given the variable nature of neuronal processing and transmission delays) would be an extremely difficult and perhaps insurmountable task. Given the comparative simplicity of the less familiar implementation, supposing that complete reconstructions of $I(\mathbf{r}, t)$ are marshaled around within the visual system is not justified.

9.5 Learning

Both statistical inference and learning use past information to affect future performance. Nevertheless, learning has received much more attention than statistical inference, in a neurobiological context. Presumably, this is because learning, in the form of connection weight changes, is much easier to observe and measure in a neural system than is statistical inference, which does not rely on changing weights. As a result, there are decades old learning rules for neurobiological systems, most famously Hebbian learning, and there continues to be active research into the nature of learning in neural systems (Bienenstock et al. 1982; Abbott and Nelson 2000). In order to make contact with the large body of literature on learning, in this section we discuss the relation between learning and this framework.

At first glance, it is not clear what role there could be for learning given our approach. After all, we always find the connection weights *analytically* (given various assumptions about encoding, etc.). Since the purpose of learning rules is to find these same weights, what role or relevance can learning have? We think there are at least three ways our approach relates to work on learning in neurobiology: 1) learning can help fine-tune models generated with our approach; 2) we can compare and contrast learned with analytically found weights to see if they are the same or different, possibly giving us insight into the nature of learning; and 3) examining the role of learning helps highlight both new challenges for, and the inherent strengths of, this approach.

As regards 1), the weights found using the techniques presented in the part two of this book are typically based on rate model approximations to spiking neurons. How accurate such approximations are depends on the spiking model being used. While there is significant leeway in the framework for making the rate model approximations more or less accurate, there will always be some degree of inaccuracy in the approximation. Thus, the connection weights found using the framework are probably best considered to be good first guesses at a set of weights that perform the desired function. Learning can thus be used to subsequently fine-tune these weights. This would be particularly true in cases like the neural integrator, where there is evidence that an explicit error signal is available to the system for updating subsequent behavior (Robinson 1981).

In the next two sections we consider in more detail the remaining ways that learning can be related to this framework.

9.5.1 Learning a communication channel

One of the simplest networks that we have discussed so far is the communication channel (see section 6.1.1). In this network, one population, $a_i(x)$, drives another population, $b_j(x)$, with no transformation taking place between them. Thus, the purpose of the network is to simply communicate the signal, x, in the original population to the subsequent one; i.e., we want $b_j(x)$ to represent just what $a_i(x)$ does. A natural way to formalize this is to note that maximizing the variance of the receiving population allows it to carry the most information it can about incoming signals. If it carries exactly the same information as is available from the sending population, we have a communication channel. To maximize the variance, we need to maximize

$$E = \frac{1}{2}\sum_j \left\langle \left(b_j(x) - \bar{b}_j(x)\right)^2 \right\rangle_x . \tag{9.24}$$

Noting that $b_j(x) = G_j\left[\sum_i \omega_{ji} a_i(x) + J_j^{bias}\right]$ as usual, and taking the derivative of (9.24) with respect to ω_{ji} gives

$$\frac{dE}{\delta\omega_{ji}} = \frac{dG_j\left[\xi\right]}{d\xi} a_i(x)\left(b_j(x) - \bar{b}_j(x)\right),$$

where $\xi = \sum_i \omega_{ji} a_i(x) + J_j^{bias}$. The G_j term in this expression indicates that we need to take into account the rate of change of the activity of the neuron for input x. Rather than performing this derivative explicitly, however, we can replace it with the simpler term $(b_j(x) > 0)$ which evaluates to 1 when the neuron is active and 0 when it is not. This does not affect the final results of learning, although it can slightly slow down the learning process. The mean of the $b_j(x)$ activity, $\bar{b}_j(x)$, can be found by keeping a running mean, i.e., $\bar{b}_j(x; t + dt) = (1 - \epsilon)\bar{b}_j(x; t) + \epsilon b_j(x; t)$. We can now use the standard 'delta rule' approach to write the learning rule explicitly:

$$\begin{aligned} \Delta\omega_{ij} &= -\kappa \frac{dE}{\delta\omega_{ji}} \qquad (9.25) \\ &= -\kappa\left(b_j(x) > 0\right) a_i(x)\left(b_j(x) - \bar{b}_j(x)\right), \qquad (9.26) \end{aligned}$$

where κ is the learning rate. Note that this is a typical Hebbian learning rule. That is, it is a *local* learning rule that will construct the expected representation in the b_j population.

Running this simulation with 200 LIF neurons in each population and starting with random coupling weights, we see that the $b_j(x)$ population does indeed learn to represent the signal in $a_i(x)$ quite well (see figure 9.5).

Given these weights and our previous analyses, it is natural to ask if we can decompose them into their encoding and decoding components. Recall that the general form of the weights we find is

$$\omega_{ji} = \tilde{\phi}_i \phi_j$$

or, in matrix form,

$$\boldsymbol{\omega} = \tilde{\boldsymbol{\phi}}\boldsymbol{\phi}.$$

In the communication channel, the encoding weights are simply ± 1. Given the learned weights, $\boldsymbol{\omega}_{learn}$, and the encoding weight vector, $\tilde{\boldsymbol{\phi}}$, we can try to find the decoders, $\boldsymbol{\phi}$. However, because the encoding weight vector is not a square matrix, we must take the

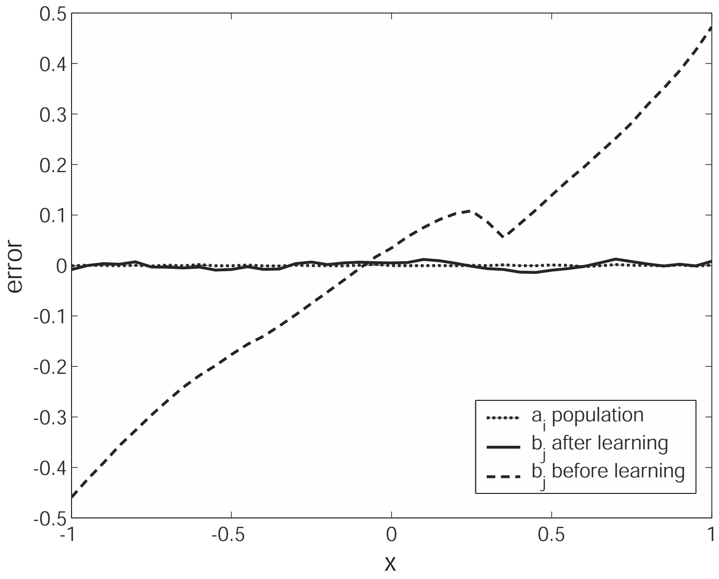


Figure 9.5
Learning to represent the signal in $a_i(x)$. The error in the $b_j(x)$ population representation of x given random weights is quite high, whereas the error after learning is comparable to that in the original population. The signal used to train the network was white noise over the range of x.

pseudo-inverse (that is, use SVD) to find the decoders. As a result, the weight matrix we reconstruct using these extracted decoders, $\boldsymbol{\omega}_{recon}$, is different than the original learned matrix (compare figures 9.6a and 9.6c).

However, the reconstructed matrix is equally good at preserving the information in the incoming signal (see figure 9.6d). This suggests that performing this kind of decomposition may be useful for analyzing the often mysterious results of employing a learning rule in a given network. In other words, using this framework makes it possible to take a set of weights generated in any particular way and then derive a set of decoders given encoding assumptions. These decoders can then be used to determine what function those weights compute (given the encoding assumptions). So, we can use this approach to make a principled guess at the function of a set of learned weights.

Furthermore, we can compare these matrices based on learning with the matrix we originally found in section 6.1.1. This matrix, $\boldsymbol{\omega}_{analytic}$, was found analytically using now familiar techniques (see figure 6.1.1b). While this matrix is again different from either the learned or reconstructed matrix, it too does an excellent job (slightly better, in fact) of preserving the original representation (see figure 9.6d). The fact that these three matrices all

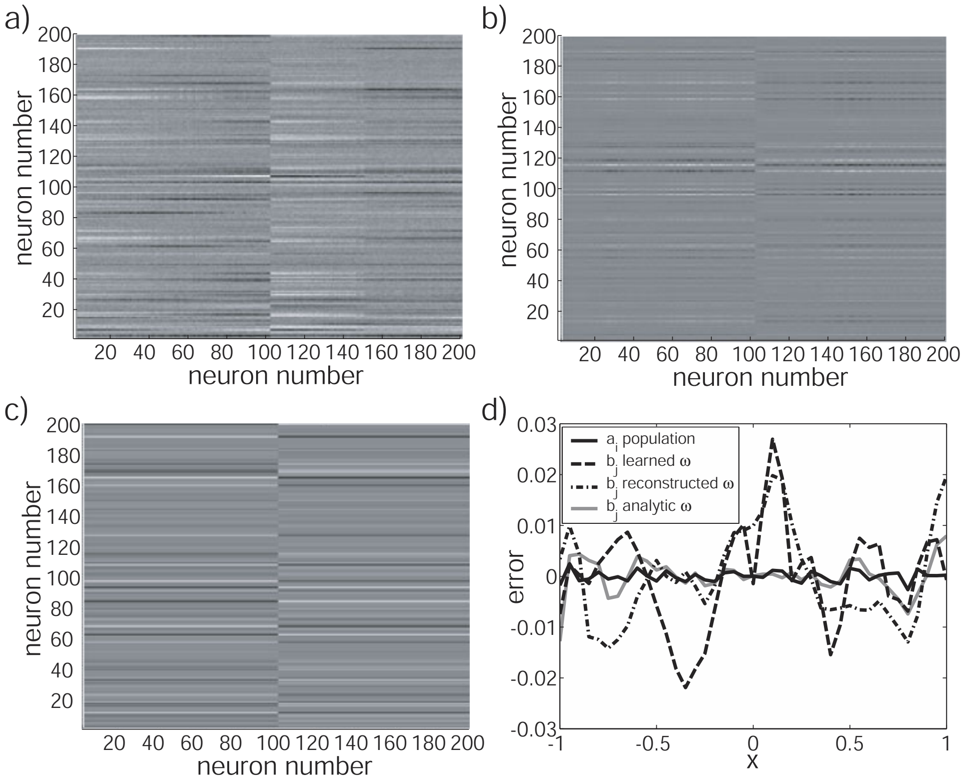


Figure 9.6
Comparison of weights and linear decoding errors. a) Shows the learned weights using the rule in (9.26), $\boldsymbol{\omega}_{learn}$. b) Shows the analytically determined weights from our previous characterization of the communication channel, $\boldsymbol{\omega}_{analytic}$ (see section 6.1.1). c) Shows the reconstruction of the weight matrix using the decoders found by decomposing the learned weight matrix, $\boldsymbol{\omega}_{recon}$. d) Shows the linear decoding errors for each of these weight matrices, as well as that for the original a_i population. Note that the errors are all of similar magnitude, with the a_i and analytic weights slightly better. By comparing this graph to that in figure 9.5 it is clear that all of these matrices are better than random weights (note the change in scale). As well, comparison of a)-c) makes clear that these connection weight matrices share a certain amount of structure, as discussed in the text.

perform similarly emphasizes that there are many solutions to problems like constructing a communication channel, given the high-dimensionality of weight space. However, looking at the commonalities amongst these three solutions also provides us with some clues as to what properties of the weights may be essential for a good communication channel. In particular, all of these approaches find a solution that divides the population about equally into ‘on’ and ‘off’ neurons. And, the pattern of connectivity is similar in that oppositely tuned neurons tend to have negative weights and the strength of positive weights depends on the similarity of the neurons' tuning curves. So, there is a general structure to these matrices that becomes apparent and understandable once we have a variety of tools for both analyzing and synthesizing weight matrices.

9.5.2 Learning from learning

The literature on learning in neural systems is enormous. The considerations in section 9.5.1 merely scratch the surface of the subject. Nevertheless, we can learn a number of important lessons; both supportive of, and challenging for, this framework. First, let us consider some of the inherent strengths highlighted by this discussion. Because this framework is essentially analytic, it can help us gain new insights regarding standard learning rules. We can apply such rules and then analyze the resulting weight matrices in a way that may give us a new or better functional characterization, as in the case of the communication channel. In addition, given the representational hierarchy, derived from our unified approach to understanding various kinds of representation, we can be confident that such rules and analyses will generalize to systems trafficking in more complex representations. Presumably, this is true both for analyzing the static transformations in a network, as we have done above, and for learning dynamic transformations. For instance, to learn an attractor, regardless of the complexity of the representation, we should be able to minimize the energy that enforces $\dot{J}_i = 0$, i.e.,

$$E_i = \frac{1}{2}\left\langle \left[J_i(t) - \langle J_i(t)\rangle_t\right]^2 \right\rangle_t,$$

though this remains to be seen. Furthermore, because we have a general means of incorporating input signals and transformations via control theory, accounting for learning in systems with explicit error signals, like the neural integrator, should be straightforward. Finally, and most importantly, this approach gives us a means of constructing complex, functional systems, *independent* of learning. This is essential for modeling complex systems because, at the moment, it is not clear how to learn many of the complex behaviors exhibited by neurobiological systems. Such considerations do not make this approach a competitor to a learning-based approach, but rather show ways in which both can contribute to our understanding of complex neural systems.

These same considerations raise the first challenge to our framework: can we generate learning rules that give rise to the systems we construct analytically? The superficial answer is 'yes', as the communication channel shows. But, for more complex transformations and more complex control structures, it is not clear what the answer will be. Notice that minimizing the variance between two populations, as we did in section 9.5.1, only constrains one degree of freedom in the network's representations. However, if we want to learn arbitrary transformations, we will need to constrain multiple degrees of freedom. To see why, suppose that we have a population, a_i, from which we can encode the lower-order polynomials, x^n. We can thus find decoders, $\phi_i^{(x^n)}$, such that

$$\hat{x}^n = \sum_i a_i(x)\phi_i^{(x^n)}.$$

Suppose we want to compute some function, $f(x)$, such that

$$f(x) = \sum_m^M A_m x^n.$$

We know we can compute this function in some b_j population using the connection weights $\omega_{ji} = \tilde{\phi}_j \sum_m A_m \phi_i^{(x^n)}$ as we have in the past.

But notice that these weights are now a function of the coefficients A_m. This means that any learning rule must be able to adjust the weights by systemmatically varying these M degrees of freedom *independently*. Furthermore, there needs to be a mechanism such that the desired values for the A_m are applied consistently across the population. But, to remain biologically plausible we have to impose these constraints using *local* learning rules. Applying such global constraints does not mean local rules are impossible to derive, just that they need to be highly sophisticated. In particular, they need to be controlled by a set of M-dimensional signals in a neurobiologically plausible way. These important challenges currently remain unresolved.

A second important issue that learning highlights is that of the ambiguity of connection weights. Because our framework analyzes weights as the projection between encoding and decoding vectors, and because projections of different sets of encoding and decoding vectors can result in the same matrix, there is no isolated ‘right’ decomposition of a weight matrix. This ambiguity can only be overcome in the context of a larger understanding of the system in which the weights are found. As in the case of the communication channel, finding the decoders for a system depends on our assumptions about the encoders. In order for these assumptions to be reasonable ones, and constrained in some way, they need to be made in the context of a larger system. So, just as justifying claims about what is represented by a system depend on the coherence of our overall story about brain function (see section 1.2), so does justifying claims about what functions the weights instantiate. This parallel should not be surprising since representations and transformations are characterized in similar ways. The challenge lies in determining methods that are better able to extract all of the structure of a weight matrix consistent with this approach. While the kind of analysis performed in section 9.5.1 is a beginning, figure 9.6 suggests that alternate analyses may be better.

A final challenge lies in trying to find ways to use the framework itself to generate novel learning rules. While this approach seems useful for analyzing the results of typical learning rules (like the Hebbian rule), it would be very useful to be able to have a means of determining plausible learning rules that would give rise to the kinds of high-level structure that the framework can build into models. It is a challenge, in other words, to not only build complex models like those we have discussed in the book, but to also explain how they may have come about given the kinds of synaptic changes observed in neurobiological systems. In general, the preceding challenges can be summarized by noting that we have not provided a systematic way to relate learning to this approach. Admittedly we have not explored learning in much detail, and thus it remains an intriguing avenue for future study.

9.6 Summary

We began this chapter by noting that, unlike previous chapters, it would be rife with speculation and simple examples. We began by arguing that, in general, statistical inference is essential for explaining the behavior of animals in a noisy, uncertain world. We then outlined some notable affinities that neurobiological systems have for being able to implement just the kinds of transformations needed to support complex statistical inference.

We demonstrated how our approach could account for the relevant kinds of transformations by considering statistical inference in three contexts. First, we presented a simple example of the integration of top-down and bottom up information. Second, we demonstrated how the top-down model in the first example could be dynamically generated on the basis of uncertain data. Third, we discussed a similar dynamic model, the Kalman filter, which was derived in the context of control theory and is thus a natural one to apply using our framework. Furthermore, the Kalman filter has recently received attention in a neurobiological context. We examined one such context, its application to the visual system, and described how a less common version of the filter is more biologically plausible that the more common prediction-correction version.

We then turned to the closely related topic of learning, and presented some preliminary analyses that relate learning to this framework. We showed how, for a simple communication channel, we could analyze the results of a typical Hebbian learning rule. While we suggested that being able to construct and analyze weights using this framework could be useful in understanding such learning rules, we also suggested that it remains unclear if there can be a more principled relation between learning and this approach. We concluded that integrating learning and this approach is an area ripe for future study.

A Appendix: Chapter 2 derivations

A.1 Determining optimal decoding weights

In section 2.1.2, we discuss the importance of being able to solve for the optimal decoding weights given the error function

$$E = \frac{1}{2}\int_{-1}^{1}\left[x - \sum_i^N a_i(x)\phi_i\right]^2 dx. \tag{A.1}$$

To do so, we can take the derivative of (A.1) with respect to ϕ_i as follows:

$$\begin{aligned} \frac{\delta E}{\delta \phi_i} &= -\frac{1}{2}\int_{-1}^{1} 2\left[x - \sum_j^N a_j(x)\phi_j\right] a_i(x)\,dx \\ &= -\int_{-1}^{1} a_i(x)x\,dx + \int_{-1}^{1}\sum_j^N a_i(x)a_j(x)\phi_j\,dx. \end{aligned} \tag{A.2}$$

Setting the derivative to zero and finding the minimum over all i gives

$$\int_{-1}^{1} a_i(x)x\,dx = \sum_j^N \left(\int_{-1}^{1} a_i(x)a_j(x)\,dx\right)\phi_j, \tag{A.3}$$

which can be written using matrix-vector notation as

$$\Upsilon = \mathbf{\Gamma}\boldsymbol{\phi}. \tag{A.4}$$

Solving for $\boldsymbol{\phi}$ gives

$$\boldsymbol{\phi} = \mathbf{\Gamma}^{-1}\Upsilon, \tag{A.5}$$

where

$$\begin{aligned} \Gamma_{ij} &= \langle a_i(x)a_j(x)\rangle_x \\ \Upsilon_i &= \langle x a_i(x)\rangle_x . \end{aligned}$$

As mentioned in the text, the notation $\langle\cdot\rangle_x$ indicates integration over the range of x. Or, equivalently, we can write

$$\phi_i = \sum_j^N \Gamma_{ij}^{-1}\Upsilon_j.$$

This page intentionally left blank

B Appendix: Chapter 4 derivations

B.1 Opponency and linearity

To get a sense of why the opponency arrangement provides for more linear responses, consider the task of trying to approximate a linear input/output function with nonlinear neuron response functions. Let $a(x)$ be a standard LIF response function. We can write down the Taylor series expansion of this function as follows:

$$a(x) = a(0) + xa'(0) + \frac{x^2 a''(0)}{2!} + \ldots .$$

The error of this nonlinear approximation to the linear input/output function will be equal to the sum of all the nonlinear terms in this series (i.e., everything other than $a(0)$). Now suppose that we attempt to approximate the same linear function with two 'mirror image' LIF neurons, $a(x)$ and $a(-x)$. The Taylor expansion of $a(-x)$ is

$$a(-x) = a(0) - xa'(0) + \frac{x^2 a''(0)}{2!} - \ldots .$$

If we take the approximation of our linear function to be the difference between these two nonlinear neurons, then the error in the approximation will be equal to the Taylor series terms remaining in that difference:

$$a(x) - a(-x) = 2xa'(0) + \ldots .$$

Notice that all of the even distortion terms cancel and that only the odd ones remain. If we attempted to approximate the linear response function with two complimentary neurons (i.e., neurons whose firing rate both increase in x), this cancellation would not occur. Thus the opponency approximation is significantly more linear than either two complimentary neurons or one of the opponency neurons on its own. This is because the distortions of the opponency neurons act to cancel each other out only when they are used together.

B.2 Leaky integrate-and-fire model derivations

Some useful insights can be gained into what the LIF model is doing by considering a means of solving equation (B.1):

$$\frac{dV(t)}{dt} = -\frac{1}{\tau^{RC}} \left(V(t) - J_M(t)R \right) . \tag{B.1}$$

Recall that $\tau^{RC} = RC$. This factor is the membrane time constant, and it controls the speed with which the membrane potential approaches the steady state value, $J_M(t)R$ (for constant input). If τ^{RC} is large, the *change* in voltage (i.e., $dV(t)/dt$) will be small since it is proportional to $1/\tau^{RC}$. We assume that the initial conditions are $V(0) = 0$ for simplicity.

To begin, then, we can presume in advance that we want a solution in the form of

$$V(t) = F(t)e^{-t/\tau^{RC}}, \tag{B.2}$$

where $F(t)$ is some arbitrary function of time. We can assume that the solution will take this form because (B.1) is of a familiar form itself. Namely, it consists of a derivative of some function, $V(t)$, equal to that function times a constant (whose solution is typically Ae^{at}), plus some other arbitrary function of time, $J_M(t)R$ (for a more detailed discussion in a biophysical context, see Wilson 1999b, chp. 2).

We can now substitute equation (B.2) into equation (B.1) to obtain

$$\frac{d}{dt}\left(F(t)e^{-t/\tau^{RC}}\right) = -\frac{1}{\tau^{RC}}\left(F(t)e^{-t/\tau^{RC}} - J_M(t)R\right).$$

Applying the product rule gives

$$\frac{dF(t)}{dt}e^{-t/\tau^{RC}} + \frac{de^{-t/\tau^{RC}}}{dt}F(t) = -\frac{1}{\tau^{RC}}F(t)e^{-t/\tau^{RC}} + \frac{1}{\tau^{RC}}J_M(t)R.$$

Knowing that $\frac{de^{at}}{dt} = ae^{at}$, canceling equal terms on both sides, and rearranging gives

$$\frac{dF(t)}{dt} = \frac{J_M(t)R}{\tau^{RC}}e^{t/\tau^{RC}}.$$

Integrating both sides, we obtain

$$F(t) = \frac{R}{\tau^{RC}}\int_0^t e^{t'/\tau^{RC}}J_M(t')dt',$$

where t' is a ‘dummy’ variable that ranges over all times between $t = 0$ and now. We can now substitute this expression for $F(t)$ into equation (B.2), which gives

$$V(t) = \frac{R}{\tau^{RC}}\int_0^t e^{-(t-t')/\tau^{RC}}J_M(t')dt'. \tag{B.3}$$

Equation (B.3) is a standard convolution integral. By examining this integral, we can get a good sense of what the LIF neuron does. In effect, this equation says that the voltage right now (at t) depends on all past current input, $J_M(t')$, where each input is weighted by a function that exponentially decays as that input gets further away (in the past) from the current time. This provides a useful qualitative way to think about the behavior of LIF neurons.

We can now evaluate the integral in equation (B.3), to provide the standard solution to the differential equation describing the LIF neuron. In doing so, we must assume that the input current, J_M, is constant:

$$V(t) = \frac{R}{\tau^{RC}} \int_0^t e^{-(t-t')/\tau^{RC}} J_M dt'.$$

Substituting variables and letting $u = -(t-t')/\tau^{RC}$, we have

$$V(t) = \frac{R}{\tau^{RC}} \int_0^t e^{u} J_M \tau^{RC} du.$$

Evaluating the integral gives

$$\begin{aligned} V(t) &= J_M R e^{-(t-t')/\tau^{RC}} \Big|_0^t \\ &= J_M R \left(1 - e^{-t/\tau^{RC}}\right). \end{aligned}$$

This solution is physiologically unrealistic but it is a reasonable approximation, especially since we do not know a general form for $J_M(t)$. As well, this mirrors quite well how physiologists find the firing rate curves for real neurons (i.e., by introducing a constant pulse current). Fortunately, it is not difficult to implement (B.3) in computer simulations, so the dynamics are better preserved in practise.

B.3 Optimal filter analysis with a sliding window

Our goal in this section is to find the optimal linear filter, $h(t)$, for encoding an ensemble of signals, $x(t;\mathbf{A})$. We begin by forming the error between our linear estimates and the original signals:

$$\begin{aligned} E &= \left\langle [x(t;\mathbf{A}) - \hat{x}(t;\mathbf{A})]^2 \right\rangle_{\mathbf{A},t} && \text{(B.4)} \\ &= \left\langle \left[x(t;\mathbf{A}) - \sum_{i,k} \phi_i h(t - t_{ik}(\mathbf{A})) \right]^2 \right\rangle_{t,\mathbf{A}} && \text{(B.5)} \\ &= \left\langle [x(t;\mathbf{p}) - h(t) * R(t;\mathbf{A})]^2 \right\rangle_{t,\mathbf{A})}, && \text{(B.6)} \end{aligned}$$

where $\phi_i = \pm 1$, and $R(t;\mathbf{A})$ is the combined response of a pair of complimentary neurons. As discussed in section 4.3.3, it is more appropriate to solve this problem in the frequency domain. Thus, by Parseval's theorem we can write

$$E = \left\langle \frac{1}{2\pi} \left| x(\omega; \mathbf{A}) - h(\omega) R(\omega; \mathbf{A}) \right|^2 \right\rangle_{\mathbf{A},\omega}. \tag{B.7}$$

Recalling that our ensemble of signals is written as

$$x(t, \mathbf{A}) = \sum_n A(\omega_n) e^{i\omega_n t}, \tag{B.8}$$

the Fourier transform of the ensemble is

$$\begin{aligned} x(\omega, \mathbf{A}) &= \int x(t, \mathbf{A}) e^{-i\omega t}\, dt && \text{(B.9)} \\ &= \sum_n A(\omega_n) \int e^{-it(\omega - \omega_n)}\, dt && \text{(B.10)} \\ &= \sum_n A(\omega_n) \delta(\omega - \omega_n). && \text{(B.11)} \end{aligned}$$

So we can write the error for each frequency channel as

$$E(\omega_n) = \left\langle \frac{1}{2\pi} \left| A(\omega_n) - h(\omega_n) R(\omega_n; \mathbf{A}) \right|^2 \right\rangle_{\mathbf{A}}. \tag{B.12}$$

As noted in the text, $\langle \cdot \rangle_{\mathbf{A}}$ is a high dimensional integral over all the amplitudes, i.e.,

$$\langle f(\mathbf{A}) \rangle_{\mathbf{A}} = \int\!\!\int \ldots \int f(\mathbf{A}) \prod_n \rho(A(\omega_n)) dA_0 dA_1 \ldots dA_{N/2}. \tag{B.13}$$

There is no analytical solution, so we must instead do a Monte Carlo estimate. Letting α denote particular choices of our amplitudes, $\mathbf{A}$, we write the error as

$$\begin{aligned} E &= \frac{1}{N^\alpha} \sum_\alpha \frac{1}{L} \int_0^L \left[x(t; \mathbf{A}^\alpha) - h(t) * R(t; \mathbf{A}^\alpha) \right]^2 dt && \text{(B.14)} \\ &= \frac{1}{N^\alpha} \sum_\alpha \sum_n \left| A^\alpha(\omega_n) - h(\omega_n) R(\omega_n; \mathbf{A}^\alpha) \right|^2. && \text{(B.15)} \end{aligned}$$

The length of each trial, L, need only be a few times longer than the 'memory' of the neuron, which in the case of the leaky integrate and fire neuron is τ_{RC}. This is because the effects of past voltage states fall off exponentially with time (see section 4.1.2), and so become lost in the noise after a few time constants, τ_{RC}, have passed. Even in more complex cellular models, it is generally possible to identify a reasonable value for L. Whichever time constant (e.g., that of the synaptic conductances, ion channels, etc.)

dominates the overall voltage decay at the soma, L need only be a few times longer than it. This is because trials separated by L can be assumed to be statistically independent.

This kind of analysis has been shown to work well in many experimental preparations. However, running many short term simulations of size L can be expensive and severely hampered by start up transients. For this reason, researchers have developed a means of windowing longer trials of length T with windows of length Δ_t that still allows the extraction of the optimal filter. This idea is discussed in detail by (Rieke et al. 1997). The procedure they suggest uses a square, or box-shaped, window (Rieke et al. 1997, p. 165):

$$W(t-t^\alpha) = \begin{cases} 1, & \text{if } |t-t^\alpha| \leq \Delta_t/2 \\ 0, & \text{otherwise} \end{cases},$$

spaced by some Δ_t to ensure statistically independent samples. Assuming that the process is ergodic, a long enough run will provide sufficient examples such that the estimated error (B.14) will approximate the mean square error under all possible signal conditions.

However, as we discuss shortly, there are some difficulties with using this kind of window. For this reason, we have developed a means of finding the filter using a sliding Gaussian window, $W(t-t^\alpha)$:

$$W(t-t^\alpha) = e^{-(t-t\alpha)/\left(2\Delta_t^2\right)}, \tag{B.16}$$

where $\Delta_t \approx L$. We let t^α take on the all values between 0 and T, since this allows the sum over the examples to be replaced by an integral over t^α. The argument from ergodicity still holds since sliding the window oversamples exactly the same data set. All of the available statistically independent samples in a trial will be taken in an unbiased manner since each is equally oversampled. In the time domain, the averaged windowed error becomes:

$$E = \frac{1}{T^2} \int_0^T \int_0^T [W(t-t^\alpha) * \Delta S(t;\mathbf{A})]^2 \, dt dt^\alpha, \tag{B.17}$$

where

$$\Delta S(t;\mathbf{A}) = S(t;\mathbf{A}) - h(t) * R(t;\mathbf{A}), \tag{B.18}$$

and the $\mathbf{A}$ are the amplitudes generated for the long trial. Technically, the integrals should be taken over the range between $\approx 2\Delta_t$ and $T - 2\Delta_t$, to avoid introducing errors at the end points. However, this introduces unnecessary complexities in the analysis, and, in any case, these errors are small for $T \gg \Delta_t$. We can now show that the error can be written as a convolution of the power of the Fourier transform of the window function $W(t)$ with the power of the Fourier transform of $\Delta S(t; \mathbf{A})$:

$$\begin{aligned}
E(t^\alpha) &= \frac{1}{T}\int_0^T |W(t - t^\alpha)\Delta S(t)|^2 \, dt \\
&= \int |W(\omega; t^\alpha) * \Delta S(\omega)|^2 \, d\omega \\
&= \iiint W(\omega - \omega')e^{it^\alpha(\omega-\omega')}\Delta S(\omega) \\
&\quad W^*(\omega - \omega'')e^{-it^\alpha(\omega-\omega'')}\Delta S^*(\omega)\, d\omega' d\omega'' d\omega \\
&= \iiint e^{it^\alpha(\omega'-\omega'')} W(\omega - \omega')W^*(\omega - \omega'')|\Delta S(\omega)|^2 \, d\omega' d\omega'' d\omega
\end{aligned}$$

Taking the integral over t^α, and using the fact that

$$\frac{1}{T}\int e^{it^\alpha(\omega'-\omega'')}\, dt^\alpha = \delta(\omega' - \omega''),$$

we get

$$\begin{aligned}
E &= \frac{1}{T}\int E(t^\alpha)\, dt^\alpha && \text{(B.19)} \\
&= \iint |W(\omega - \omega')|^2 |\Delta S(\omega; \mathbf{A})|^2 \, d\omega' d\omega. && \text{(B.20)}
\end{aligned}$$

Essentially, (B.20) shows that we can replace averaging over trials in the time domain by a convolution in the frequency domain, which locally averages the extra frequency channels introduced by the long runtime. This smoothing reduces the number of degrees of freedom needed to compute the optimal filter, which is desirable (as we discuss in section 4.3.3).

We can now see what advantage the Gaussian window has over using a box window. Note that the Fourier transform of a box window is a sync function in the frequency domain (which has ringing out to high frequencies). Any loss of these high frequency components is likely to be reflected in our estimate, introducing spurious errors. In order to minimize the effects of these errors, we would have to average over longer run times and have more examples. In contrast, the Fourier transform of a Gaussian is another Gaussian, which is a well-localized low pass filter (guaranteeing a reduction in the number of degrees of freedom without spurious errors). In other words, the continuous Gaussian filtering allows us to make better use of the data available because the estimation process itself does not introduce possible sources of error.

We now take the functional derivative of equation (B.20) with respect to $h^*(\omega)$ to find an expression for the optimal filter:

$$h(\omega) = \frac{\langle A(\omega)R^*(\omega)\rangle_{\mathbf{A}}}{\left\langle |R(\omega;\mathbf{A})|^2 \right\rangle_{\mathbf{A}}}, \tag{B.21}$$

where $\langle\cdot\rangle_{\mathbf{A}}$ is the convolution with the window to emphasize that this providing an approximation to the average over the stochastic variables, $\mathbf{A}$.

The residual error obtained by substituting (B.21) into (B.20) is

$$E_{residual}(\omega) = \left\langle |A(\omega)|^2 \right\rangle_{\mathbf{A}} - \frac{|\langle A(\omega)R^*(\omega)\rangle_{\mathbf{A}}|^2}{\left\langle |R(\omega;\mathbf{A})|^2 \right\rangle_{\mathbf{A}}}. \tag{B.22}$$

This error will be equal to zero only if there are no inter-harmonic distortions generating spurious power at that frequency.

It is worth noting that the filter, $h(\omega)$, not only adjusts for the difference in gain, but also for any shift in phase. In a sense, it automatically adjusts for any propagation delays and phase distortions created by the spike generation process so long as the width of the Gaussian window is large enough to accommodate them. It is again worth emphasizing that the difference between this and previous methods for finding $h(t)$ is the way the average over $\mathbf{A}$ is carried out.

B.4 Information transmission of linear estimators for nonlinear systems

The problem of understanding information transmission through spiking neurons is a specific case of a more general problem. Recall that we are interested in finding an optimal linear estimator, h, that will give us an estimate, $\hat{x}$, of the original signal, x, given the response of the neurons, $R(x)$. Of course, we can let the neuron response in our particular problem be the response of any nonlinear 'black box' system. In this appendix, we derive a general expression for the information transmission through such a nonlinear black box given a linear decoding.

We begin by assuming that the signal is drawn from a Gaussian distribution with a mean of zero and variance σ_x. A linear estimate of the original signal given the response is given by

$$\hat{x} = hR(x), \tag{B.23}$$

where h is the linear decoder. The average error in our reconstruction can be found by taking the difference between our estimate and the original signal over all possible signals, which gives the mean squared error (MSE)

$$MSE = \left\langle [x - \hat{x}]^2 \right\rangle_x = \left\langle [x - hR(x)]^2 \right\rangle_x . \tag{B.24}$$

In order to find the *optimal* decoder, we want to minimize this error, and thus set the derivative of (B.24) to zero and solve for h. This gives

$$h = \frac{\langle x \cdot R(x) \rangle_x}{\langle R^2(x) \rangle_x}. \tag{B.25}$$

We can now re-write the error in terms of the optimal decoder:

$$\begin{aligned} MSE &= \left\langle x^2 \right\rangle_x - h^2 \left\langle R^2(x) \right\rangle_x \\ &= \left\langle x^2 \right\rangle_x - \frac{\langle x \cdot R(x) \rangle_x^2}{\langle R^2(x) \rangle_x}. \end{aligned} \tag{B.26}$$

When we use the optimal decoder, our expression for the the MSE captures the error that is introduced simply by the fact that we are trying to decode a nonlinear system with a linear decoder. In a sense, this is error that we cannot explain, i.e., the error that we cannot remove from our chosen estimation process, no matter what linear decoder we use.

In order to determine the information that is transmitted under this decoding, we can use the following standard expression for the amount of information transmitted by a channel:

$$\text{Info} = \frac{1}{2} \log_2(1 + SNR), \tag{B.27}$$

where SNR is the signal-to-noise ratio of the channel. In our case, we do not have any noise, but we do have an analogous quantity, the unexplained variance. This is the variance that comes from our unexplained error, i.e., $\langle MSE \rangle_x$. The signal, or explained variance, is the variance of our estimate:

$$\begin{aligned} \left\langle \hat{x}^2 \right\rangle_x &= h^2 \left\langle R^2(x) \right\rangle_x \\ &= \frac{\langle x \cdot R(x) \rangle_x^2}{\langle R^2(x) \rangle_x}. \end{aligned} \tag{B.28}$$

Thus, the SNR is given by

$$\begin{aligned} SNR &= \frac{\left\langle \hat{x}^2 \right\rangle_x}{\langle MSE \rangle_x} \\ &= \frac{\langle x \cdot R(x) \rangle_x^2}{\langle x^2 \rangle_x \langle R^2(x) \rangle_x - \langle x \cdot R(x) \rangle_x^2}. \end{aligned} \tag{B.29}$$

So, we can write the $(1 + SNR)$ factor in equation (B.27) in several different ways:

$$(1+SNR) = \frac{\left\langle x^2\right\rangle_x \left\langle R^2(x)\right\rangle_x}{\langle x^2\rangle_x \langle R^2(x)\rangle_x - \langle x \cdot R(x)\rangle_x^2} \tag{B.30}$$

$$= \frac{\left\langle x^2\right\rangle_x}{\langle x^2\rangle_x - \frac{\langle x\cdot R(x)\rangle_x^2}{\langle R^2(x)\rangle_x}} \tag{B.31}$$

$$= \frac{\left\langle R^2(x)\right\rangle_x}{\langle R^2(x)\rangle_x - \frac{\langle x\cdot R(x)\rangle_x^2}{\langle x^2\rangle_x}}. \tag{B.32}$$

Since $x = \sum_n A(\omega_n)\Phi_n$, we can rewrite this in terms of the $\mathbf{A}$ coefficients and substitute it into (B.27) to give

$$\text{Info}(\omega_n) = \frac{1}{2}\log_2\left[\frac{\left\langle |A(\omega_n)|^2\right\rangle_{\mathbf{A}}}{\left\langle |A(\omega_n)|^2\right\rangle_{\mathbf{A}} - \frac{\left|\langle A(\omega_n)R^*(\omega_n;\mathbf{A})\rangle_{\mathbf{A}}\right|^2}{\left\langle |R(\omega_n;\mathbf{A})|^2\right\rangle_{\mathbf{A}}}}\right] \tag{B.33}$$

$$= \frac{1}{2}\log_2\left[\frac{\left\langle |R(\omega_n;\mathbf{A})|^2\right\rangle_{\mathbf{A}}}{\left\langle |R(\omega_n;\mathbf{A})|^2\right\rangle_{\mathbf{A}} - \frac{\left|\langle A(\omega_n)R^*(\omega_n;\mathbf{A})\rangle_{\mathbf{A}}\right|^2}{\left\langle |A(\omega_n)|^2\right\rangle_{\mathbf{A}}}}\right]. \tag{B.34}$$

The information rate, in bits/s, is thus

$$R_I = \frac{1}{2}\frac{\Delta\omega}{2\pi}\sum_n \text{Info}(\omega_n).$$

This page intentionally left blank

C Appendix: Chapter 5 derivations

C.1 Residual fluctuations due to spike trains

In this appendix we show how to determine the variance introduced into the representation of a scalar, x, through the filtering of spike trains. We first determine the fluctations that result when x is held at a constant value. We then derive an approximation that shows that on average these fluctuations are similar to the effects of random Gaussian noise.

Let us characterize the postsynaptic activity as

$$\alpha_i(x,t) = \sum_n h_i\left(t - n\Delta_i(x) + t_{i_0}\right), \tag{C.1}$$

where t_{i_0} is the time of the first spike, or the phase of the spike train relative to some fixed clock. We noted that these t_{i_0} are random variables that can take on any value between 0 and $\Delta_i(x)$ with equal probability, and are statistically independent across neurons, where $\Delta_i(x)$ is the interspike interval for neuron i given the value x. Note that even though the value of x is constant, the estimate of x will not be. It is given by

$$\hat{x}(t) = \sum_i \alpha_i(x,t). \tag{C.2}$$

As done in Rieke et al. (1997), we take the mean activity of each neuron to be

$$\begin{aligned} a_i(x) &= \langle \alpha_i(x,t)\rangle_T & \text{(C.3)}\\ &= \frac{1}{T}\sum_n \int_0^T h_i\left(t - n\Delta_i(x) + t_{i_0}\right)dt & \text{(C.4)}\\ &\approx \frac{N_i(x)}{T} & \text{(C.5)}\\ &= G_i[x], & \text{(C.6)}\end{aligned}$$

where $N_i(x)$ is the number of spikes observed during the time period T. The approximation above is due to the presence of the t_{i_0} phase variables, which can cause the spike count to vary by ± 1. As a result, the approximation becomes quite precise when T is long.

Equation (C.6) shows that the time average of the postsynaptic activity can be identified with the rate model of the neuron whose spikes gave rise to that activity. So, the mean value of the estimate of $\hat{x}(t)$ is given by

$$\langle \hat{x}(t)\rangle_T = \sum_i \phi_i a_i(x). \tag{C.7}$$

We would now like to estimate the variance of the fluctuations about this mean value, i.e.,

$$\sigma^2_{\hat{x}(t)} = \left\langle \left[\hat{x}(t) - \langle \hat{x}(t)\rangle_T\right]^2 \right\rangle_T.$$

Expanding this expression, we have

$$\begin{aligned}\sigma^2_{\hat{x}(t)} &= \frac{1}{T}\int_0^T \left[\hat{x}(t) - \langle \hat{x}(t)\rangle_T\right]^2 dt \\ &= \left\langle \frac{1}{T}\int_0^T \left[\sum_i \phi_i \left(\sum_n h_i\left(t - n\Delta_i(x) - t_{i_0}\right) - a_i(x)\right)\right]^2 dt \right\rangle_{t_{i_0}}.\end{aligned}$$

Averaging over the variables, t_{i_0}, causes the cross terms in this expression to vanish. This is because averaging over randomly started spike trains is just like averaging over time; i.e.,

$$\left\langle \sum_n h_i\left(t - n\Delta_i(x) - t_{i_0}\right) - a_i(x) \right\rangle_{t_{i_0}} = 0.$$

So, we can simply sum the variance for each neuron to determine the total variance of our estimate. Therefore,

$$\begin{aligned}\sigma^2_{\hat{x}(t)} &= \left\langle \frac{1}{T}\sum_i \phi_i^2 \int_0^T \left[\sum_n h_i\left(t - n\Delta_i(x) - t_{i_0}\right)\right]\right. \\ &\qquad \left.\left[\sum_m h_i\left(t - m\Delta_i(x) - t_{i_0}\right)\right] - a_i^2(x)\, dt \right\rangle_{t_{i_0}}.\end{aligned}$$

For each neuron we can write

$$\begin{aligned}&\left\langle \frac{1}{T}\int_0^T \sum_n h_i\left(t - n\Delta_i(x) - t_{i_0}\right) \sum_m h_i\left(t - m\Delta_i(x) - t_{i_0}\right) dt \right\rangle_{t_{i_0}} \\ &= \frac{N_i(x)}{T}\sum_m \int_0^T h_i(t')h_i(t' - m\Delta_i(x))dt' \\ &= \frac{N_i(x)}{T}\sum_m g_i\left(m\Delta_i(x)\right) \\ &= a_i(x)\sum_m g_i\left(m\Delta_i(x)\right),\end{aligned}$$

where, for each n in the sum over n, we set $t' = t - n\Delta_i(x) - t_{i_0}$ and

$$g_i(\tau) = \int_{-\infty}^{\infty} h_i(t)h_i(t-\tau)dt.$$

Therefore,

$$\sigma^2_{\hat{x}(t)} = \sum_i \phi_i^2 a_i(x)\left[\sum_m g_i\left(m\Delta_i(x)\right) - a_i(x)\right]. \tag{C.8}$$

As shown in the main text, in the limit where the filter time constant τ_{syn} is long compared to the interspike intervals (i.e., so the filtered spikes overlap one another) we find

$$\sum_m g_i\left(m\Delta_i(x)\right) \to a_i(x). \tag{C.9}$$

This means that the fluctuations in $\hat{x}(t)$ caused by the spikes become small, as expected. Conversely, when $\tau_{syn} < \Delta_i(x)$, then

$$\sum_m g_i\left(m\Delta_i(x)\right) \approx g_i(0) = k/\tau_{syn}, \tag{C.10}$$

where k depends on the functional form for the filter.

For the simple filter

$$h(t) = \begin{cases} \frac{1}{\tau_{syn}} e^{-t/\tau_{syn}} & \text{for } t \geq 0 \\ 0 & \text{for } t < 0 \end{cases} \tag{C.11}$$

we find that

$$g(\tau) = \frac{1}{2\tau_{syn}} e^{-|\tau|/\tau_{syn}}.$$

As a result,

$$\sum_{m=-\infty}^{\infty} g(m\Delta) = \frac{1}{\tau_{syn}}\left[\frac{1}{1-e^{-\mu}} - \frac{1}{2}\right]$$

where $\mu = \Delta/\tau_{syn}$. So, for a fixed value of x, the contribution to the fluctuation $\sigma^2_{\hat{x}}$ by each neuron can be found using (C.8) to be

$$\phi_i^2 a_i(x)\left[\sum_m g_i\left(m\Delta_i(x)\right) - a_i(x)\right]$$

$$= \phi_i^2 \frac{1}{\tau_{syn}^2 \mu_i(x)} \left[\frac{1}{1-e^{\mu_i(x)}} - \left(\frac{1}{2} + \frac{1}{\mu_i(x)} \right) \right] \tag{C.12}$$

$$\approx \phi_i^2 \frac{1}{\tau_{syn}^2} \left[\frac{1}{12} - \frac{\mu_i(x)^2}{720} + \frac{\mu_i(x)^4}{30240} \right]. \tag{C.13}$$

In the limit where there are many spikes per filter time constant, i.e., $\mu_i(x) = \Delta_i(x)/\tau_{syn} \to 0$, the fluctuations go to the constant value of $\frac{1}{12\tau_{syn}^2} = \frac{0.08\dot{3}}{\tau_{syn}^2}$. However, when the filter $h(t)$ is given a finite rise time at $t = 0$ (unlike the filter expressed by (C.11)) this factor goes to a maximum value of about $\frac{0.08\dot{3}}{\tau_{syn}^2}$ and then to zero as the firing rate increases to infinity (i.e., as $\Delta_i \to 0$).

We can average this fluctuation factor over x and over the neural population, to find an approximation to the variance of the error due to spiking fluctations of

$$E_{fluctuations} = \left\langle \sigma_{\hat{x}}^2 \right\rangle_x \approx \frac{0.04}{\tau_{syn}^2} \sum_i \phi_i^2. \tag{C.14}$$

In the models presented elsewhere in the book, we set the signal to noise level of the neurons to 10:1 when they are at their maximum firing rate, a_{max}, over the range of x. This leads to

$$E_{noise} \approx (0.1 a_{max})^2 \sum_i \phi_i^2 \propto (0.1 a_{max})^2 / N. \tag{C.15}$$

Comparing $E_{fluctuations}$ with E_{noise}, we see that they have the same functional dependence on the neuron number N and the decoding factors ϕ_i (recall that $\sum_i \phi_i^2 \propto \frac{1}{N}$). The contributions from the spikes and from other sources of noise become approximately equal when

$$a_{max} \approx 2/\tau_{syn}. \tag{C.16}$$

Roughly, the two are the same when the mean interspike interval is about equal to the synaptic filter time constant.

This result suggests that because neural systems use ensembles of spiking neurons to transmit analog signals, there is a built in level of fluctuation in the value being transmitted that cannot be avoided. To the extent that these fluctuations cannot be distinquished from other sources of noise, it does not make sense to have individual neurons with a higher signal to noise ratio than is set by this limit. In other words, if signals are going to be transmitted by a population of spiking neurons, then it does not help to use neurons that operate with a noise level much above a few bits per spike, as seems to be the case (see section 4.4.2).

D Appendix: Chapter 6 derivations

D.1 Coincidence detection

From section 6.3.1, we begin with

$$a_i(x)b_j(y) = \sum_{n,m} h_i(t - t_{in})h_j(t - t_{jm}).$$

We suppose that the filters $h(t)$ are narrow Gaussians and write this multiplication as

$$\begin{aligned} a_i(x)b_j(y) &= \sum_{n,m} e^{-(t-t_{in})^2/2\Delta^2} e^{-(t-t_{jm})^2/2\Delta^2} && \text{(D.1)} \\ &= \sum_{n,m} e^{-[(t-t_{in})^2+(t-t_{jm})^2]/2\Delta^2}. && \text{(D.2)} \end{aligned}$$

We now focus on re-writing the term in the square brackets,

$$\left[(t - t_{in})^2 + (t - t_{jm})^2\right] = 2t^2 - 2t\left(t_{in} + t_{jm}\right) + t_{in}^2 + t_{jm}^2,$$

into which we substitute

$$t_{in}^2 + t_{jm}^2 = \frac{\left(t_{in} + t_{jm}\right)^2}{2} + \frac{\left(t_{in} - t_{jm}\right)^2}{2},$$

which gives

$$\begin{aligned} &\left[(t - t_{in})^2 + (t - t_{jm})^2\right] \\ &\quad = 2t^2 - 2t\left(t_{in} + t_{jm}\right) + \frac{\left(t_{in} + t_{jm}\right)^2}{2} + \frac{\left(t_{in} - t_{jm}\right)^2}{2} \\ &\quad = 2\left[t^2 - t\left(t_{in} + t_{jm}\right) + \frac{\left(t_{in} + t_{jm}\right)^2}{4}\right] + \frac{\left(t_{in} - t_{jm}\right)^2}{2} \\ &\quad = 2\left[\left(t - \frac{t_{in} + t_{jm}}{2}\right)^2\right] + \frac{\left(t_{in} - t_{jm}\right)^2}{2}. \end{aligned}$$

Substituting this back into (D.2) gives the desired result:

$$\begin{aligned} a_i(x)b_j(y) &= \sum_{n,m} e^{-\left[2\left(t-\frac{t_{in}+t_{jm}}{2}\right)^2+\frac{1}{2}(t_{in}-t_{jm})^2\right]/2\Delta^2} \\ &= \sum_{n,m} e^{-2\left(t-\frac{t_{in}+t_{jm}}{2}\right)^2/2\Delta^2} e^{-\frac{1}{2}(t_{in}-t_{jm})^2/2\Delta^2}. \end{aligned}$$

This page intentionally left blank

E Appendix: Chapter 7 derivations

E.1 Practical considerations for finding linear decoders for $\mathbf{x}$ and $f(\mathbf{x})$

We have a good sense of the kinds of functions that can be computed with the population in figure 7.4. Namely, lower-order polynomials and their linear combinations are transformations that are easily supported by this population. Knowing this has important theoretical and practical consequences. From a theoretical standpoint, since we know that we can construct polynomial approximations to arbitrary functions, and that the arbitrary functions we can approximate with lower-order polynomials will be better supported by our population. In practice, we can use this knowledge to write very efficient code.

In particular, we show in this section that the two high-dimensional (i.e., $D > 2$) integrals required to determine the optimal decoding vectors for a population of neurons that represent a D-dimensional vector space can be reduced to 1 and 2 dimensions. This is important because it allows for the computationally efficient simulation of high-dimensional problems with large numbers of neurons (i.e., $D \sim 10$ and $N_{neurons} \sim 10^3$ or more).[1]

Let the neurons a_i encode the vector $\mathbf{x}$ as usual:

$$a_i(\mathbf{x}) = G_i\left[\alpha_i \left\langle \tilde{\phi}_i \mathbf{x} \right\rangle_n + J_i^{bias}\right]. \tag{E.1}$$

To keep the notation compact, we will use $a_i(\mathbf{x}) = G_i[\tilde{\phi}_i \mathbf{x}]$ in place of equation (E.1).

The optimal linear decoding vectors are found by computing the quantities

$$\begin{aligned} \Upsilon_i^{(1)} &= \int_{S_D} \mathbf{x} a_i(\mathbf{x})\, d\mathbf{x} \\ \gamma_{ij} &= \int_{S_D} a_i(\mathbf{x}) a_j(\mathbf{x})\, d\mathbf{x}, \end{aligned} \tag{E.2}$$

where the integrals are taken over the volume of the D-dimensional hypersphere S_D. The decoding vectors are then given by

$$\phi_i = \sum_j \Gamma_{ij}^{-1} \Upsilon_i^{(1)}, \tag{E.3}$$

where

$$\Gamma_{ij} = \gamma_{ij} + \sigma_\eta^2 \delta_{ij}. \tag{E.4}$$

1 For one- and two-dimensional problems, these integrals can be computed efficiently in the usual manner.

The superscript on $\Upsilon_i^{(1)}$ indicates that these are the first moments of the neuronal activites. The first moments are used to define the *representational* decoders, ϕ_i. We subsequently show how to compute other moments that can be used to find the *transformational* decoders used in estimating functions of $\mathbf{x}$.

First, let us consider the first moment in more detail. The first moment along the direction of the encoding vector $\tilde{\boldsymbol{\phi}}_i$ is given by

$$\begin{aligned}\Upsilon_i &= \int_{-1}^{1}\int_{S_{D-1}(x_1)} x_1 G_i[x_1]\, dx_1 d\mathbf{x}_{D-1} \\ &= \int_{-1}^{1} \mathcal{V}_{D-1}(x_1) x_1 G_i[x_1]\, dx_1 \\ &= K(D-1)\int_{-1}^{1} (1-x_1^2)^{\frac{D-1}{2}} x_1 G_i[x_1]\, dx_1. \end{aligned} \tag{E.5}$$

x_1 is the component along the first axis and $\mathcal{V}_{D-1}(x_1)$ is the volume of a $D-1$ hypersphere of radius $\sqrt{1-x_1^2}$. The volume of a D-dimensional hypersphere of radius R is equal to $K(D)R^D$, where

$$K(D) = \frac{\pi^{D/2}}{\Gamma\left(\frac{D}{2}+1\right)}. \tag{E.6}$$

$\Gamma(\cdot)$ is the well-known 'gamma function' whose definition can be found in a table of mathematical functions.[2] The integrals for all first order moments $\Upsilon_i^{(1)} = \Upsilon_{1,i}^{(1)}, \ldots \Upsilon_{Di}^{(1)}$ are then given by

$$\Upsilon_{ni}^{(1)} = \Upsilon_i^{(1)} \tilde{\phi}_{ni}, \tag{E.7}$$

where $\tilde{\phi}_{ni}$ are the components of the encoding vector $\tilde{\boldsymbol{\phi}}_i$. Thus, for any dimension of vector, we need only compute the 1-dimensional integral in (E.5).

Simiarly, the integrals for the γ_{ij} can reduced to 2 dimensions by aligning the first axis along the direction of the first encoding vector $\tilde{\boldsymbol{\phi}}_i$ and the second axis along a direction perpendicular to $\tilde{\boldsymbol{\phi}}_i$ and lying in the plane defined by the two encoding vectors $\tilde{\boldsymbol{\phi}}_i$ and $\tilde{\boldsymbol{\phi}}_j$. The integrals then become

$$\gamma_{ij} = K(D-2)\int_{-1}^{1}\int_{-\sqrt{1-x_1^2}}^{\sqrt{1-x_1^2}} (1-x_1^2-x_2^2)^{\frac{D-2}{2}} G_i[x_1] \tag{E.8}$$

$$G_j\left[x_1\cos(\theta_{ij}) + x_2\sin(\theta_{ij})\right]\, dx_1 dx_2, \tag{E.9}$$

2 Note that $\Gamma(M) = M!$, if M is an integer.

where

$$\begin{aligned}\cos(\theta_{ij}) &= \left\langle \boldsymbol{\phi}_i \boldsymbol{\phi}_j \right\rangle_n \\ \sin(\theta_{ij}) &= \sqrt{1-\cos^2(\theta_{ij})},\end{aligned} \tag{E.10}$$

where $\left\langle \boldsymbol{\phi}_i \boldsymbol{\phi}_j \right\rangle_n$ is the dot product of the two encoding vectors.

These computations can be sped up considerably by using the following modification. First we note that the lower limit to integral over x_1 starts at x_i^{thres}, where $G_i[x^{thres}] = 0$ is the point along the encoding vector where the neuron starts firing. Sorting the neurons from the largest to smallest values of x_i^{thres} ensures that the integrations are carried out over the neuron with the smallest range of nonzero values. Then we can set $\gamma_{ji} = \gamma_{ij}$ because the γ matrix is symmetric by definition.

As mentioned earlier, additional moments are required for decoding estimates of functions from the neuron activities; i.e., for determining the transformational decoders. The zeroth-order moment is readily computed as the 1-dimensional integral

$$\begin{aligned}\Upsilon_i^{(0)} &= \int_{S_D} a_i(\mathbf{x})\, d\mathbf{x} \\ &= K(D-1)\int_{-1}^{1} (1-x_1^2)^{\frac{D-1}{2}} G_i[x_1]\, dx_1.\end{aligned} \tag{E.11}$$

The second order moments,

$$\Upsilon_{mni}^{(2)} = \int_{S_D} x_m x_n a_i(\mathbf{x})\, d\mathbf{x}, \tag{E.12}$$

in principle require computing many integrals, but in fact it is only necessary to compute 2 for each neuron. The reason is that the neuronal activities are completely symmetrical about the axis of the encoding vector $\tilde{\boldsymbol{\phi}}_i$. Thus, in a coordinate system with the first axis aligned along this encoding direction, the moment matrix is diagonal and all of the terms except the first are identical.

The first moment is thus given by

$$\begin{aligned}\Upsilon_{1,1,i}^{(2)} &= \int_{S_D} x_1^2 a_i(\mathbf{x})\, d\mathbf{x} \\ &= K(D-1)\int_{-1}^{1} \left[1-x_1^2\right]^{\left(\frac{D-1}{2}\right)} x_1^2 a_i(x_1)\, dx_1,\end{aligned} \tag{E.13}$$

and the trace of the moment matrix, $\sum_n \Upsilon^{(2)}_{nni}$, by

$$\begin{aligned}\Upsilon^{(2)}_{ti} &= \int_{S_D} |\mathbf{x}|^2 a_i(\mathbf{x})\, d\mathbf{x} \\ &= K(D-2)H(D)\int_{-1}^{1} \left[1-x_1^2\right]^{\left(\frac{D+1}{2}\right)} a_i(x_1)\, dx_1,\end{aligned} \tag{E.14}$$

where

$$H(D) = \sqrt{\pi}\left(\frac{\Gamma(D/2)}{\Gamma((D+1)/2)} - \frac{\Gamma((D+2)/2)}{\Gamma((D+3)/2))}\right). \tag{E.15}$$

The weights ϕ_i^f for decoding the bilinear function

$$\begin{aligned}f(\mathbf{x}) &= \sum_{m,n} C_{mn}x_n x_m \\ \hat{f}(\mathbf{x}) &= \sum_i \phi_i^f a_i(\mathbf{x})\end{aligned} \tag{E.16}$$

are given by

$$\phi_i^f = \sum_j \Gamma_{ij}^{-1}\left(\text{trace}(\mathbf{C})\Upsilon^{(2)}_{tj} + \sum_{m,n}\tilde{\phi}_{mj}C_{mn}\tilde{\phi}_{nj}(\Upsilon^{(2)}_{1,1,j} - \Upsilon^{(2)}_{tj})\right), \tag{E.17}$$

where

$$\Gamma_{ij} = \gamma_{ij} + \sigma_\eta^2\delta_{ij}. \tag{E.18}$$

Additionally, only two moments for each neuron are required for third order functions. However, higher-order functions demand ever increasing numbers of moments. These techniques have proven very useful for computing nonlinear functions of high-dimensional vectors, as in the vestibular system example (see section 6.5).

Finally, it is often useful to introduce a weighting function to these derivations, that puts more emphasis on the mean square decoding error at the origin. This kind of strategy seems to have been adopted by some neurobiological systems, like the neural integrator (see, e.g., figure 5.2). So, let us introduce the weighting function

$$W(\mathbf{x}) = (1-|\mathbf{x}^2|)^{(\frac{\alpha}{2})}, \tag{E.19}$$

with $\alpha > 0$. This function has its maximum at $\mathbf{x} = \mathbf{0}$ and goes to zero on the surface of the hypersphere. Fortunately, the only modification needed to the integrals given above is to change D to $D + \alpha$. For example,

$$\begin{aligned}
\Upsilon_i &= K(D+\alpha-1)\int_{-1}^{1}(1-x_1^2)^{\left(\frac{D+\alpha-1}{2}\right)} x_1 G_i[x_1]\, dx_1 \\
\gamma_{ij} &= K(D+\alpha-2)\int_{-1}^{1}\int_{-\sqrt{1-x_1^2}}^{\sqrt{1-x_1^2}}(1-x_1^2-x_2^2)^{\left(\frac{D+\alpha-2}{2}\right)} G_i[x_1] \\
&\qquad G_j\left[\phi_{1,j}x_1 + \phi_{2,j}x_2\right]\, dx_1 dx_2 .
\end{aligned}$$

E.2 Finding the useful representational space

In this appendix, we show how to relate the noise and singular values of the neuron response function matrix, γ, to find the *useful* representational space. Here we concentrate on the case of vectors, but the same derivation applies to scalars and functions with slight notational modifications. To begin, we write our estimate as

$$\begin{aligned}
\hat{\mathbf{x}} &= \sum_i a_i(\mathbf{x})\boldsymbol{\phi}_i \\
&= \sum_m \chi_m \Phi_m, \qquad \text{(E.20)}
\end{aligned}$$

where

$$\begin{aligned}
\chi_m(\mathbf{x}) &= \sum_i a_i(\mathbf{x}) U_{im} \\
\Phi_m &= \sum_i U_{mi}\boldsymbol{\phi}_i. \qquad \text{(E.21)}
\end{aligned}$$

For simplicity, we will write χ_m in place of $\chi_m(\mathbf{x})$, though it is important to keep in mind that these are a function of $\mathbf{x}$. Note that the χ vectors are orthogonal:

$$\begin{aligned}
\langle \chi_n \chi_m \rangle_{\mathbf{x}} &= \left\langle \sum_i U_{ni} a_i(\mathbf{x}) \sum_j U_{mj} a_j(\mathbf{x}) \right\rangle_{\mathbf{x}} \\
&= \sum_{i,j} U_{ni} U_{mj} \langle a_i(\mathbf{x}) a_j(\mathbf{x}) \rangle_{\mathbf{x}} \qquad \text{(E.22)} \\
&= \sum_{i,j,k} U_{ni} U_{mj} U_{ik} S_k U_{jk} \qquad \text{(E.23)} \\
&= \sum_k \delta_{nk}\delta_{mk} S_k \\
&= \delta_{nm} S_n. \qquad \text{(E.24)}
\end{aligned}$$

In (E.22) we see the expression for $\boldsymbol{\gamma}$, to which we apply SVD. The results of the SVD are expressed in terms of $\mathbf{U}$ because of the symmetry of the γ matrix. Substituting this result gives (E.23) which can be simplified to (E.24). Notably, (E.24) shows that the χ_n are a non-normalized set of orthogonal functions. This result will be useful for simplifying the following derivations.

From (2.9), in chapter 2, it is clear that the error with noise can be written in the vector case as

$$\begin{aligned} E &= \left\langle \left[\mathbf{x} - \sum_i \left(a_i(\mathbf{x}) + \eta_i\right) \boldsymbol{\phi}_i \right]^2 \right\rangle_{\mathbf{x},\eta} \\ &= \left\langle \left[\mathbf{x} - \sum_i a_i(\mathbf{x}) \boldsymbol{\phi}_i \right]^2 + \sigma_\eta^2 \sum_i \boldsymbol{\phi}_i^2 \right\rangle_{\mathbf{x}}. \end{aligned} \tag{E.25}$$

We can now use our expression from (E.20) and (E.21) in (E.25) to give

$$\begin{aligned} E &= \left\langle \left[\mathbf{x} - \sum_m \chi_m \Phi_m \right]^2 + \sigma_\eta^2 \sum_{i,j} \boldsymbol{\phi}_i \delta_{ij} \boldsymbol{\phi}_j \right\rangle_{\mathbf{x}} \\ &= \left\langle \left[\mathbf{x} - \sum_m \chi_m \Phi_m \right]^2 + \sigma_\eta^2 \sum_{i,j} \boldsymbol{\phi}_i \sum_m U_{im} U_{mj} \boldsymbol{\phi}_j \right\rangle_{\mathbf{x}} \\ &= \left\langle \left[\mathbf{x} - \sum_m \chi_m \Phi_m \right]^2 + \sigma_\eta^2 \sum_m \Phi_m^2 \right\rangle_{\mathbf{x}}. \end{aligned} \tag{E.26}$$

We minimize this error by taking the derivative of (E.26) and setting it to zero to get an expression for the optimal Φ functions under noise:

$$\begin{aligned} \frac{dE}{d\Phi_n} &= \left\langle 2 \left[\mathbf{x} - \sum_m \chi_m \Phi_m \right] (-\chi_n) + 2\sigma_\eta^2 \Phi_n \right\rangle_{\mathbf{x}} \\ 0 &= -2 \left\langle \chi_n \mathbf{x} \right\rangle_{\mathbf{x}} + 2 \left\langle \sum_m \chi_m \chi_n \Phi_n \right\rangle_{\mathbf{x}} + 2\sigma_\eta^2 \Phi_n \\ \left\langle \chi_n \mathbf{x} \right\rangle_{\mathbf{x}} &= S_n \Phi_n + \sigma_\eta^2 \Phi_n \\ \Phi_n &= \frac{\left\langle \chi_n \mathbf{x} \right\rangle_{\mathbf{x}}}{S_n + \sigma_\eta^2}. \end{aligned} \tag{E.27}$$

We can use this expression to determine what the residual error will be (i.e., the expected error using these Φ functions). This analysis needs to be done under noisy conditions. So it will be useful to determine the effects of noise on the orthogonality of our χ functions:

$$\begin{aligned}
&\langle(\chi_n(\mathbf{x})+\eta_i U_{ni})(\chi_m(\mathbf{x})+\eta_j U_{mj})\rangle_{\mathbf{x},\eta}\\
&=\left\langle\sum_i U_{ni}\left(a_i(\mathbf{x})+\eta_i\right)\sum_j U_{mj}\left(a_j(\mathbf{x})+\eta_j\right)\right\rangle_{\mathbf{x},\eta}\\
&=\sum_{i,j} U_{ni}U_{mj}\langle a_i(\mathbf{x})a_j(\mathbf{x})\rangle_{\mathbf{x}}+\left\langle\sum_{i,j} U_{ni}U_{mj}\eta_i\eta_j\right\rangle_{\eta} &&\text{(E.28)}\\
&=\sum_{i,j,k} U_{ni}U_{mj}U_{ik}S_kU_{jk}+\delta_{nm}\sigma_\eta^2 &&\text{(E.29)}\\
&=\sum_k \delta_{nk}\delta_{mk}S_k+\delta_{nm}\sigma_\eta^2\\
&=\delta_{nm}(S_n+\sigma_\eta^2). &&\text{(E.30)}
\end{aligned}$$

As can be seen from (E.30), the functions remain orthogonal, although they are scaled by the variance of the noise along with the singular values. We can now determine the residual error as follows:

$$\begin{aligned}
E_r &= \left\langle[\mathbf{x}-\hat{\mathbf{x}}]^2\right\rangle_{\mathbf{x},\eta}\\
&= \left\langle\left[\mathbf{x}-\sum_m \chi_m\Phi_m\right]^2\right\rangle_{\mathbf{x},\eta}\\
&= \left\langle\mathbf{x}^2\right\rangle_{\mathbf{x},\eta}-2\left\langle\mathbf{x}\sum_m\chi_m\Phi_m\right\rangle_{\mathbf{x},\eta}+\left\langle\sum_m\chi_m\Phi_m\right\rangle^2_{\mathbf{x},\eta}.
\end{aligned}$$

Substituting the expression in (E.27) for Φ_m gives

$$\begin{aligned}
E_r &= \left\langle\mathbf{x}^2\right\rangle_{\mathbf{x},\eta}-2\left\langle\mathbf{x}\sum_m\chi_m\frac{\langle\chi_m\mathbf{x}\rangle_{\mathbf{x}}}{S_m+\sigma_\eta^2}\right\rangle_{\mathbf{x},\eta}+\left\langle\sum_m\chi_m\frac{\langle\chi_m\mathbf{x}\rangle_{\mathbf{x}}}{S_m+\sigma_\eta^2}\right\rangle^2_{\mathbf{x},\eta}\\
&= \left\langle\mathbf{x}^2\right\rangle_{\mathbf{x}}-2\sum_m\frac{\langle\chi_m\mathbf{x}\rangle^2_{\mathbf{x}}}{S_m+\sigma_\eta^2}+\sum_m(S_m+\sigma_\eta^2)\frac{\langle\chi_m\mathbf{x}\rangle^2_{\mathbf{x}}}{\left(S_m+\sigma_\eta^2\right)^2}\\
&= \left\langle\mathbf{x}^2\right\rangle_{\mathbf{x}}-\sum_m\frac{\langle\chi_m\mathbf{x}\rangle^2_{\mathbf{x}}}{S_m+\sigma_\eta^2}.
\end{aligned}$$

So the mth element in the sum determines by how much the mth basis function helps reduce the error. Essentially, the singular values, S_m act to normalize the contribution from the related basis function χ_m. However, as the noise grows relative to that normalization factor, the normalization will become inappropriate and the χ_m will contribute too much or too little to the reconstruction. So, as mentioned in the main text, this expression provides a good way to determine which basis functions are the important ones since those with singular values $S_m < \sigma_\eta^2$ will not be usefully contributing to the representation.

F Appendix: Chapter 8 derivations

F.1 Synaptic dynamics dominate neural dynamics

To see that synaptic dynamics generally dominate neural dynamics, consider the expression for $V(t)$ that we derived in section 4.1.2:

$$V(t) = J_M R \left(1 - e^{-t/\tau^{RC}}\right).$$

In general, $e^{-x} \approx 1 - x$ for small values of x, so we can approximate the factor in the brackets as $\frac{t}{\tau^{RC}}$. Given that $\tau^{RC} = RC$, we know

$$V(t) \approx \frac{J_M}{C} t. \tag{F.1}$$

So, for an input current, J_M, which is over the threshold, the time it takes the neuron to reach threshold, ΔT, is

$$\Delta T = V_{th} \frac{C}{J_M}. \tag{F.2}$$

Notably, (F.2) does not depend on the membrane resistance, R. In fact, this equation holds true for changing input current as well, as long as the change in current is small over ΔT. However, when the neuron is operating near threshold, these assumptions will be poor ones. Nevertheless, supposing that neurons tend not to spend most of their time near threshold, and knowing that the capacitance of neural membranes tends to lie within a small range, (F.2) suggests that differences in the RC time constant of neurons will not greatly affect the dynamic properties of a neural system. Furthermore, because the input current, J_M, is coming from the synapses, and because it is the only real variable quantity in (F.1), it will be the dynamics of the synapses that dominates this kind of system. This same basic point has been made by Knight (1972), and more recently by Dayan and Abbott (2001).

F.2 Derivations for the lamprey model

F.2.1 Determining muscle tension

To relate the tensions, T to the normal forces F^n, we balance torques about the point n (see figures 8.24 and 8.25). In particular, we know that the forces created by the muscle tensions will be equal and opposite to those created by the resistance of the water. Thus we can write the following two expressions, whose difference is the total torque around n:

$$\begin{aligned} l_1 F_{n+1}^n &= l_2(T_n^+ - T_n^- - T_{n+1}^+ + T_{n+1}^-) \\ l_1 F_n &= l_2(T_{n-1}^+ + T_n^- - T_n^+ - T_{n-1}^-). \end{aligned}$$

Letting $\gamma = \frac{l_1}{l_2}$, we can form the difference, giving

$$\begin{aligned} \gamma(F_{n+1}^n - F_n^n) &= (2T_n^+ - 2T_n^- - T_{n+1}^+ + T_{n+1}^- - T_{n-1}^+ + T_{n-1}^-) \\ &= 2T_n - T_{n-1} - T_{n+1}, \end{aligned}$$

where $T_n = T_n^+ - T_n^-$ is the total tension at n from muscles on both sides of the lamprey's body.

Now notice that letting

$$\frac{\Delta T}{\Delta z} = \frac{T_n - T_{n-1}}{n - (n-1)} = G,$$

we find the second derivative as

$$\begin{aligned} \frac{\Delta G}{\Delta z} &= \frac{G_n - G_{n-1}}{n - (n-1)} \\ &= \frac{\frac{T_n - T_{n-1}}{n-(n-1)} - \frac{T_{n-1} - T_{n-2}}{(n-1)-(n-2)}}{n - (n-1)} \\ &= (T_n - T_{n-1}) - (T_{n-1} - T_{n-2}) \\ &= T_n - 2T_{n-1} + T_{n-2}. \end{aligned}$$

So, we can write

$$-\frac{\partial^2 T}{\partial z^2} = \gamma \frac{\partial F^n}{\partial z} = \gamma\eta\omega k A \sin(\omega t - kz).$$

We find T by integrating:

$$T(z,t) = \frac{\gamma\eta\omega A}{k} \sin(\omega t - kz) + K_1(t) + K_2(t)z.$$

The boundary conditions for this expression are

$$T(0,t) = T(L,t) = 0,$$

because there can be no torque at the head and tail. Letting $\kappa = \frac{\gamma\eta\omega A}{k}$, we find that

$$K_1(t) = -\kappa \sin(\omega t),$$

and

$$K_2(t) = \frac{\kappa}{L}(\sin(\omega t) - \sin(\omega t - kL)).$$

We also know that $k = \frac{2\pi}{L}$, because the lamprey swims in such a way that one period of the wave is present over its length, no matter the speed. Therefore,

$$\begin{aligned} K_2(t) &= \frac{\kappa}{L}(\sin(\omega t) - \sin(\omega t - 2\pi)) \\ &= 0. \end{aligned}$$

As a result, the tension needed to produce the observed swimming is given by

$$T(z,t) = \kappa(\sin(\omega t - 2\pi z/L) - \sin(\omega t)). \tag{F.3}$$

We can further simplify by allowing $\gamma = \eta = A = L = 1$ in their respective units, which is fine so long as we do not need to know the actual tensions produced (i.e., we just need the relative magnitudes).

F.2.2 Error

As mentioned in the text, we need to solve the error

$$E = \left\langle \left[T(z,t) - \hat{T}(z,t)\right]^2 \right\rangle_{t,z}. \tag{F.4}$$

To do this, we take as our two expressions

$$\begin{aligned} T(z,t) &= \kappa(\sin(\omega t - 2\pi z/L) - \sin(\omega t)) && \text{(F.5)} \\ &= \kappa(\sin(\omega t)\cos(kz) - \cos(\omega t)\sin(kz) - \sin(\omega t)), && \text{(F.6)} \end{aligned}$$

and

$$\hat{T}(z,t;\mathbf{A}) = \kappa\left(A_0 + \textstyle\sum_{n=1}^{N} A_{2n-1}(t)\sin(2\pi nz) + A_{2n}(t)\cos(2\pi nz)\right). \tag{F.7}$$

To make our derivation clearer, let us define

$$\sum_n B_n = \sum_{n=1}^{N} A_{2n-1}(t)\sin(2\pi nz) + A_{2n}(t)\cos(2\pi nz).$$

As well, we leave out the constant κ, which simply carries through the derivation.

We now explicitly write the difference between our representation and the desired function, which gives

$$\begin{aligned}E &= \Big\langle\Big(-\sin(\omega t) - A_0 - \cos(\omega t)\sin(kz) - A_1\sin(kz) + \sin(\omega t)\cos(kz) \\ &\qquad -A_2\cos(kz) - \sum_n B_n\Big)^2\Big\rangle_{z,t} \\ &= \Big\langle\Big(-(\sin(\omega t) + A_0) - \sin(kz)(\cos(\omega t) + A_1) + \cos(kz)(\sin(\omega t) - A_2) \\ &\qquad -\sum_n B_n^2\Big)^2\Big\rangle_{z,t}.\end{aligned}$$

Performing the square and averaging over z gives

$$\begin{aligned}E &= \left\langle(\sin(\omega t) + A_0)^2 + \frac{1}{2}(\cos(\omega t) + A_1)^2 + \frac{1}{2}(\sin(\omega t) - A_2)^2\right. \\ &\qquad \left.+\frac{1}{2}\sum_{n=3} A_n^2 + \frac{1}{2}\sum_{n=3} A_n^2\right\rangle_t.\end{aligned} \tag{F.8}$$

This follows since all of the $\sin(knz)$ and $\cos(knz)$ terms are orthogonal, so the cross terms cancel. As well, note that $\langle\cos^2(kz)\rangle_z = \langle\sin^2(kz)\rangle_z = \frac{1}{2}$. From (F.8) we can see that in order for the error to be zero, we want $A_0 = -\sin(\omega t)$, $A_1 = -\cos(\omega t)$, $A_2 = \sin(\omega t)$, and $A_n = 0$ for higher terms.

We can proceed further with an analysis of the error to determine precisely how the dynamics of these coefficients need to be constrained. We notice that the first two coefficients form an oscillator. However, it is not clear how the third coefficient contributes to the behavior of the oscillator. Noting that $A_0 = -A_2$, we suspect that the third term does not contribute significantly. To ensure this, we focus for a moment on the first and third terms in (F.8), to determine their relation. Performing the square on these terms gives

$$\begin{aligned}&\left\langle\sin^2(\omega t) + 2A_0\sin(\omega t) + A_0^2 + \frac{1}{2}\sin^2(\omega t) - A_2\sin(\omega t) + \frac{1}{2}A_2^2\right\rangle_t \\ &\quad = \left\langle\frac{3}{2}\sin^2(\omega t) + (2A_0 - A_2)\sin(\omega t) + A_0^2 + \frac{1}{2}A_2^2\right\rangle_t \\ &\quad = \left\langle\frac{1}{2}(A_0 + \sin(\omega t))^2 + \left(\frac{A_0 + A_2}{2}\right)^2 + \left(\frac{A_0 - A_2}{2} + \sin(\omega t)\right)^2\right\rangle_t,\end{aligned}$$

resulting in a final error of

$$\begin{aligned} E &= \Bigg\langle \frac{1}{2}(A_1 + \cos(\omega t))^2 + \frac{1}{2}(A_0 + \sin(\omega t))^2 + \left(\frac{A_0 + A_2}{2}\right)^2 \\ &\quad + \left(\frac{A_0 - A_2}{2} + \sin(\omega t)\right)^2 + \sum_{n=3} A_n^2(t) \Bigg\rangle_t . \end{aligned} \tag{F.9}$$

This expression makes it clear that *over time* we must ensure that $A_0 = -A_2 = -\sin(\omega t)$, in order to minimize the error. As well, the higher order terms must be damped to zero. We can also see that A_1 and A_0 form an oscillator which comprises the central dynamics of the swimming lamprey.

F.2.3 Oscillator dynamics

We begin with the standard oscillator equation,

$$\begin{aligned} \ddot{y} + \omega^2 y &= 0 \\ \ddot{y} &= -\omega^2 y, \end{aligned}$$

which intuitively says that the acceleration of an oscillating body is proportional but directed opposite to the displacement of the body. This is precisely what we see in a pendulum, for example. Each time, and no matter which direction a pendulum crosses the vertical plane, there is an acceleration (usually gravity) pulling in the opposite direction. We can now define the variables

$$\begin{aligned} x_1 &= \dot{y} \\ x_2 &= \omega y, \end{aligned}$$

which gives

$$\begin{aligned} \dot{x}_1 &= \ddot{y} = -\omega^2 y = -\omega x_2 \\ \dot{x}_2 &= \omega \dot{y} = \omega x_1 . \end{aligned}$$

Therefore, the position of the oscillating body at any time is given by $x_2/\omega = y$. We can now write these equations in matrix form to give

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & \omega \\ -\omega & 0 \end{bmatrix} \mathbf{x}. \tag{F.10}$$

A solution for $\mathbf{x}$ is

$$\begin{aligned} x_1 &= -\sin(\omega t) \\ x_2 &= -\cos(\omega t), \end{aligned}$$

so

$$\begin{aligned} \dot{x}_1 &= -\omega\cos(\omega t) \\ \dot{x}_2 &= \omega\sin(\omega t), \end{aligned}$$

which we also find by doing the matrix multiplication in (F.10), as expected. Our first two coefficients in the orthogonal expression for the lamprey locomotion are precisely x_1 and x_2. This shows that they implement an oscillator that can be expressed by an equation of the form of (F.10).

F.2.4 Coordinate changes with matrices

Coordinate changes are useful for transforming some linear transformation or vector to a convenient coordinate system. In fact, any linear transform is a coordinate change, so this kind of operation is ubiquitous. We begin with an oscillator defined by

$$\begin{aligned} \dot{A}_0 &= -\omega A_1 \\ \dot{A}_1 &= \omega A_0 \\ \frac{d(\frac{A_0+A_2}{2})}{dt} &= 0, \end{aligned}$$

and

$$\dot{\mathbf{A}}' = \mathbf{M}\mathbf{A}',$$

where

$$\mathbf{M} = \begin{bmatrix} 0 & \omega & 0 \\ -\omega & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

In order to transform this matrix to the space of our original variables, we need first to relate the two coordinate systems. Thus, we write

$$\mathbf{A}' = \mathbf{R}\mathbf{A}$$

$$\begin{bmatrix} A_0 \\ A_1 \\ \frac{A_0+A_2}{2} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} A_0 \\ A_1 \\ A_2 \end{bmatrix}.$$

Now we can express the oscillator in our original coordinates by noting that

$$\begin{aligned} \dot{\mathbf{A}}' &= \mathbf{M}\mathbf{A}' \\ \mathbf{R}\dot{\mathbf{A}} &= \mathbf{M}\mathbf{R}\mathbf{A} \\ \mathbf{R}^{-1}\mathbf{R}\dot{\mathbf{A}} &= \mathbf{R}^{-1}\mathbf{M}\mathbf{R}\mathbf{A} \\ \dot{\mathbf{A}} &= \mathbf{R}^{-1}\mathbf{M}\mathbf{R}\mathbf{A}. \end{aligned}$$

Thus, the transformation matrix for the oscillator in our original coordinates is

$$\begin{aligned} \mathbf{M}_{\mathbf{A}} &= \mathbf{R}^{-1}\mathbf{M}\mathbf{R} \\ &= \begin{bmatrix} 0 & \omega & 0 \\ -\omega & 0 & 0 \\ 0 & -\omega & 0 \end{bmatrix}. \end{aligned}$$

F.2.5 Projection matrices

To construct and use projection matrices for moving between the orthogonal and intermediate spaces, let us first recall our two representations of the tension,

$$T(z,t) = \mathbf{\Phi}\mathbf{A}$$

and

$$\mathbf{a} = \tilde{\phi}T(z,t).$$

We can now substitute the expression for $T(z,t)$ into the expression for $\mathbf{a}$ and simplify to get an expression for the orthogonal coefficients in terms of the intermediate ones:

$$\begin{aligned} \mathbf{a} &= \tilde{\phi}\mathbf{\Phi}\mathbf{A} \\ \left[\tilde{\phi}\mathbf{\Phi}\right]^{-1}\mathbf{a} &= \mathbf{A} \\ \mathbf{A} &= \Theta\mathbf{a}, \end{aligned}$$

where $\Theta = \left[\tilde{\phi}\mathbf{\Phi}\right]^{-1}$ is a pseudo-inverse (i.e., using SVD).

Substituting this expression for $\mathbf{A}$ into our dynamical equation, and recalling that $\mathbf{U} = \mathbf{A}$ during startup gives

$$\begin{aligned} \dot{\mathbf{A}} &= \mathbf{M}_D\mathbf{A} + \mathbf{M}_I\mathbf{U} \\ \Theta\dot{\mathbf{a}} &= \mathbf{M}_D\Theta\mathbf{a} + \mathbf{M}_I\Theta\mathbf{a} \\ \dot{\mathbf{a}} &= \Theta^{-1}\mathbf{M}_D\Theta\mathbf{a} + \Theta^{-1}\mathbf{M}_I\Theta\mathbf{a}. \end{aligned}$$

Therefore,

$$\dot{\mathbf{a}} = \mathbf{m}_D\mathbf{a} + \mathbf{m}_I\mathbf{a}, \tag{F.11}$$

where

$$\begin{aligned} \mathbf{m}_D &= \Theta^{-1}\mathbf{M}_D\Theta \\ \mathbf{m}_I &= \Theta^{-1}\mathbf{M}_I\Theta. \end{aligned}$$

References

Abbott, L. F. (1994). Decoding neuronal firing and modelling neural networks. *Quarterly Review of Biophysics 27*, 291–331.

Abbott, L. F. and S. B. Nelson (2000). Synaptic plasticity: Taming the beast. *Nature Neuroscience 3*, 1178–1183.

Adelson, E. and J. Bergen (1985). Spatiotemporal energy models for the perception of motion. *Journal of the Optical Society of America A2*, 284–299.

Adrian, E. D. (1928). *The basis of sensation: The action of the sense organs*. New York, NY: Norton.

Amit, D. J. (1989). *Modeling brain function: The world of attractor neural networks*. New York, NY: Cambridge University Press.

Amit, Y. and D. Geman (1999). A computational model for visual selection. *Neural Computation 11*, 1691–1715.

Andersen, R. A., L. H. Snyder, D. C. Bradley, and J. Xing (1997). Multimodal representation of space in the posterior parietal cortex and its use in planning movements. *Annual Review of Neuroscience 20*, 303–330.

Anderson, C. H., Q. Huang, and J. Clark (2000). Harmonic analysis of spiking neuronal pairs. *Neurocomputing 32–33*, 279–284.

Anderson, C. H. and D. C. Van Essen (1987). Shifter circuits: A computational strategy for directed visual attention and other dynamic neural processes. *Proceedings of the National Academy of Sciences USA 84*, 6297–6301.

Angelaki, D. E. and J. D. Dickman (2000). Spatiotemporal processing of linear acceleration: Primary afferent and central vestibular neuron responses. *Journal of Neurophysiology 84*, 2113–2132.

Angelaki, D. E., M. Q. McHenry, J. D. Dickman, S. D. Newlands, and B. J. M. Hess (1999). Computation of inertial motion: neural strategies to resolve ambiguous otolith information. *Journal of Neuroscience 19*, 316–327.

Arbib, M. (1995). Background. In M. Arbib (Ed.), *The handbook of brain theory and neural networks*. Cambridge, MA: MIT Press.

Arnold, D. B. and D. A. Robinson (1991). A learning network model of the neural integrator of the oculomotor system. *Biological Cybernetics 64*, 447–454.

Arvanitaki, A. (1938). *Les variations graduées de la polarisation des systèmes excitables*. Paris: Univ. Lyons, Hermann et cie.

Askay, E., R. Baker, H. S. Seung, R. Baker, and D. Tank (2000). Anatomy and discharge properties of pre-motor neurons in the goldfish medulla that have eye-position signals during fixations. *Journal of Neurophysiology 84*, 1035–1049.

Askay, E., G. Gamkrelidze, H. S. Seung, R. Baker, and D. Tank (2001). In vivo intracellular recording and perturbation of persistent activity in a neural integrator. *Nature Neuroscience 4*, 184–193.

Avoli, M., G. G. C. Hwa, J.-C. Lacaille, A. Olivier, and J.-G. Villemure (1994). Electrophysiological and repetitive firing properties of neurons in the superficial/middle layers of the human neocortex maintained in vitro. *Experimental Brain Research 98*, 135–144.

Baddeley, R. (1996). An efficient code in V1? *Nature 381*, 560–561.

Bair, W. and C. Koch (1995). Precision and reliability of neocortical spike trains in the behaving monkey. In *The Neurobiology of Computation: Proceedings of the 3rd Computation and Neural Systems Conference*. Kluwer Academic Press.

Barber, M. J. (1999). *Studies in neural networks: Neural belief networks and synapse elimination*. PhD dissertation, Washington University in St. Louis, Department of Physics.

Bechtel, W. and R. C. Richardson (1993). *Discovering complexity: Decomposition and localization as strategies in scientific research*. Princeton, NJ: Princeton University Press.

Becker, G. C. (1983). *Fishes of Wisconsin*. Madison, WI: University of Wisconsin Press.

Berry II, M. J. and M. Meister (in press). Firing events: Fundamental symbols in the neural code of retinal ganglion cells? In *Computational Neuroscience: Trends in Research 2000*, New York, NY. Plenum Press.

Bialek, W. and F. Rieke (1992). Reliability and information transmission in spiking neurons. *Trends in Neuroscience 1*, 428–434.

Bialek, W., F. Rieke, R. R. de Ruyter van Steveninck, and D. Warland (1991). Reading a neural code. *Science 252*, 1854–1857.

Bialek, W. and A. Zee (1990). Coding and computation with neural spike trains. *Journal of Statistical Physics 59*, 103–115.

Bickle, J., C. Worley, and M. Bernstein (2000). Vector subtraction implemented neurally: A neurocomputational model of sequential cognitive and conscious processing. *Counsciousness and Cognition 9*, 117–144.

Bienenstock, E., L. N. Cooper, and P. Munro (1982). On the development of neuron selectivity: Orientation specificity and binocular interaction in visual cortex. *Journal of Neuroscience 2*, 32–48.

Bousquet, O., K. Balakrishnan, and V. Honavar (1998). Is the hippocampus a kalman filter? In *Pacific Symposium on Biocomputing*, River Edge, NJ, pp. 655–666. World Scientific.

Bower, J. M. and D. Beeman (Eds.) (1995). *The book of Genesis: Exploring realistic neural models with the GEneral NEural SImulation System*. Santa Clara, CA: Springer-Verlag.

Brödel, M., P. D. Malone, S. R. Guild, and S. J. Crowe (1946). *Three unpublished drawings of the anatomy of the human ear*. Philadelphia: Saunders.

Bugmann, G. (1991). Summation and multiplication: Two distinct operation domains of leaky integrate-and-fire neurons. *Network 2*, 489–509.

Buracas, G. T., A. M. Zador, M. R. DeWeese, and T. D. Albright (1998). Efficient discrimination of temporal patterns by motion-sensitive neurons in primate visual cortex. *Neuron 20*, 959–969.

Büttner-Ennever, J. A. (1999). A review of otolith pathways to brainstem and cerebellum. *Annals of New York Academy of Science 871*, 51–64.

Camperi, M. and X. J. Wang (1998). A model of visuospatial working memory in prefrontal cortex: recurrent network and cellular bistability. *Journal of Computational Neuroscience 5*, 383–405.

Cavanagh, P. (1991). What's up in top-down processing? In A. Gorea (Ed.), *Representations of vision: Trends and tacit assumptions in vision research*. Cambridge, UK: Cambridge University Press.

Chen, L. L., L. H. Lin, C. A. Barnes, and B. L. McNaughton (1994). Head-direction cells in the rat posterior cortex. II. contributions of visual and ideothetic information to the directional firing. *Experimental Brain Research 101*, 24–34.

Chen, L. W., K. K. L. Yung, and Y. S. Chan (2000). Co-localization of NMDA receptors and AMPA receptors in neurons. *Brain Research 884*, 87–97.

Churchland, P. S. and T. J. Sejnowski (1992). *The computational brain*. Cambridge, MA: MIT Press.

Cohen, A. H., P. J. Holmes, and R. H. Rand (1982). The nature of the coupling between segmental oscillators of the lamprey spinal generator for locomotion. *Journal of Mathematical Biology 13*, 345–369.

Colby, C. L. and M. E. Goldberg (1999). Space and attention in parietal cortex. *Annual Review of Neuroscience 22*, 319–349.

Connors, B. W. and M. J. Gutnick (1990). Intrinsic firing patterns of diverse neocortical neurons. *Trends in Neuroscience 13*, 99–104.

Crick, F. and C. Koch (1998). Consciousness and neuroscience. *Cerbral Cortex 8*, 97–107.

Curthoys, I. S. and C. H. Markham (1971). Convergence of labyrinthine influences on units in the vestibular nuclei of the cat. I: Natural stimulation. *Brain Research 35*, 469–490.

Davson, H. (1990). *Physiology of the eye* (5th ed.). New York, NY: Pergamon Press.

Dayan, P. and L. F. Abbott (2001). *Theoretical neuroscience: Computational and mathematical modeling of neural systems*. Cambridge, MA: MIT Press.

de Nó, R. L. and G. Condouris (1959). Decremental conduction in peripheral nerves: Integration of stimuli in the neuron. *Proceedings of the National Academy of Science USA 45*, 592–617.

de ruyter van Steveninck, R. R. and W. Bialek (1988). Real-time performance of a movement-sensitive neuron in the blowfly visual system: coding and information transfer in short spike sequences. *Proceedings of the Royal Society of London, Series B 234*, 379–414.

de ruyter van Steveninck, R. R., G. D. Lewen, S. P. Strong, R. Koberle, and W. Bialek (1997). Reproducibility and variability in neural spike trains. *Science 275*, 1805–1808.

Deco, G. and B. Schurmann (1998). The coding of information by spiking neurons: an analytical study. *Network: Computational Neural Systems 9*, 303–317.

Delgado-Garcia, J. M., P. P. Vidal, C. Gomez, and A. Berthoz (1989). A neurophysiological study of prepositus hypoglossi neurons projecting to oculomotor and preoculomotor nuclei in the alert cat. *Neuroscience 29*, 291–307.

Desimone, R. (1991). Face-selective cells in the temporal cortex of monkeys. *Journal of Cognitive Neuroscience 3*, 1–8.

DeWeese, M. and A. Zador (1998). Asymmetric dynamics in optimal variance adaptation. *Neural Computation 10*, 1179–1202.

Douglas, R., M. Mahowald, and C. Mead (1995). Neuromorphic analogue VLSI. *Annual Review of Neuroscience 18*, 255–281.

Douglas, R. J., C. Koch, M. Mahowald, K. A. C. Martin, and H. H. Suarez (1995). Recurrent excitation in neocortical circuits. *Science 269*, 981–985.

Durbin, R. and D. Rumelhart (1989). Product units: A computationally powerful and biologically plausible extension to backpropagation networks. *Neural Computation 1*, 133–142.

Einstein, A. (1945). *The meaning of relativity*. Princeton, NJ: Princeton University Press.

Ekekberg, O. and S. Grillner (1999). Simulations of neuromuscular control in lamprey swimming. *Philosophical Transactions of the Royal Society of London B 354*, 895–902.

Eliasmith, C. (2000). *How neurons mean: A neurocomputational theory of representational content*. PhD dissertation, Washington University in St. Louis, Department of Philosophy.

Eliasmith, C. (2001). Is the brain analog or digital?: The solution and its consequences for cognitive science. *Cognitive Science Quarterly 1*, 147–170.

Eliasmith, C. and C. H. Anderson (2001). Beyond bumps: Spiking networks that store sets of functions. *Neurocomputing 38*, 581–586.

Eliasmith, C., M. B. Westover, and C. H. Anderson (2002). A general framework for neurobiological modeling: An application to the vestibular system. *Neurocomputing 46*, 1071–1076.

Ermentrout, G. B. (1996). Type I membranes, phase resetting curves, and synchrony. *Neural Computation 8*, 979–1001.

Ermentrout, G. B. (1998a). Linearization of f-i curves by adaptation. *Neural Computation 10*, 1721–1729.

Ermentrout, G. B. (1998b). Neural networks as spatio-temporal pattern-forming systems. *Reports on progress in physics 61*, 353–430.

Ermentrout, G. B. and N. Kopell (1986). Parabolic bursting in an excitable system coupled with a slow oscillation. *SIAM Journal of Applied Mathematics 15*, 215–237.

Felleman, D. J. and D. C. V. Essen (1991). Distributed hierarchical processing in primate visual cortex. *Cerebral Cortex 1*, 1–47.

Fernández, C. and J. M. Goldberg (1976a). Physiology of peripheral neurons innervating otolith organs of the squirrel monkey. I: Response to static tilts and to long-duration centrifugal force. *Journal of Neurophysiology 39*, 970–984.

Fernández, C. and J. M. Goldberg (1976b). Physiology of peripheral neurons innervating otolith organs of the squirrel monkey. II: Directional selectivity and force-response relations. *Journal of Neurophysiology 39*, 985–995.

Fernández, C. and J. M. Goldberg (1976c). Physiology of peripheral neurons innervating otolith organs of the squirrel monkey. III: Response dynamics. *Journal of Neurophysiology 39*, 996–1008.

Field, D. J. (1996). Relations between the statistics of natural images and the response properties of cortical cells. *Journal of the Optical Society of America, A 4*, 2379–2394.

Fitzhugh, R. (1958). A statistical analyzer for optic nerve messages. *Journal of General Physiology 41*, 675–692.

FitzHugh, R. (1961). Impulses and physiological states in models of nerve membrane. *Biophysics Journal 1*, 445–466.

Fodor, J. and Z. Pylyshyn (1988). Connectionism and cognitive science: A critical analysis. *Behavioral and Brain Sciences 28*, 3–71.

Freeman, W. J. (1987). Nonlinear neural dynamics in olfaction as a model for cognition. In E. Basar (Ed.), *Dynamics of sensory and cognitive processing in the brain*. Berlin: Springer-Verlag.

Fukushima, K., C. R. S. Kaneko, and A. F. Fuchs (1992). The neuronal substrate of integration in the oculomotor system. *Progress in Neurobiology 39*, 609–639.

Fukushima, K., S. Miyake, and T. Ito (1983). Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics SMC-13*, 826–834.

Gelder, T. V. (1995). What might cognition be, if not computation? *Journal of Philosophy 91*, 345–381.

Georgopoulos, A. P., J. F. Kalasaka, M. D. Crutcher, R. Caminiti, and J. T. Massey (1984). The representation of movement direction in the motor cortex: Single cell and population studies. In G. M. Edelman, W. E. Gail, and W. M. Cowan (Eds.), *Dynamic aspects of neocortical function*. Neurosciences Research Foundation.

Georgopoulos, A. P., J. T. Lurito, M. Petrides, A. Schwartz, and J. T. Massey (1989). Mental rotation of the neuronal population vector. *Science 243*, 234–236.

Georgopoulos, A. P., A. Schwartz, and R. E. Kettner (1986). Neuronal population coding of movement direction. *Science 233*, 1416–1419.

Georgopoulos, A. P., M. Taira, and A. Lukashin (1993). Cognitive neurophysiology of the motor cortex. *Science 260*, 47–52.

Gilbert, C. D. and T. N. Wiesel (1990). The influence of contextual stimuli on the orientation of selectivity of cells in primary visual cortex of the cat. *Vision Research 30*, 1689–1701.

Giles, C. L. and T. Maxwell (1987). Learning, invariance, and generalization in high-order neural networks. *Applied Optics 26*, 4972–4978.

Gnadt, J. W. and R. A. Andersen (1988). Memory related motor planning activity in posterior parietal cortex of macaque. *Experimental Brain Research 70*, 216–220.

Goldberg, M. E. and C. B. Bruce (1990). Primate frontal eye fields III: Maintenance of a spatially accurate saccade signal. *Journal of Neurophysiology 64*, 489–508.

Goodridge, J. P. and D. S. Touretzky (2000). Modeling attractor deformation in the rodent head-direction system. *Journal of Neurophysiology 83*, 3402–3410.

Gould, J. (1975). *Honey bee communication: The dance-language controversy*. Rockefeller University, Ph.D. thesis.

Goyal, V. K., J. Kovacevic, and J. A. Kelner (2001). Quantized frame expansions with erasures. *Journal of Applied and Computational Harmonic Analysis 10*, 203–233.

Goyal, V. K., M. Vetterli, and N. T. Thao (1998). Quantized overcomplete expansions in r^n: Analysis, synthesis and algorithms. *IEEE Transactions on Information Theory 44*, 16–31.

Grandin, T. and M. J. Deesing (1998). Behavioral genetics and animal science. In T. Grandin (Ed.), *Genetics and the behavior of domestic animals*. San Diego, CA: Academic Press.

Gray, J. (1933). The movement of fish with special reference to the eel. *Journal of Experimental Biology 10*, 88–104.

Gregory, R. L. (1997). Knowledge in perception and illusion. *Philosophical Transactions Royal Society of London B 352*, 1121–1128.

Grenander, U. (1976-1981). *Lectures in pattern theory I, II, and III: Pattern analysis, pattern synthesis, and regular structures*. New York, NY: Springer-Verlag.

Grillner, S., P. Wallén, L. Brodin, and A. Lansner (1991). The neuronal network generating locomotor behavior in lamprey: Circuitry, transmitters, membrane properties, and simulation. *Annual Review of Neuroscience 14*, 169–199.

Guedry, F. E. (1974). Psychophysics of vestibular sensation. In F. E. Guedry (Ed.), *Handbook of sensory physiology: The vestibular system*, Volume 2. Berlin: Springer.

Gutkin, B. S. and G. B. Ermentrout (1998a). Dynamics of membrane excitability determine interspike interval variability: A link between spike generation mechanisms and cortical spike train statistics. *Neural Computation 10*, 1047–1065.

Gutkin, B. S. and G. B. Ermentrout (1998b). Theta-neuron, a 1-dimensional spiking model that reproduces in vitro and in vivo spiking characteristics of cortical neurons. In M. Holcombe and R. Paton (Eds.), *Information processing in cells and tissues*. New York: Plenum Press.

Gutnick, M. J. and W. E. Crill (1995). The cortical neuron as an electrophysiological unit. In M. J. Gutnick and I. Moody (Eds.), *The cortical neuron*. New York, NY: Oxford University Press.

Hakimian, S., C. H. Anderson, and T. W. Thach (1999). A PDF model of populations of purkinje cells. *Neurocomputing 26*, 169–175.

Hallinan, P. W., G. Gordon, A. L. Yuille, P. Giblin, and D. Mumford (1999). *Two- and three-dimensional patterns of the face*. Natick, MA: A. K. Peters.

Hammerstrom, D. (1995). Digital VLSI for neural networks. In M. Arbib (Ed.), *The handbook of brain theory and neural networks*. Cambridge, MA: MIT Press.

Hansel, D. and H. Sompolinsky (1998). Modeling feature selectivity in local cortical circuits. In C. Koch and I. Segev (Eds.), *Methods in neuronal modeling*. Cambridge, MA: MIT Press.

Hardcastle, V. (1997). Consciousness and the neurobiology of perceptual binding.

Seminars in Neuroscience 17, 163–170.

Hebb, D. O. (1949). *The organization of behavior*. New York, NY: Wiley.

Henneman, E. and L. Mendell (1981). Functional organization of motoneuron pool and its inputs. In V. B. Brooks (Ed.), *Handbook of physiology: The nervous system*, Volume 2. Bethesda, MD: American Physiological Society.

Hess, B. J. and D. E. Angelaki (1997). Inertial vestibular coding of motion: concepts and evidence. *Current Opinion in Neurobiology 7*, 860–866.

Hess, K., H. Reisine, and M. Dürsteler (1985). Normal eye drift and saccadic drift correction in darkness. *Neuro-ophthalmology 5*, 247–252.

Hodgkin, A. L. (1948). The local electric changes associated with repetitive action in a non-medulated axon. *Journal of Physiology (London) 107*, 165–181.

Hodgkin, A. L. and A. Huxley (1952). A quantitative description of membrane current and its application to conduction and excitation in nerves. *Journal of Physiology (London) 117*, 500–544.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences 79*, 2554–2558.

Hoppensteadt, F. and E. Izhikevich (in press). Canonical neural models. In M. Arbib (Ed.), *The handbook of brain theory and neural networks* (Second ed.). Cambridge, MA: MIT Press.

Hoppensteadt, F. C. and E. M. Izhikevich (1997). *Weakly connected neural networks.* New York, NY: Springer-Verlag.

Hubel, D. H. and T. N. Wiesel (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *Journal of Physiology (London) 160*, 106–154.

Jack, J. J. B., D. Noble, and R. W. Tsien (1975). *Electric current flow in excitable cells.* Oxford, UK: Oxford University Press.

Johnston, D. and S. M. Wu (1995). *Foundations of cellular neurophysiology*. Cambridge, MA: MIT Press.

Jordan, M. I. and T. J. Sejnowski (2001). *Graphical models: Foundations of neural computation*. Cambridge, MA: MIT Press.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME Journal of Basic Engineering 82*, 35–45.

Kang, S. M. and Y. Leblebici (1996). *CMOS digital integrated circuits: Analysis and design*. New York, NY: McGraw-Hill.

Kawato, M. (1995). Cerebellum and motor control. In M. Arbib (Ed.), *The handbook of brain theory and neural networks*. Cambridge, MA: MIT Press.

Kelly, J. P. (1991). The sense of balance. In E. R. Kandel, J. H. Schwartz, and T. M. Jessel (Eds.), *Principles of neural science*. Elsevier.

King, C. C. (1991). Fractal and chaotic dynamics in nervous systems. *Progress in Neurobiology 36*, 279–308.

Kiper, D., R. Karl, and J. Movshon (1996). Coritcal oscillatory responses do not affect visual segmentation. *Vision Research 36*, 539–544.

Kishimoto, K. and S. Amari (1979). Existence and stability of local excitations in homogeneous neural fields. *Journal of Mathematical Biology 7*, 303–318.

Knight, B. W. (1972). Dynamics of encoding in a population of neurons. *Journal of General Physiology 59*, 734–766.

Knill, D. and W. Richards (1996). *Perception as Bayesian inference*. Cambridge, UK: Cambridge University Press.

Koch, C. (1999). *Biophysics of computation: Information processing in single neurons*. New York, NY: Oxford University Press.

Koch, C. and T. Poggio (1992). Multiplying with synapses and neurons. In T. McKenna, J. Davis, and S. F. Zornetzer (Eds.), *Single neuron computation*. Boston, MA: Academic Press.

Koch, C., T. Poggio, and V. Torre (1983). Nonlinear interactions in a dendritic tree: Localization, timing and role of information processing. *Proceedings of the National Academy of Sciences USA 80*, 2799–2802.

Koch, C. and I. Segev (2001). The role of single neurons in information processing. *Nature Neuroscience Supplement 3*, 1171–1177.

Kopell, N. and G. B. Ermentrout (1988). Coupled oscillators and the design of central pattern generators. *Mathematical Biosciences 90*, 87–109.

Küpfmüller, K. and F. Jenik (1961). Über die nachrichtenverarbeitung in der nervenzelle. *Kybernetik 1*, 1–6.

Laing, C. R. and C. C. Chow (2001). Stationary bumps in networks of spiking neurons. *Neural Computation 13*, 1473–1494.

Lansner, A., O. Ekeberg, and S. Grillner (1997). Realistic modeling of burst generation and swimming in lamprey. In P. Stein, S. Grillner, A. Selverston, and D. Stuart (Eds.), *Neurons, networks, and motor behavior*. Cambridge, MA: MIT Press.

Lass, Y. and M. Abeles (1975). Transmission of information by the axon. I: Noise and memory in the myelinated nerve fiber of the frog. *Biological Cybernetics 19*, 61–67.

Lee, C., W. H. Rohrer, and D. L. Sparks (1988). Population coding of saccadic eye movements by neurons in the superior colliculus. *Nature 332*, 357–360.

Lee, D. D., B. Y. Reis, H. S. Seung, and D. W. Tank (1997). Nonlinear network models of the oculomotor integrator. In J. M. Bower (Ed.), *Computational neuroscience: Trends in research 1997*. New York: Plenum Press.

Lev, L. J. (1997). The kalman filter: Navigation's integration workhorse. *GPS World 8*, 65–71.

Lewis, F. L. (1992). *Applied optimal control and estimation*. New York, NY: Prentice-Hall.

Llinás, R. and M. Sugimori (1980). Electrophysiological properties of in vitro purkinje cell somata in mammalian cerebellar slices. *Journal of Physiology 305*, 171–195.

MacKay, D. and W. S. McCulloch (1952). The limiting information capacity of a neuronal link. *Bulletin of Mathematical Biophysics 14*, 127–135.

Mainen, Z. F. and T. J. Sejnowksi (1995). Reliability of spike timing in neocortical neurons. *Science 268*, 1503–1506.

Marder, E., N. Kopell, and K. Sigvardt (1997). How computation aids in understanding biological networks. In P. Stein, S. Grillner, A. Selverston, and D. Stuart (Eds.), *Neurons, networks, and motor behavior*. Cambridge, MA: MIT Press.

Marr, D. (1982). *Vision*. San Francisco, CA: W. H. Freeman.

Martin, J. H. (1991). Coding and processing of sensory information. In E. R. Kandel, J. H. Schwartz, and T. M. Jessel (Eds.), *Principles of neural science*. Elsevier.

Mayne, R. A. (1974). A systems concept of the vestibular organs. In H. H. Kornhuber (Ed.), *Handbook of sensory physiology: The vestibular system*. New York, NY: Springer.

Mazzoni, P., R. M. Bracewell, S. Barash, and R. A. Andersen (1996). Spatially tuned auditory responses in area LIP of macaques performing delayed memory saccades to acoustic targets. *Journal of Neurophysiology 75*, 1233–1241.

McAdams, C. J. and J. Maunsell (1999). Effects of attention on orientation-tuning functions of single neurons in macaque cortical area v4. *Journal of Neuroscience 19*, 431–441.

McCormick, D. A., B. W. Connors, J. W. Lighthall, and D. A. Prince (1985). Comparative electrophysiology of pyramidal and sparsely spiny stellate neuron. *Journal of Neurophysiology 54*, 782–806.

McGrew, W. C. (1992). *Chimpanzee material culture: Implications for human evolution*. Cambridge, UK: Cambridge University Press.

McKenna, T., J. Davis, and S. F. Zornetzer (Eds.) (1992). *Single neuron computation*. Boston, MA: Academic Press.

McNaughton, B. L., L. L. Chen, and E. J. Markus (1991). "dead reckoning", landmark learning, and the sense of direction: A neurophysiological and computational hypothesis. *Journal of Cognitive Neuroscience 3*, 190–202.

Mead, C. (1990). Neuromorphic electronic systems. *Proceedings of IEEE 78*, 1629–1636.

Mechler, F. and D. L. Ringach (2002). On the classification of simple and complex cells. *Vision Research 42*, 1017–1033.

Mel, B. (1994). Information processing in dendritic trees. *Neural Computation 6*, 1031–1085.

Mel, B. (1999). Why have dendrites? a computational perspective. In G. Stuart, N. Spruston, and M. Häusser (Eds.), *Dendrites*. New York, NY: Oxford University Press.

Merfeld, D. M., L. Zupan, and R. J. Peterka (1999). Humans use internal models to estimate gravity and linear acceleration. *Nature 398*, 615–618.

Miller, J., G. A. Jacobs, and F. Theunissen (1991). Representation of sensory information in the cricket cercal sensory system. I: Response properties of the primary interneurons. *Journal of Neurophysiology 66*, 1680–1703.

Moran, D. W. and A. B. Schwartz (1999). Motor cortical representation of speed and direction during reaching. *Journal of Neurophysiology 82*, 2676–2692.

Morris, C. and H. Lecar (1981). Voltage oscillations in the barnacle giant muscle fiber. *Biophysics Journal 35*, 193–213.

Moschovakis, A. K. (1997). The neural integrators of the mammalian saccadic system. *Frontiers of Bioscience 2*, D552–D577.

Mumford, D. (1996). Pattern theory: A unifying perspective. In D. C. Knill and W. Richards (Eds.), *Perception as Bayesian inference*. Cambridge, UK: Cambridge University Press.

Nelson, M. and J. Rinzel (1995). The hodgkin-huxley model. In J. M. Bower and D. Beeman (Eds.), *The book of Genesis: Exploring realistic neural models with the GEneral NEural SImulation System*. Santa Clara, CA: Springer-Verlag.

Nenadic, Z., C. H. Anderson, and B. Ghosh (2000). Control of arm movement using a population of neurons. In J. Bower (Ed.), *Computational neuroscience: Trends in research 2000*. Elsevier Press.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.

Nowlan, S. J. and T. J. Sejnowski (1993). Filter selection model for generating visual motion signals. In S. Hanson, J. Cowan, and L. Giles (Eds.), *Advances in neural information processing systems 5*. San Monteo, CA: Morgan Kaufmann.

Ohzawa, I., G. DeAngelis, and R. Freeman (1990). Seteroscopic depth discrimination in the visual cortex: Neurons ideally suited as disparity detectors. *Science 279*, 1037–1041.

Olshausen, B. (2000). Sparse coding of time-varying natural images. In P. Pajunen and J. Karhunen (Eds.), *Proceedings of Second International Workshop on Independent Component Analysis and Blind Signal Separation (ICA2000)*, pp. 603–608.

Olshausen, B. A., C. H. Anderson, and D. C. Van Essen (1993). A neurobiological model of visual attention and invariant pattern recognition based on dynamic routing of information. *The Journal of Neuroscience 13*, 4700–4719.

Olshausen, B. A. and D. J. Field (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature 381*, 607–609.

O'Neill, W. E. and N. Suga (1982). Encoding of target range information and its representation in the auditory cortex of the mustache bat. *Journal of Neuroscience 2*, 17–31.

Optican, L. M. and B. J. Richmond (1987). Temporal encoding of two-dimensional patterns by single units in primate primary visual cortex III: Information theoretic analysis. *Journal of Neurophysiology 57*, 162–178.

Paige, G. D. and D. L. Tomko (1991). Eye movement responses to linear head motion in the squirrel monkey. I: Basic characteristics. *Journal of Neurophysiology 65*, 1170–1182.

Paradiso, M. A. (1988). A theory for the use of visual orientation information which exploits the columnar structure of striate cortex. *Biological Cybernetics 58*, 35–49.

Partridge, L. (1966). A possible source of nerve signal distortion arising in pulse rate encoding of signals. *Journal of Theoretical Biology 11*, 257–281.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible* *inference*. Palo Alto, CA: Morgan Kaufmann.

Peterhans, E. and R. von der Heydt (1989). Mechanisms of contour perception in monkey visual cortex II: contours bridging gaps. *Journal of Neuroscience 9*, 1749–1763.

Platt, M. L. and G. W. Glimcher (1997). Responses of intraparietal neurons to saccadic targets and visual distractors. *Journal of Neurophysiology 78*, 1574–1589.

Platt, M. L. and G. W. Glimcher (1998). Response fields of intraparietal neurons quantified with multiple saccadic targets. *Experimental Brain Research 121*, 65–75.

Poggio, T. (1981). A theory of synaptic interactions. In T. Poggio and V. Torre (Eds.), *Theoretical approaches in neurobiology*. Cambridge, MA: MIT Press.

Poggio, T. (1990). A theory of how the brain might work. *Cold Spring Harbor Symposium on Quantitative Biology 55*, 899–910.

Pouget, A., S. A. Fisher, and T. J. Sejnowski (1993). Egocentric spatial representation in early vision. *Journal of Cognitive Neuroscience 5*, 150–161.

Pouget, A. and T. J. Sejnowski (1997). Spatial transformations in the parietal cortex using basis functions. *Journal of Cognitive Neuroscience 9*, 222–237.

Pouget, A., K. Zhang, S. Deneve, and P. E. Latham (1998). Statistically efficient estimation using population coding. *Neural Computation 10*, 373–401.

Rall, W. and I. Segev (1987). Functional possibilities for synapses on dendrites and dendritic spines. In G. Edelman, W. Gall, and W. Cowan (Eds.), *Synaptic function*. New York, NY: Wiley.

Ranck Jr., J. B. (1984). Head-direction cells in the deep cell layers of dorsal presubiculum in freely moving rats. *Society of Neuroscience Abstracts 10*, 599.

Rao, R. P. N. and D. H. Ballard (1997). Dynamic model of visual recognition predicts neural response properties in the visual cortex. *Neural Computation 9*, 721–763.

Rao, R. P. N. and D. H. Ballard (1999). Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience 2*, 79–87.

Redish, A. D. (1999). *Beyond the cognitive map*. Cambridge, MA: MIT Press.

Reza, F. M. (1961). *An introduction to information theory*. McGraw-Hill Electrical and Electronic Engineering Series. New York, NY: McGraw-Hill.

Richmond, B. J. and L. M. Optican (1990). Temporal encoding of two-dimensional patterns by single units in primate primary visual cortex II: Information transmission. *Journal of Neurophysiology 64*, 370–380.

Rieke, F., D. Warland, R. R. de Ruyter van Steveninick, and W. Bialek (1997). *Spikes: Exploring the neural code*. Cambridge, MA: MIT Press.

Rinzel, J. (1985). Excitation dynamics: Insights from simplified membrane models. *Federation Proceedings 44*, 2944–2946.

Rinzel, J. and G. B. Ermentrout (1989). Analysis of neural excitability and oscillations. In C. Koch and I. Segev (Eds.), *Methods in neuronal modeling*. Cambridge, MA: MIT Press.

Robinson, D. A. (1968). A note on the oculomotor pathway. *Experimental Neurology 22*, 130–132.

Robinson, D. A. (1981). Control of eye movements. In V. B. Brooks (Ed.), *Handbook of physiology: The nervous system*, Volume 2. Bethesda, MD: American Physiological Society.

Robinson, D. A. (1989). Integrating with neurons. *Annual Review of Neuroscience 12*, 33–45.

Roddey, J. C. and G. A. Jacobs (1996). Information theoretical analysis of dynamical encoding by filiform mechanoreceptors in the cricket cercal system. *Journal of Neurophysiology 75*, 1365–1376.

Rolls, E. T. and S. M. O'Mara (1995). View-responsive neurons in the primate hippocampal complex. *Hippocampus 5*, 409–424.

Romo, R., C. D. Brody, A. Hernández, and L. Lemus (1999). Neuronal correlates of parametric working memory in the prefrontal cortex. *Nature 399*, 470–473.

Rose, R. M. and J. L. Hindmarsh (1989). The assembly of ionic currents in a thalamic neuron I: The three-dimensional model. *The Proceedings of the Royal Society London, Series B 237*, 267–288.

Rosenblueth, A., N. Wiener, and J. Bigelow (1943). Behavior, purpose, and teleology. *Philosophy of Science 10*, 18–24.

Rush, M. E. and J. Rinzel (1994). Analysis of bursting in a thalamic neuron model. *Biological Cybernetics 71*, 281–291.

Salinas, E. and L. F. Abbott (1994). Vector reconstruction from firing rates. *Journal of Computational Neuroscience 1*, 89–107.

Salinas, E. and L. F. Abbott (1996). A model of multiplicative neural responses in parietal cortex. *Proceedings of the National Academy of Sciences USA 93*, 11956–11961.

Sawdai, D. and D. Pavlidis (1999). InP-based complementary HBT amplifiers for use in communication systems. *Solid State Electronics 43*, 1507–1512.

Schwartz, A. B. (1994). Direct cortical representation of drawing. *Science 265*, 540–542.

Schweighofer, N., J. Spoelstra, M. A. Arbib, and M. Kawato (1998). Role of the cerebellum in reaching movements II: A neural model of the intermediate cerebellum. *European Journal of Neuroscience 10*, 95–105.

Schwindt, P. and W. Crill (1998). Synaptically evoked dendritic action potentials in rat neocortical pyramidal neurons. *Journal of Neurophysiology 79*, 2432–2446.

Searles, E. J. and C. D. Barnes (1977). Ipsilateral utricular and semicircular canal interactions from electrical stimulation of individual vestibular nerve branches recorded in the descending medial longitudinal fasciculus. *Brain Research 125*, 23–36.

Segundo, J. P., G. P. Moore, L. J. Sensaas, and T. H. Bullock (1963). Sensitivity of neurones in aplysia to temporal pattern of arriving impulses. *Journal of Experimental Biology 40*, 643–667.

Selverston, A. I. (1980). Are central pattern generators understandable? *Behavioral and Brain Sciences 3*, 535–571.

Sereno, A. B. and J. H. R. Maunsell (1998). Shape selectivity in primate lateral intraparietal cortex. *Natuer 395*, 500–503.

Seung, H. S. (1996). How the brain keeps the eyes still. *Proceedings of the National Academy of Sciences, USA 93*, 13339–13344.

Seung, H. S., D. Lee, B. Reis, and D. Tank (2000). Stability of the memory of eye position in a recurrent network of conductance-based model neurons. *Neuron 26*, 259–271.

Seung, H. S. and H. Sompolinsky (1993). Simple models of reading neuronal population codes. *Proceedings of the National Academy of Science, USA 90*, 10740–10753.

Shadlen, M. and W. Newsome (1994). Noise, neural codes and cortical organization. *Current Opinion in Neurobiology 4*, 569–579.

Shadlen, M. and W. Newsome (1995). Is there a signal in the noise? *Current Opinion in Neurobiology 5*, 248–250.

Shamma, S. (1989). Spatial and temporal processing in central auditory networks. In C. Koch and I. Segev (Eds.), *Methods in neuronal modeling*. Cambridge, MA: MIT Press.

Shevelev, I. A. (1998). Second-order features extraction in the cat visual cortex: selective and invariant sensitivity of neurons to the shape and orientation of crosses and corners. *Biosystems 48*, 195–204.

Shevelev, I. A., K. U. Jirmann, G. A. Sharaev, and U. T. Eysel (1998). Contribution of GABAergic inhibition to sensitivity to cross-like figures in striate cortex. *Neuroreport 9*, 3153–3157.

Shields, F. C. (1968). *Elementary linear algebra*. New York, NY: Worth Publishers.

Skaggs, W. E., J. J. Knierim, H. S. Kudrimoti, and B. L. McNaughton (1995). A model of the neural basis of the rat's sense of direction. In *Advances in Neural Information Processing Systems 7*, pp. 173–180. Cambridge, MA: MIT Press.

Skarda, C. J. and W. J. Freeman (1987). How brains make chaos in order to make sense of the world. *Behavioral and Brain Sciences 10*, 161–195.

Snippe, H. P. and J. J. Koenderink (1992). Information in channel-coded systems: correlated receivers. *Biological Cybernetics 66*, 543–551.

Snyder, L. (1999). This way up: Illusions and internal models in the vestibular system. *Nature Neuroscience 2*, 396–398.

Snyder, L. H., A. P. Batista, and R. A. Andersen (1997). Coding of intention in the posterior parietal cortex. *Nature 386*, 167–170.

Softky, W. (1995). Simple codes versus efficient codes. *Current Opinion in Neurobiology 5*, 239–247.

Softky, W. and C. Koch (1993). The highly irregular firing of cortical cells is inconsistent with the temporal integration of random EPSPs. *Journal of Neuroscience 13*, 334–350.

Softky, W. and C. Koch (1995). Single-cell models. In M. Arbib (Ed.), *The handbook of brain theory and neural networks*. Cambridge, MA: MIT Press.

Spoendlin, H. H. (1966). The ultrastructure of the vestibular sense organ. In R. J. Wolfson (Ed.), *The vestibular system and its diseases*. Philadelphia: University of Pennsylvanina Press.

Stevens, C. F. (1994). What form should a cortical theory take? In C. Koch and J. L. Davis (Eds.), *Large-scale neuronal theories of the brain*. Cambridge, MA: MIT Press.

Stevens, C. F. (2000). Models are common; good theories are scarce. *Nature Neuroscience 3*, 1177.

Stevens, C. F. and Y. Wang (1994). Changes in reliability of synaptic function as a mechanism for plasticity. *Nature 371*, 704–707.

Stevens, C. F. and A. M. Zador (1995). Neural coding: The enigma of the brain. *Current Biology 5*, 1370–1371.

Stevens, C. F. and A. M. Zador (1998). Input synchrony and the irregular firing of cortical neurons. *Nature Neuroscience 1*, 210–217.

Stevens, C. F. and Z. Zador (1996). Information through a spiking neuron. In *Advances in Neural Information Processing Systems 8*. Cambridge, MA: MIT Press.

Sumer, B. M. and J. Fredsøe (1997). *Hydrodynamics around cylindrical structures*. Freemont, CA: World Scientific.

Tal, D. and E. L. Schwartz (1997). Computing with the leaky integrate-and-fire neuron: Logarithmic computation and multiplication. *Neural Computation 9*, 305–318.

Taube, J. S., R. U. Muller, and J. B. Ranck Jr. (1990a). Head-direction cells recorded from the postsubiculum in freely moving rats. I. description and quantitative analysis. *Journal of Neuroscience 10*, 420–435.

Taube, J. S., R. U. Muller, and J. B. Ranck Jr. (1990b). Head-direction cells recorded from the postsubiculum in freely moving rats. II. effects of environmental manipulations. *Journal of Neuroscience 10*, 436–447.

Telford, L., S. H. Seidman, and G. D. Paige (1997). Dynamics of squirrel monkey linear vestibuloocular reflex and interactions with fixation distance. *Journal of Neurophysiology 78*, 1775–1790.

Thomson, A. M. and J. Deuchars (1997). Synaptic interactions in neocortical local circuits: dual intracellular recordings in vitro. *Cerebral Cortex 7*, 510–522.

Todorov, E. (2000). Direct cortical control of muscle activation in voluntary arm movements: A model. *Nature Neuroscience 3*, 391–398.

Torre, V. and T. Poggio (1978). A synaptic mechanism possibly underlying directional selectivity to motion. *Proceedings of the Royal Society of London, B 202*, 409–416.

Touretzky, D. S. and A. D. Redish (1996). Theory of rodent navigation based on interacting representations of space. *Hippocampus 6*, 247–270.

Tovee, M. J., E. T. Rolls, A. Treves, and R. P. Bellis (1993). Information encoding and the responses of single neurons in the primate temporal visual cortex. *Journal of Neurophysiology 70*, 640–654.

Van Essen, D. C. and C. H. Anderson (1995). Information processing strategies and pathways in the primate visual system. In S. F. Zornetzer, J. L. Davis, C. Lau, and T. McKenna (Eds.), *An introduction to neural and electronic networks*. Academic Press.

Van Essen, D. C., C. H. Anderson, and D. J. Felleman (1992). Information processing in the primate visual system: An integrated systems perspective. *Science 255*, 419–423.

van Gelder, T. and R. Port (1995). It's about time: An overview of the dynamic approach to cognition. In R. Port and T. van Gelder (Eds.), *Mind as motion: Explorations in the dynamics of cognition*. Cambridge, MA: MIT Press.

von der Heydt, R., E. Peterhans, and M. Dursteler (1991). Grating cells in monkey visual cortex: Coding texture? In B. Blum (Ed.), *Channels in the visual nervous system: Neurophysiology, psychophysics, and models*. London: Freund.

Wadden, T., J. Hellgren, A. Lansner, and S. Grillner (1997). Intersegmental coordination in the lamprey: Simulations using a network model without segmental boundaries. *Biological Cybernetics 76*, 1–9.

Wannier, T., T. G. Deliagina, G. N. Orlovsky, and S. Grillner (1998). Differential effects of the reticulospinal system on locomotion in lamprey. *Journal of Neurophysiology 80*, 103–112.

Watanabe, S. (1965). Karhunen-Loève expansion and factor analysis: Theoretical remarks and applications. In *Transactions of the Fourth Prague Conference on Information Theory, Statistical Decision Functions, and Random Processes*, pp. 635–660.

Watson, R. A. (1995). *Representational ideas: From Plato to Patricia Churchland*. New York, NY: Kluwer Academic Publishers.

Wehmeier, U., D. Dong, C. Koch, and D. C. Van Essen (1989). Modeling the mammalian visual system. In C. Koch and I. Segev (Eds.), *Methods in neuronal modeling*. Cambridge, MA: MIT Press.

Weiner, N. (1948). *Cybernetics: or control and communication in the animal and the machine*. Cambridge, MA: MIT Press.

Wessel, R., C. Koch, and F. Gabbiani (1996). Coding of time-varying electric filed amplitude modulations in a wave-type electric fish. *Journal of Neurophysiology 75*, 2280–2293.

Westover, M. B., C. Eliasmith, and C. H. Anderson (2002). Linearly decodable functions from neural population codes. *Neurocomputing 45*, 691–696.

Wilson, H. R. (1999a). Simplified dynamics of human and mammalian neocortical neurons. *Journal of Theoretical Biology 200*, 375–388.

Wilson, H. R. (1999b). *Spikes, decisions, and actions: dynamical foundations of neuroscience*. New York, NY: Oxford University Press.

Wilson, V. J. and G. M. Jones (1979). *Mammalian vestibular physiology*. New York, NY: Plenum Press.

Wolpert, D. M. and M. Kawato (1998). Multiple paired forward and inverse models for motor control. *Neural Networks 11*, 1317–1329.

Yokota, J., H. Reisine, and B. Cohen (1992). Nystagmus induced by electrical stimulation of the vestibular and prepositus hypoglossi nuclei in the monkey: evidence for site of induction of velocity storage. *Experimental Brain Research 92*, 123–138.

Yuille, A. L. and J. J. Clark (1993). Bayesian models, deformable templates and competitive priors. In L. Harris and M. Jenkins (Eds.), *Spatial vision in humans and robots*. Cambridge, UK: Cambridge University Press.

Zador, A., B. Claiborne, and T. Brown (1992). Nonlinear pattern separation in single hippocampal neurons with active dendritic membrane. In J. Moody, S. Hanson, and R. Lippmann (Eds.), *Advances in neural information processing systems 4*. San Monteo, CA: Morgan Kaufmann.

Zador, A. M. (1998). Impact of synaptic unreliability on the information transmitted by spiking neurons. *Journal of Neurophysiology 79*, 1219–1229.

Zhang, K. (1996). Representation of spatial orientation by the intrinsic dynamics of the head-direction cell ensemble: A theory. *Journal of Neuroscience 16*, 2112–2126.

Zipser, D. and R. A. Andersen (1988). A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons. *Nature 331*, 679–684.

Index