Part I

How to build a brain

Chapter 2

An introduction to brain building

Before turning to my main purpose of answer the four questions regarding semantics, syntax, control, and memory and learning, I need to provide an introduction to the main method I will be relying in providing answers to these questions. As I argued in the last chapter, we need to provided biologically-based answers to these questions, and we need to provide answers that will let us build such systems. In the sections that follow, I provide an overview of the relevant biology, and introduce the Neural Engineering Framework (NEF), which provides basic methods for constructing large-scale neurally realistic models of the brain. Perhaps this discussion will be valuable in its own right. But, in fact, its purpose is to lay a foundation for what is to come: a neural architecture for biological cognition.

2.1 Brain parts

- more pictures?

- references to Jove?

Brains are absolutely fantastic devices. They are relatively small compared to the size of our bodies. A typical brain weighs between 1 and 2 kg and comprises only

2% of our body weight. Nevertheless, brains contain about 25% of the energy used by the body. This is especially surprising when you consider the serious energy demands of muscles, which must do actual physical work. Presumably, since the brain is such a power hog, it is doing something important for our survival or it would not have been preserved by evolution. Presumably, that "something important" is somewhat obvious: brains control the four Fs (feeding, fleeing, fighting, and reproduction); brains provide animals with behavioral flexibility that is unmatched by our most sophisticated machines; and brains are constantly adapting to be uncertain, noisy, and rapidly changing world in which they find themselves embedded. Perhaps most surprising of all is that brains do all of these things while consuming only about 20 W of power – the equivalent of a compact fluorescent lightbulb. To put this power efficiency in some perspective, the world's most powerful supercomputer "roadrunner" at Los Alamos labs in the United States, which, as far as we know, is unable to match the computational power of the mammalian brain, consumes 2.35 MW (about 100,000 times more).

We often think of this incredibly efficient device is something like a soft pillow crammed inside our skulls. While the texture of brains has often been compared to that of a thick pudding, it is more accurate to think of the brain as being a large sheet, equivalent in size to about four sheets of writing paper, and about 3 mm thick. In almost all animals, the sheet has six distinct layers which are composed mostly of the cell bodies of neurons, very long thin processes used for communication, and glial cells, which are a very prevalent but poorly understood companion to neurons. In each square millimeter of human cortex there are crammed about 100,000 neurons.[1] So, there are about 25 billion neurons in human cortex. Overall, however, there are approximately 100 billion neurons in the human brain. The additional neurons come from "subcortical" areas, which include cerebellum, basal ganglia, thalamus, and the brainstem, among others. To get a perspective on the special nature of our brains, it is worth noting that monkey brains are approximately the size of one sheet of paper, and rats have brains the size of a Post-it note.

In general, it is believed that what provides brains with their impressive computational abilities is the organization of the connections among individual neurons. These connections allow cells to collect, process, and transmit information. In fact, neurons are specialized precisely for communication. In most respects, neurons are exactly like other cells in our bodies, they have a cell membrane, a

[1] **These and many other brain facts can be found at: http://faculty.washington.edu/chudler/facts.html**

nucleus, and they have similar metabolic processes. What makes neurons stand out under a microscope, are the many branching processes which project outwards from the somewhat bulbous, main cell body (figure???). These processes are there to enable short and long distance communication with other neural cells. This is a hint: if we want to understand how brains work, we need to have some sense of how neurons communicate in order to compute.

Figure 2.1 outlines the main components and activities of the mechanisms underlying cellular communication. The cellular processes that carry information to the cell body are called dendrites. The dendrites carry signals, in the form of an ionic current to the main cell body. If sufficient current enters the cell body at a given point in time, a series of nonlinear events will be triggered that result in an action potential, or "spike," that will proceed down the output process of the neuron, which is call and axon. Neural spikes are very brief events, lasting for only a few milliseconds, which travel in a wave-like fashion down the axon until they reach the end of the axon, called the bouton, where they cause the release of tiny packets of chemicals called neurotransmitters. Axons end near the dendrites of subsequent neurons. Hence, neurotransmitters are released into the small space between axons and dendrites, which is called the synaptic cleft. The neurotransmitters very quickly cross the cleft and bind to special proteins in the cell membrane of the next neuron. This binding causes small gates, or channels, in the dendrite of the next neuron to open, which allows charged ions to flow into the dendrite. These ions result in a current signal in the receiving dendrite, which flows to the cell body of a subsequent neurons and so the process continues.

A slightly simpler description of this process, but one which retains the central relevant features, is as follows:

1. Signals flow down the dendrites of a neuron into its cell body.

2. If the overall input to the cell body from the dendrites crosses a threshold, the neuron generates a stereotypical spike that travels down the axon.

3. When the spike gets to the end of the axon, it causes chemicals to be released that travel to the connected dendrite and, like a key into a lock, cause the opening of channels that produce a current in the receiving dendrite.

4. This current is the signal in step 1., that then flows to the cell body of the next neuron.

As 'clean' as this story sounds, it is made much more complex in real brains by a number of factors. First, neurons are not all the same. There are hundreds of

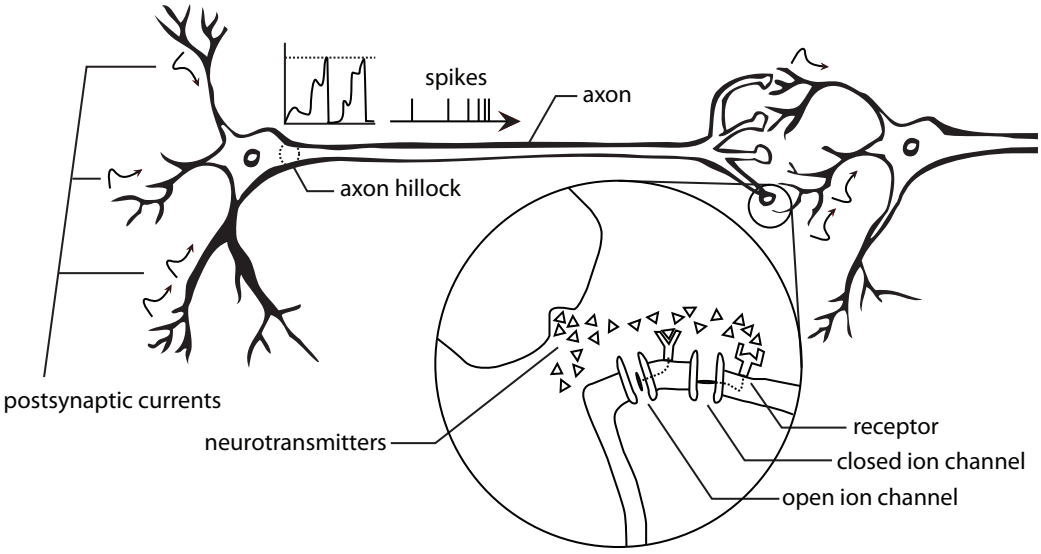

Figure 2.1: Cellular communication. This figure diagrams the main components and kinds of activities of a majority of neural cells during communication. The flow of information begins at the left side of the image with postsynaptic currents travelling along dendrites towards the cell body of the neuron. At the axon hillock, these currents are added together and a spike is generated if the sum is above the neuron's threshold. The spikes travel along the axon and cause the release of neurotransmitters at the end of the axon. These neurotransmitter bind to matching receptors on the dendrites of nearby neurons causing ion channels to open; it is significant that receptors must match the specific neurotransmitters present, otherwise the ion channel remains closed. Following the opening of ion channels, postsynaptic currents are induced in the receptor neuron and the process continues.

different kinds of neurons that have been identified in mammalian brains. The neurons can range in size from 10^-4 to 5 m in length. The number of inputs to a cell can range from about 500 or fewer, to well over 200,000. The number of outputs, that is, branches of a single axon, cover a similar range. Given all of these connections, it is not surprising to learn that there are approximately 72 km of fiber in the human brain. Finally, there are hundreds of different kinds of neurotransmitters and many different kinds of receptors. Different combinations of neural transmitters and receptors, can cause different kinds of currents to flow in the dendrite. As a result, a single spike transmitted down an axon can be received by many different neurons, and can have different kinds of effect on each neuron depending on the mediating neurotransmitter and receptors.

The variability found in real biological networks makes for an extremely complex system, one which, at first glance, seems designed to frustrate our analysis. Typically, in mathematics, homogeneous systems (those that look similar no matter where you are in the system) are much easier to understand. ???Paul Smolensky, a leader in the field of artificial neural networks explicitly says connectionism should be homogeneous (1988: p. 1)??? The brain, in contrast, is clearly highly heterogeneous: there are hundreds of kinds of neurons, many with different kinds of intrinsic dynamical properties; even neurons of the same kind often respond differently, i.e., generate different patterns of spikes, to exactly the same current injected into their soma; and even neurons that shared response properties and type, can still differ in the number and kinds of channels in their dendrites, meaning that the same spikes coming from preceding neurons will generate different responses.

Experimental neuroscientists who record the activity of single neurons in response to perceptual stimuli shown to an animal, will tell you that no two neurons seem to respond in the same manner. This kind of variability is captured in terms of a neurons 'tuning curve.' An example tuning curve from a cell in primary visual cortex is shown in figure 2.2. In general, I tuning curve is a graph that shows the frequency of spiking of a neuron in response to a given input stimuli. In this figure, we can see that as the orientation of a presented bar changes, the neuron responds more or less strongly. Its peak response happens at about ??? degrees. Nevertheless, there is some information about the orientation of the stimulus available from this activity whenever the neuron responds. This tuning curve, while typical of cells in primary visual cortex, includes an additional implicit assumption. That is, this is not the response of the cell to any oriented bar at any position in the visual field. Rather, it is a response to an oriented bar in what is called the 'receptive field' of the cell. Since this is a visual cell, the receptive field indicates which part

Figure 2.2: A tuning curve from a cell in primary visual cortex (V1). The x-axis indicates the angle of rotation of a bar, and the y-axis indicates the firing rate of the neuron when shown that stimulus. The graph is a result of many trials at each rotation, and hence has error bars that indicate the standard deviation of the response (reproduced with permission??? from Eliasmith & Anderson, 2003).

of possible visual input space the neuron seems to care about. So, to completely capture the 'tuning' of the cell to visual stimuli, we ideally want to combine the traditionally distinguished receptive field and tuning curve. In the remainder of this book, I will use the notion of tuning curve in this more general sense. That is, what I subsequently defined as a tuning curve often includes information about the receptive field of the cell in question. Returning to the issue of heterogeneity, the reason that experimental neuroscientists would suggest that no two cells are the same, is precisely because their tuning curves (coupled with their receptive fields) seem to never perfectly overlap. As a result, while neurons near those shown in figure 2.2 may share similar tuning properties, the peak, width, and rolloff of the graph will be somewhat different.

We can begin to get a handle on the heterogeneity observed in neural systems by noticing that there seemed to be to sources for the variability: intrinsic; and extrinsic. Intrinsic heterogeneity, such as different responses to the same injected current, will have to be captured by variability in the models of individual cells. Extrinsic heterogeneity, such as the variability and tuning curves, can be understood more as a consequence of where a particular cell sits in the network. That is, the reason neighboring cells have different tuning curves is not merely because of intrinsic heterogeneity, but also because they are receiving slightly different inputs than their neighbors. So, even if all of the cells were intrinsically homogeneous, their tuning curves might look very different depending on the processing of the cells before them, to which they are connected. This difference between extrinsic and intrinsic heterogeneity will prove important in understanding the sources of different kinds of dynamics observed in real neural networks.

Of course, this distinction does not in any way mitigate the fact of heterogeneity itself. We must still overcome the observed diversity of neural systems, which has traditionally been seen as a barrier to theoretical analysis. In the next sections we will see that despite this complexity, it is possible to suggest and quantify underlying principles which do a good job of describing the functional properties of neural networks that display broad heterogeneity. In fact, we can come to understand why this massive heterogeneity that we observe might provide for more

robust computations than those of a system whose individual components were identical (see section 2.3.1).

- any qualifications to this simple picture and how many of the methods discussed subsequently take those into account: e.g. additional neural dynamics (bursting, adaptation), nonlinear dendritic effects, Dale's principle, many of the other 'heterogeneities' that exist in natural neural systems (citations to papers that deal with each of these... & perhaps short descriptions here in brackets). I don't go into those it too much detail here for reasons of brevity. I will introduce them to the extent they matter to specific example models described later on....perhaps?

2.2 Theoretical neuroscience

In recent years, most of the behavioral sciences have seen a heavy influx of influence from the neurosciences. Most obviously, the psychology of a mere two decades ago was almost bereft of brain-related talk. Now, however, one of the fastest growing areas of psychology is cognitive neuroscience, whose practitioners regularly talk of neuroanatomical features, neurotransmitters, and employ various brain imaging and measurement techniques, like fMRI and EEG. However, the same trend is beginning to appear in linguistics, economics (as behavioral economics), and even philosophy (as experimental philosophy, aka 'X-phi').

Despite this tendency of neuroscience to become ubiquitous, it would be a mistake to think that neuroscience is in any sense monolithic. Quite the contrary: the tens of thousands of posters and presentations at the Society for Neuroscience meeting held every year are divided in to many sections, such as molecular neuroscience, cellular neuroscience, systems neuroscience, neuroanatomy, developmental neuroscience, behavioral neuroscience, and cognitive neuroscience among others. Often, the researchers in one area are not able to understand or communicate effectively with those from another, despite the fact that ultimately, everyone in the building wants to understand how the brain works.

This, of course, is not a problem unique to neuroscience. Many other areas of biology suffer the same difficulties, as do other sciences such as geology, chemistry, and even physics. However, one main advantage that, for example, physics has in mitigating these challenges is a widely shared technical vocabulary for describing both problems and solutions in the field. The development and application of these quantitative tools have helped physics develop rapidly, leading us to

exciting new discoveries, as well as deep, challenging problems. The subfield of physics most centrally concerned with the development of such tools is theoretical physics.

Interestingly, an analogous subfield has been developing in neuroscience over the last few decades as well, and it is often appropriately called 'theoretical neuroscience' (though perhaps equally as often called 'computational neuroscience'). In fact, the analogy between theoretical neuroscience and theoretical physics is both strong, and useful for understanding its importance to neuroscience. For instance, both are centrally interested in quantifying the phenomena under study. This does not mean merely statistically quantifying the data generated by the phenomena, but rather coming up with quantitative descriptions of the deterministic regularities and mechanisms giving rise to that data. Take, for instance, one of the greatest advances in theoretical physics, the development of Newton's three laws of motion. The second, perhaps most famous, law is that "The alteration of motion is ever proportional to the motive force impressed; and is made in the direction of the right line in which that force is impressed" Newton (1729, p. 19). In short form: $F = ma$. The purpose of this statement is to make a clear, straightforward hypothesis about motion. I describe similar principles in the sections 2.3.1-2.3.3 for neural representation, computation, and dynamics.

A second analogy between these two subfields is that both are interested in summarizing an enormous amount of data. Newton's second law is intended to apply to all forms of motion, be it rectilinear, circular, or what have you. Measuring all such forms of motion, and describing the results statistically would not be nearly as concised. Similarly, theoretical neuroscientists are attempting to understand the basic principles behind neural function. Often, they would like their mathematical descriptions to be as general as possible, although there is some debate regarding whether or not the kind of unification being strived for in physics should be a goal for theoretical neuroscience.

A third crucial analogy between theoretical physics and theoretical neuroscience is that the disciplines are speculative. Most such quantitative descriptions of general mechanisms go beyond the available data. As such, they almost always suggest more structure than the data warrants, and hence more experiments to perform. Looking again to Newton's second law, he intended it to be true for all velocities. However, special relativity has subsequently demonstrated that the law is measurably violated for large velocities. With speculation comes risk. Especially for a young field like theoretical neuroscience, the risk of being wrong is high. But, the risk of becoming lost in the complexity of the data without such speculation is much higher.

Of course, there are crucial disanalogies between these fields as well. Perhaps, physics can be succinctly described as being interested in questions of *what there is*, while neuroscience is interested in *who we are*. As well, neuroscience is a much younger discipline than physics, and so the methods for making measurements of the systems of interest are still rapidly developing.

Nevertheless, the similarities can help us understand why theoretical neuroscience is important for the development of neuroscience. Like theoretical physics, theoretical neuroscience can help in at least two crucial ways. Specifically, it should:

1. Quantify, and hence make more precise and testable, hypotheses about the functioning of neural systems

2. Summarize large amounts of experimental data, and hence serve to unify the many sources of data from different 'neuro' and behavioral subdisciplines

The first of these stem from the commitment to using mathematics to describe principles of neural function. The second is crucial for trying to deal with the unavoidable complexities of neural systems. This is a challenge not faced to the same degree by many physicists.[2] Characterizing a system of billions of parts as if each is identical, and as if the connections between all of them are approximately the same can lead to very accurate characterizations of physical systems (e.g. the ideal gas law). The same is not true of neural systems. Large neural systems where all of the parts are the same, and interact in similar ways simply do not exist.

Thus, the links between the 'lowest' and 'highest' levels of characterizing the system are complex and unavoidable. It is perhaps in this kind of circumstance that quantification of neural systems plays its most crucial role. If we can state our quantitative hypotheses about neural function at a 'high' level, and quantify the relationship between levels, then our high-level hypothesis will connect to low-level details. In fact, ideally, a hypothesis at any level should contact data at all other levels. It is precisely this kind of unification of experimental data that is desperately needed in the behavioral sciences to support cross-(sub)disciplinary communication.

This ideal role for theoretical neuroscience has not yet been realized. I believe this is because there has been a focus on 'low' levels of neural systems (i.e. single

[2]This observation should make it obvious that I am in no way suggesting the behavioral sciences should become physics, a mistake which has been made in the past Carnap (1931).

cells or small networks). I also think this is perfectly understandable in light of the complexity of the system being tackled. Nevertheless, I also think we are now in a position to begin to move past this state of affairs.

So, in the context of this book, I'm adopting methods from theoretical neuroscience because of their benefits, and at the same time prodding theoretical neuroscience to expand the viable areas of application to all of the behavioral sciences. To do so, in the next section I begin by introducing a series of theoretical principles developed in the context of traditional theoretical neuroscience. In the next chapter I suggest a way of applying these same principles to large-scale, cognitive modeling. This should help to not only test, but also to refine such principles, and it should make our cognitive models subject to data from all the various subdisciplines of the behavioral sciences.

2.3 A framework for building a brain

In 2003, Charles H. Anderson and I wrote a book called *Neural Engineering*. In it, we presented and demonstrated a mathematical theory of how biological neural systems could implement a wide variety of dynamic functions. As with most work in theoretical neuroscience, we focussed on 'low-level' systems, including parts of the brainstem involved in controlling stable eye position, parts of the inner ear and brainstem for controlling a vestibulo-ocular reflex (VOR), and spinal circuits in the lamprey for controlling swimming.

In more recent work, these methods have been used by us and others to propose novel models of a wider variety of neural systems, including the barn owl auditory system (Fischer, 2005; Fischer et al., 2007), parts of the rodent navigation system (Conklin and Eliasmith, 2005), escape and swimming control in zebrafish (Kuo and Eliasmith, 2005), tactile working memory in monkeys (Singh and Eliasmith, 2006), and simple decision making in humans (Litt et al., 2008) and rats (ref Laubach???), among others. We have also used these methods to better understand more general issues about neural function, such as how neural systems might perform time derivatives (Tripp and Eliasmith, 2010), how the variability of neural spike trains and the timing of individual spikes relates to information that can be extracted from spike patterns (Tripp and Eliasmith, 2007), how we can ensure that biological constraints such as Dale's Principle – the principle that a

given neuron typically has either excitatory or inhibitory effects but not both – are respected by neural models (Parisien et al., 2008). ???Also new learning paper ref???.

One reason the NEF has such broad application is because it does not make assumptions about what the brain actually does. Rather, it is a set of three principles that can help determine *how* the brain performs a given function. For this reason, John Miller once suggested that the NEF is a kind of 'neural compiler'. If you have a guess about the high-level function of the system you are interested in, and you know (or assume) some information about how individual neurons respond to relevant input, the NEF provides a way of connecting populations of neurons together to realized that function. This, of course, is exactly what a compiler does in computer science. The programmer specifies a program in a high-level language like Java. The Java compiler knows something about the low-level machine language implemented in a given chip, and it translates that high-level description into an appropriate low-level one.

Of course, things are not so clean in neurobiology. We do not have a perfect description of the machine language, and our high-level language may be able to define functions that we cannot actually implement in neurons. Consequently, building models with the NEF can be an iterative or bootstrapping process: first you gather data from the neural system and have a hypothesis about what it does; then you build a model and see if it behaves like the real system; then, if it does not behave consistently with the data, you alter your hypothesis or perform experiments to figure out why the two are different. Often, such models will behave in ways that there is no data speak to. In these lucky cases, you can make a prediction and perform an experiment. Of course this process is not unique to the NEF. Instead, it will be familiar to any modeller. What the NEF offers is a systematic method of performing these steps.

It is worth emphasizing that, also like a compiler, the NEF does *not* specify what the system does. This specification is brought to the characterization of the system by the modeler (or programmer). In short, the NEF is about *how* brains compute, not *what* they compute. The bulk of this book is about the 'what,' but those considerations do not begin until chapter 3.

In the remainder of this section, I provide an outline of the three principles of the NEF. There are, of course, many more details (at least a book's worth!). So, a few points are in order. First, the methods here are by no means the sole invention of our group. We have drawn heavily on other work in theoretical neuroscience. To keep this description as brief as possible, I refer the reader to other descriptions of the methods that better place the NEF in its historical context (Eliasmith

and Anderson, 2003; Eliasmith, 2005; Tripp and Eliasmith, 2007). Second, the original *Neural Engineering* book is a useful source for far more mathematical detail than I provide here. However, the framework has been extended recently, so that book should be taken as a starting point. Finally, some mathematics can be useful for interested readers, but I have placed most of it in the appendices to emphasize a more intuitive grasp of the principles. This is at least partially because the Nengo neural simulator has been designed to handle the mathematical detail, allowing the modeller to focus effort on capturing the neural data, and the hypothesis she or he wishes to test.

The following three principles describe the NEF:[3]

1. Neural representations are defined by the combination of nonlinear encoding (exemplified by neuron tuning curves, and neural spiking) and weighted linear decoding (over populations of neurons and over time).

2. Transformations of neural representations are functions of the variables represented by neural populations. Transformations are determined using an alternately weighted linear decoding.

3. Neural dynamics are characterized by considering neural representations as state variables of dynamic systems. Thus, the dynamics of neurobiological systems can be analyzed using control (or dynamics systems) theory.

In addition to these main principles, the following addendum is taken to be important for analyzing neural systems:

- Neural systems are subject to significant amounts of noise. Therefore, any analysis of such systems must account for the effects of noise.

The ubiquity of noise in neural systems is well documented, be it from synaptic unreliability (Stevens and Wang, 1994; Zucker, 1973), variability in the amount of neurotransmitter in each vesicle (Burger et al., 1989), or jitter introduced by axons into the timing of neural spikes (Lass and Abeles, 1975). Consequently, there are limits on how much information can be passed by neurons: it seems that neurons tend to encode approximately 2-7 bits of information per spike (Bialek and Rieke, 1992; Rieke et al., 1997). Here I do not consider this addendum separately, though it is included in the formulation of each of the subsequent principles, and their implementation in Nengo.

[3]For a quantitative statement of the three principles, see Eliasmith and Anderson (2003, pp. 230-231).

Let me now consider each of the three principles in turn. To make the application of these principles concrete, I adopt the example of a 'controlled integrator'. This neural circuit can be thought of as a simple memory circuit which can be loaded with a memory, hold the memory over time, and then erase the memory (see figure 2.5).

2.3.1 Representation

A central tenet of the NEF is that we can adapt the information theoretic account of *codes* to understanding representation in neural systems. 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.

In order to characterize representation in a neural system, we can identify each of these procedures and their relevant alphabets. The encoding procedure is straightforward to identify: it is the mapping of stimuli into a series of neural spikes. Indeed, encoding is what neuroscientists typically talk about, and what I have covered in section 2.1. When we show a brain a stimulus, some neurons or other 'fire' (see the spike rasters in figure 2.4 for a simple example). The precise nature of this encoding has been explored in-depth via quantitative models (see Appendix A.1.1).

Unfortunately, neuroscientists often stop here in their characterization of representation, but this is insufficient. We also need to identify a decoding procedure, otherwise there is no way to determine the relevance of the encoding for the system. If no information about the stimulus can be extracted from the spiking neurons, then it makes no sense to say that it represents the stimulus. Representations, at a minimum, must potentially be able to 'stand-in' for the things they represent.

Quite surprisingly, despite typically nonlinear encoding (i.e., mapping a continuously varying parameter like stimulus intensity into a bunch of discontinuous spikes), a good linear decoding can be found.[4] And, there are several established

[4] As demonstrated by F. Rieke, D. Warland, R. de Ruyter van Steveninick, and W. Bialek *Spikes: Exploring the Neural Code* (Cambridge: MIT Press, 1997), pp. 76-87.

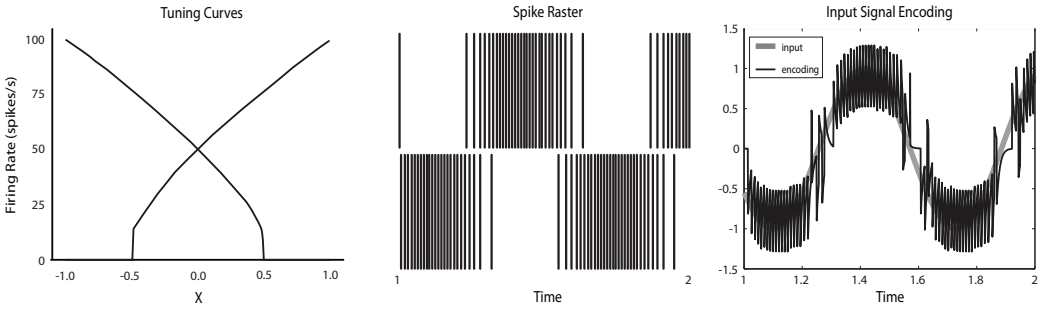

Figure 2.3: Encoding and decoding of an input signal by two neurons. The neurons fire at rates specified by their tuning curves, shown here to be symmetric. With a sinusoidally varying input, these tuning curves cause the neurons to fire as shown in the spike raster. The PSCs are them summed in a receiving neuron to give an estimate, plotted in black, of the original input signal, plotted in gray. The estimate is poor, but can be made much better with additional neurons (see figure 2.4).

methods for determining linear decoders given the the neural populations that respond to certain stimuli (see Appendix A.1.2 for one). Notably, these decoders must decode time-varying signals over a population of partially redundant neurons. Thus they are determined by both temporal and 'population' aspects of the encoding.

The temporal aspects are determined by the biophysics of cellular communication. In short, the post-synaptic current (PSC) discussed in section 2.1, is used to convert spikes into an estimate of the input signal (see figure 2.3).

The population aspects are determined by weighting each neural response, depending on how good it is for representing the input signal. That is, each neuron is 'weighted' by how useful it is for carrying information about the stimulus in the context of the whole population. If it's very useful, it has a high weight, if not it has a low weight. Finding decoders in the NEF is accomplished by optimally weighting neurons (see Appendix A.1.2). Thus, many neurons can 'cooperate' to give very good representations of time-varying input signals (see figure 2.4).

Having specified the encoding and decoding procedures, we still need to specify the relevant alphabets. While the specific cases will diverge greatly, we can describe the alphabets generally: neural responses (encoded alphabet) code physical

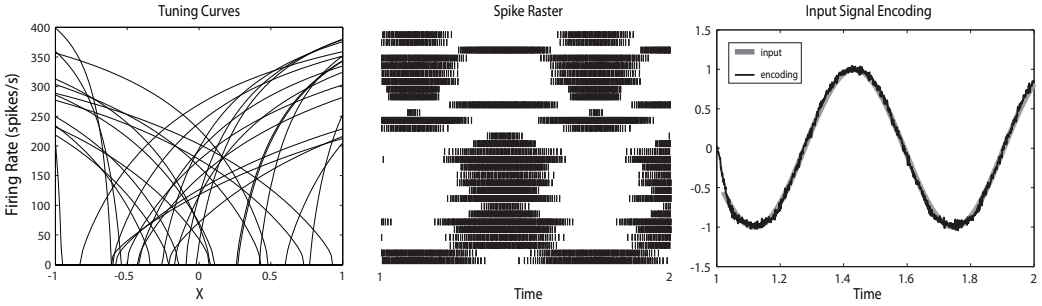

Figure 2.4: Encoding and decoding of an input signal by 30 neurons. As above in 2.3, the tuning curves of the neurons characterize their general response to input, the spike raster depicts the response to a specific input signal, and the final encoding sums PSCs to estimate the input signal. The temporal decoders are PSCs and the population decoders are optimal weights.

properties (decoded alphabet). Slightly more specifically, the encoded alphabet is the set of temporally patterned neural spikes over populations of neurons. This is reasonably uncontroversial.

However, it is much more difficult to be 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 use to describe the physical world do seem to be represented in nervous systems: there are neurons sensitive to displacement, velocity, acceleration, wavelength, temperature, pressure, mass, etc. But, there are many physical properties not discussed by physicists that also seem to be encoded in nervous systems: such as red, hot, square, dangerous, edible, object, conspecific, etc.[5] It is reasonable to begin with the hypothesis that these 'higher-level' properties are inferred on the basis of representations of properties more like those that physicists talk about.[6]

[5]I am allowing any property reducible to or abstractable from physical properties to count as physical. My suspicion is that this captures all properties, but that is a conversation for another book. Notice, however, that many typical physical properties, like temperature, are both abstracted from lower-level properties and ar part of physics. So, abstracting from physical properties does not make properties non-physical. In philosophical terms, this is the claim that properties 'supervening' on physical properties are physical.

[6]Clearly, this hypothesis can be wrong. It may turn out that no standard physical properties are directly encoded by neurons, but in the end this probably does not matter. We can still *describe* what they encode in terms of those properties, without being inaccurate (though our terminology

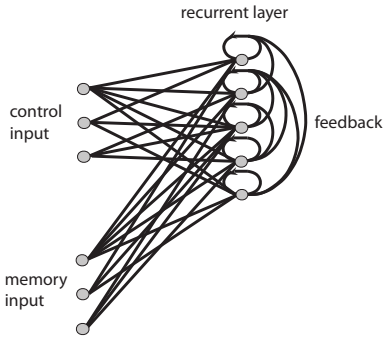

Figure 2.5: The architecture of a controlled integrator. This network can act like a loadable and erasable memory. Both the control input and the memory input are connected to the recurrent layer of neurons and the feedback to the recurrent layer is the product of both inputs. The NEF allows the neural connection weights in this network of spiking neurons to be determined, once we have defined the desired representations, computations, and dynamics of the system.

In other words, encodings of 'edible' depend, in some complex way, on encodings of 'lower-level' physical properties like wavelength, velocity, etc. The NEF itself does not determine precisely what is involved in such complex relations, although I do suggest that it provides the necessary tools for describing such relations. I return to these issues – related to the meaning (or 'semantics') of representations – throughout much of the book, starting in chapter 3.

For now, we can be content with the claim that whatever is represented, it can be described as some kind of structure with units. A precise way to describe structure is to use mathematics. Hence, this is equivalent to saying that the decoded alphabet consists in mathematical objects with units. The first principle of the NEF, then, provides a general characterization of the encoding and decoding relationship between mathematical objects with units and patterns of spikes in populations of neurons.

To make this characterization more concrete, let us turn to considering the example of the controlled integrator (see figure 2.5).

Perhaps the simplest such mathematical object is a scalar value. We can char-

---

may be slightly cumbersome; i.e. we may speak of 'derivatives-of-light-intensity', and so on). In short, it makes sense to begin to describing the properties encoded by animals in terms of familiar physical properties, because so much of our science is characterized in terms of those properties, we are familiar with how to manipulate those properties, and in the end we need a description that works (not necessarily the only, or even best possible, description).

acterize the horizontal position of an object in the environment as a scalar value, whose units are degrees from midline. There are neurons in a part of the monkey's brain called the lateral intraparietal (LIP) cortex, that are sensitive to this scalar value (Andersen et al., 1985). Indeed, these parts of the brain seem to act as a kind of memory for object location, as they are active even after the object has disappeared. I should be clear that I do not think a simple controlled integrator that remembers only a scalar value maps well to these specific neurons, but a similar architecture with a more complex representation has been used to model many properties of LIP activity (Eliasmith and Anderson, 2003). What is relevant for this example is that, as a population, neurons in this area *encode* an object's position over time.

To summarize, the representation of object position can be understood as a scalar variable, whose units are degrees from midline (decoded alphabet), that is encoded into a series of neural spikes across a population (encoded alphabet). Using the quantitative tools mentioned earlier, we can determine the relevant decoder (see appendix A.1.2). Once we have such a decoder, we can then estimate what the actual position of the object is given the neural spiking in this population, as in figure 2.4. Thus we can determine precisely how well, or what aspect of, the original property (in this case the actual position) is represented by the neural population. We can then use this characterization to understand the role that the representation plays in the system as a whole.

One crucial aspect of this principle of representation, is that it can be used to characterize arbitrarily complex representations. The example I have described here is the representation of a scalar variable. However, this same principle applies to representations of vectors (such as movement or motion vectors found in motor cortex, brainstem, cerebellum, and many other areas), representations of functions (such as stimulus intensity across a spatial contiuum as found in auditory systems, and many working memory systems), representations of vector fields (such as the representation of a vector of intensity, color, depth, etc. at each spatial location in the visual field as found in visual cortex), and representations of composable symbol-like objects (as argued for throughout this book). Suggesting that one of these kinds of representation can be found in a particular brain area, crucially, does not rule out the characterization of the same areas in terms of other kinds of representations. This is because we can quantitatively define a 'representational hierarchy' that relates such representations to one another. I return to this issue in section 2.4.

A second crucial aspect of this principle is that it distinguishes the mathematical object being represented from the neurons that are representing it. I refer to the

former as the 'state space', and the latter as the 'neuron space'. There are many reasons it is advantageous to distinguish the neuron space from the state space. Most such advantages should come clear in subsequent chapters, but perhaps most obviously, distinguishing these spaces naturally accounts for the well-known redundancy found in neural systems. Familiarity with Cartesian plots makes us think of axes in a space as being perpendicular. However, the common redundancy found in neural systems suggests that their 'natural' axes are in fact not perpendicular, but rather slanted towards one another (this is sometimes called an 'overcomplete' representation). Distinguishing the state space, where the axes typically are perpendicular, from the neuron space, where they are not, captures this feature of neurobiological representation.

The tutorial at the end of this chapter demonstrates how to build and interact with simulations of scalar and vector representations in Nengo. Both of these central features of the principle are further highlighted there with concrete examples.

- ???some comments on the importance of heterogeneity for improving the robustness of the system, and for being nearly optimal in representational capacity, etc. (out of book)??? (If not take out earlier reference to this section).

2.3.2 Transformation

A representational characterization is not be useful if it does not help us understand how the system functions. Conveniently, this characterization of neural representation paves the way for a general characterization of how these representations can be transformed. This is because, like representations, transformations (or computations) can be characterized using decoding. But, rather than using the 'representational decoder' discussed above, we can use a 'transformational decoder'. We can think of the transformational decoder as defining a kind of *biased* decoding. That is, in determining a transformation, we extract information *other than* what the population is taken to represent. The bias, then, is away from a 'pure', or representational, decoding of the encoded information.

For example, if we think that the quantity x is encoded in some neural population, when defining the representation we determine the representational decoders that estimate x. However, when defining a computation we identify transformational decoders that estimate some function, $f(x)$, of the represented quantity. In

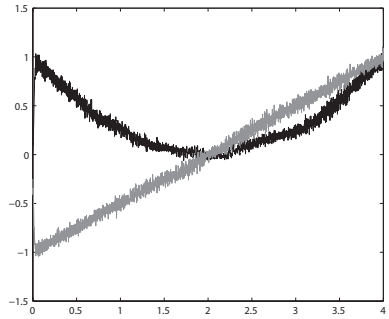

Figure 2.6: A nonlinear function of the input computed over time. The graph overlays the input signal in gray and the decoded output signal from a population of 50 neurons in black. As can be seen, a constantly ramping input from 0 to 1 is transformed to a quadratic function over the same range. This network is thus computing x^2.

other words, we find decoders that, rather than extracting the signal represented by a population, extract some transformed version of that signal. The same techniques used to find representational decoders are applicable in this case, and result in decoders that can support both linear and nonlinear transformations (see Appendix A.2). Figure 2.6 demonstrates how a simple nonlinear function can be computed in this manner.

Importantly, this understanding of neural computation applies at all levels of the representational hierarchy, and accounts for complex transformations. For example, it can be used to define inference relations, be they statistical (Eliasmith and Anderson, 2003, chp. 9), or more linguistic (see section 4.3). So, although linear decoding is simple, it can support the kinds of complex transformations needed to articulate descriptions of cognitive behavior. A main purpose of this book is to present one such description through the discussion and models presented in later chapters.

For present purposes, let us again consider the simple, specific example of a controlled integrator. The reason this integrator is 'controlled' is because, unlike a standard integrator, we can change the dynamics of the system by changing the input. Comparison of an ideal controlled, and non-controlled integrator are shown in figure 2.7. It is evident here that the dynamics of the system in the controlled integrator is a function of its input. Most directly, the 'control variable' is able to make the integrator act as a standard integrator, or exponentially forget its current

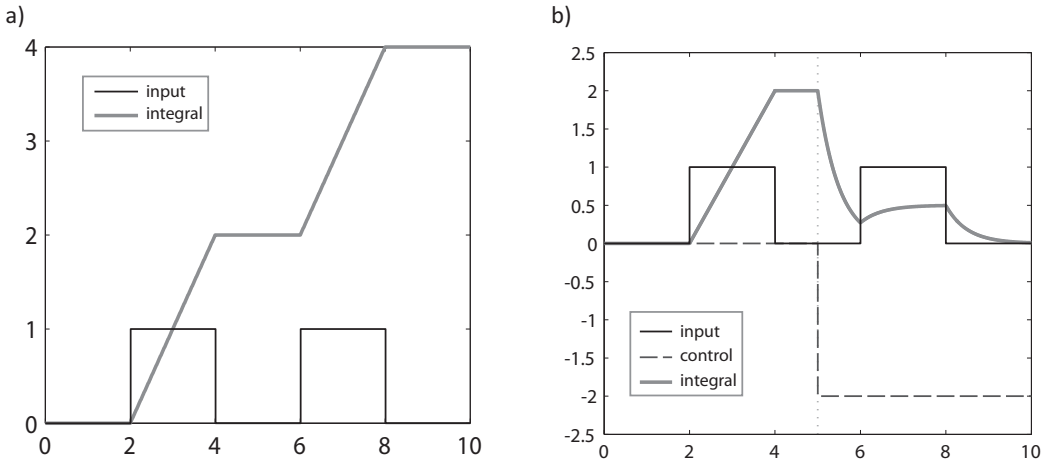

Figure 2.7: Comparison of a controlled and uncontrolled integrator. A) A standard integrator continuously adds its input to its current state. These dynamics do not change regardless of the input signal. B) A controlled integrator acts like a standard integrator if the control variable is equal to zero. Values less than zero cause the integrator to exponentially forget its current state. The speed of forgetting is proportional to the control variable.

state.

In order to build such a system, it is necessary to compute the product of the current state (i.e., the memory) and the control variable. Consequently, to implement such a system in a neural network, it is necessary to transform the represented state space of the recurrent layer (see figure 2.5) in such a way that it estimates this nonlinear function. To do so, it is important note that the state space of the recurrent layer represents both the memory and the control inputs. Hence, it contains a 2D vector representation $\mathbf{x} = [x_1, x_2]$. To compute the necessary transformation of this state space, we can find transformational decoders to estimate the function $f(\mathbf{x}) = x_1 x_2$, just as we earlier found the decoders to estimate the function $f(x) = x^2$ for a 1D scalar (see figure 2.6). The tutorial in section 3.7 demonstrates how to construct a network that performs scalar multiplication in this manner.

Before moving on to a consideration of dynamics, it is important to realize that this way of characterizing representation and computation does not demand that there are 'little decoders' inside the head. That is, this view does not entail that the system itself needs to decode the representations it employs. In fact,

according to this account, there are no directly observable counterparts to the representational or transformational decoders. Rather, as discussed in section 2.3.4, they are embedded in the synaptic weights between connected neurons. That is, coupling weights of connected neurons indirectly reflect a particular population decoder, but they are not identical to the population decoder. This is because connection weights are best characterized as determined by both the decoding of the incoming signal and the encoding of the outgoing signal. 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 both connectionism and theoretical neuroscience, this is exactly what happens in such networks. 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. Consequently, decoders – both transformational and representational – are theoretical constructs. The only place something like decoding actually happens is at the final outputs of the nervous system, such as at the motor periphery.

2.3.3 Dynamics

In the history of cognitive science, computation and representation have always been central players in our cognitive theories, but dynamics is a relative newcomer. And, while it may be understandable that dynamics were initially ignored by those studying cognitive systems as purely computational systems, it would be strange indeed to leave dynamics out of the study of minds as physical, neurobiological systems. Even the simplest nervous systems performing the simplest functions demand temporal characterizations: consider moving, eating, and sensing in a constantly changing world. It is not surprising, then, that single neural cells have almost always been modeled by neuroscientists as essentially dynamic systems. In contemporary neuroscience, researchers often analyze neural responses in terms of ‘onsets’, ‘latencies’, ‘stimulus intervals’, ‘steady states’, ‘decays’, etc. – these are all terms describing temporal aspects of a neurobiological response. The fact is, the systems under study in neurobiology are dynamic systems and as such they make it very difficult to ignore time.

Notably, modern control theory was developed precisely because understanding complex dynamics is essential for building something that works in the real

world. Modern control theory permits both the analysis and synthesis of elaborate dynamic systems. Because of its general formulation, modern control theory applies to chemical, mechanical, electrical, digital, or analog systems. As well, it can be used to characterize non-linear, time-varying, probabilistic, or noisy systems. As a result of this generality, modern control theory is applied to a huge variety of control problems, including autopilot design, spacecraft control, design of manufacturing facilities, robotics, chemical process control, electrical systems design, design of environmental regulators, and so on. It should not be surprising, then, that it also proves useful for characterzing the dynamics of complex neurobiological systems.

Central to employing modern control theory for understanding the dynamics of a system is the identification of the 'system state variable' ($\mathbf{x}(t)$ in figure 2.8). It is not a coincidence that the terminology introduced earlier has us calling the represented mathematical objects the 'state space'. This is because the third principle of the NEF is the suggestion that the representations of neural populations can be characterized as the state variables of a dynamical system using control theory.

However, things are not quite so simple. Because neurons have intrinsic dynamics dictated by their particular physical characteristics, we must adapt standard control theory to neurobiological systems (see figure 2.8). Fortunately, this can be done without loss of generality for linear and nonlinear dynamic systems (see Appendix A.3). Notably, all of the computations needed to implement such systems can be implemented using transformations as defined earlier in principle 2. As a result, we can directly apply the myriad techniques for analyzing complex dynamic systems that have been developed using modern control (and dynamic systems) theory to this quantitative characterization of neurobiological systems.

To get a sense of how representation and dynamics can be integrated, let us revisit the simple controlled integrator. As shown in figure 2.7, a standard integrator constantly sums its input. If we call the input $u(t)$ and the state of the controlled $x(t)$, we can write this relation as

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{2.1}$$

which is the scalar version of the system shown in figure 2.8a. This equation says that the change in $x(t)$ (written $\dot{x}(t)$) at the next moment in time is equal to its current value plus the input $u(t)$ (times some number B). For the rest of the discussion, we can let $B = 1$.

If we also let $A = 0$, then we get a standard integrator: the value of the state

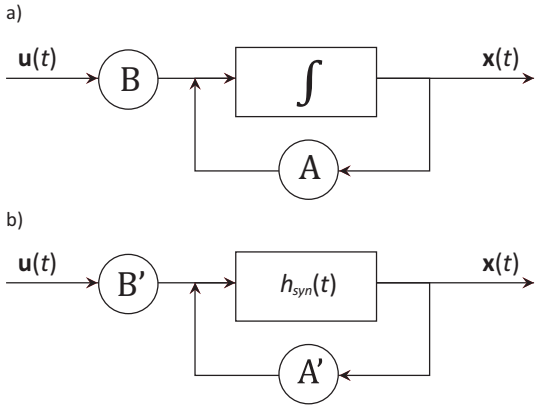

Figure 2.8: A control theoretic description of neurobiological systems. a) The canonical diagram for any linear system in control theory. **A** determines what aspects of the current state affect the future state. **B** maps the input to the system into its state space. The transfer function is perfect integration. b) The equivalent description for a neurobiological system. The matrices $\mathbf{A}'$ and $\mathbf{B}'$ take into account the effects of the transfer function, $h_{syn}(t)$, which captures the dynamics of neurons.

$x(t)$ at the next moment in time will be equal to itself plus the change $\dot{x}(t)$. That change is just equal to the input $u(t)$ (since $0x(t) = 0$), so the value of $x(t)$ will result from constantly be summing the input, just as in a standard integrator. In short,

$$x(t) = \int u(t)\,dt.$$

In many ways, this simple integrator is like a memory. After all, if there is no input (i.e., $u(t) = 0$) then the state of the memory will not change. This captures the essence of an ideal memory – its state stays constant over time with no input. The problem is that this simple integrator always just adds new input to the current state. If you can only remember one thing, and I ask you to remember the word 'cat', and then later ask you to remember the word 'dog', we would not expect you to report some kind of 'sum' of 'cat' and 'dog' as the contents of your memory when queried later. Instead, we would expect you to *replace* your memory of 'cat' with 'dog'. In short, there must be a way to empty the integrator before a new memory is remembered.

This is why we need to control the value of A in the above system. Consider

for a moment effect of changing the value of A in equation 2.1 to lie anywhere between -1 and 1. Let us suppose that the current value of $x(t)$ is not 0, and there is no input. If A is positive, then all future values of $x(t)$ will move increasingly far away from 0. This is because the change in $x(t)$ will always be positive for positive $x(t)$ and negative for negative $x(t)$. In fact, this movement away from zero will be exponential because the next change is always a constant fraction of the current state (which keeps getting bigger) – adding (or subtracting) a fraction of something to itself over and over results in an exponential curve. Now suppose that A is negative. In this case, the opposite happens. The current state will move exponentially towards 0. This is because the next state will be equal to itself minus some fraction of the current state.

To build a controlled integrator that acts as an erasable memory, we can thus implement the dynamical system in equation , with an additional input that controls A. We can use principle 3 to determine how to map this equation into one that accounts for neural dynamics. We then need to employ principle 2 to compute the product of A and $x(t)$ to implement this equation. And, we need to employ principle 1 to represent the state variable, the input variable, and the control variable in spiking neurons. The result of employing all three principles is shown in figure 2.9.

It is perhaps worth emphasizing that while the controlled integrator is simple – it computes only products and sums, it represents only scalars and a 2D vector, and it has nearly linear time-invariant dynamics – it employs each of the principles of the NEF. Furthermore, there is nothing about these 'simplicities' that are a consequence of the principles employed. NEF models can compute complex nonlinear functions, have sophisticated high-dimensional representations, and display interesting nonlinear dynamics. In fact it is this generality, I believe, that puts the NEF in a unique position to help develop and test novel hypotheses about biological cognition. Developing and testing one such hypothesis is the purpose of the remainder of this book, beginning with the next chapter.

2.3.4 The three principles

Hopefully it is clear that this presentation of the NEF is somewhat superficial. It is not intended to satisfy those deeply familiar with theoretical neuroscience. I have side-stepped issues related to optimal decoding, nonlinear versus linear decoding, information transfer characteristics, nonlinear dendritic function, rate

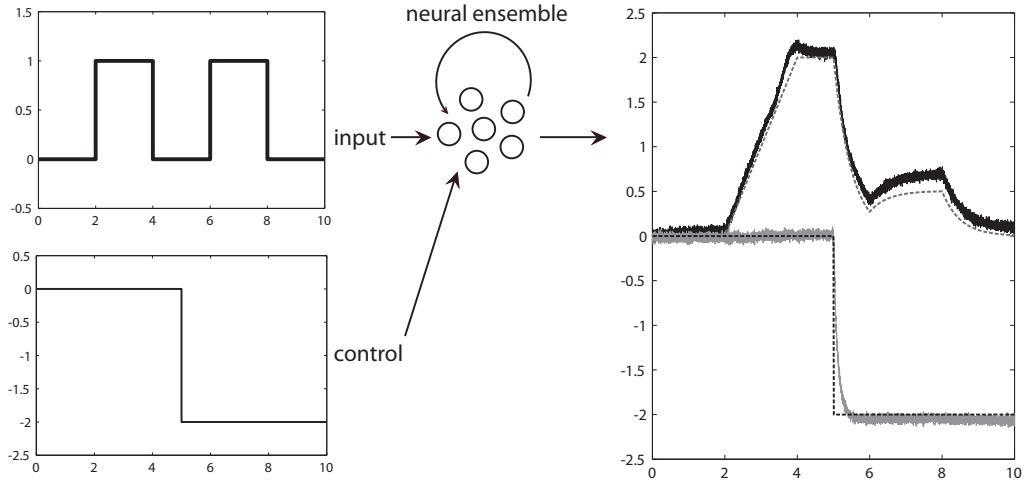

Figure 2.9: A recurrent spiking network implementing a controlled integrator. Input and control signals are connected to a neural ensemble which has connections to its own feedback terminal. The neural ensemble contains representations of both the control signal and the controlled integral of the input signal. These representations are shown in the rightmost graph: the solid black line is the decoded representation of the integral, the solid gray line is the representation of the control signal, and dotted lines depict the ideal values of the integral and the control. The tutorial at the end of chapter 5 demonstrates how to build this controlled integrator in Nengo.

versus timing code considerations, and so on. While I believe the NEF satisfactorily addresses issues, my purpose here is to give the necessary background to make the methods plausible, usable, and not unnecessarily complex.

In that spirit, it is perhaps useful to summarize the principles diagramatically, to both clarify how they relate to neurobiology and to demonstrate what they contribute to an analysis of neural systems. Figure 2.10 shows what I have called a 'generic neural subsystem'. The purpose of identifying such a subsystem is to note that such 'blocks' can be strung together indefinitely to describe neural function. This subsystem is generic because it has input spikes, which generate post-synaptic currents (PSCs) that are then weighted and passed through a neural non-linearity which generates outputs spikes. This characterization is generic across almost all areas of the mammalian brain.[7]

In addition, the subsystem in figure 2.10 captures the contributions of the principles. Together, the principles determine the synaptic weights. That determination depends on the contributions of each of the principles independently: the representation principle identifies encoders and representational decoders; the transformation principle identifies transformational decoders; and the dynamics principle identifies the dynamics matrices (i.e. the matrices identified in figure 2.8). The synaptic weights themselves are the product of these elements.

Crucially, while this theoretical characterization of the subsystem is generic, its application is not. To determine decoders, the tuning curves of neurons play a role. To determine dynamics, the kinds of neurotransmitters found in a given circuit are crucial. To determine what kind of spiking occurs, the choice of a single cell model is crucial. To determine encoders, both the single cell model and the tuning curves need to be known. To determine transformational decoders and dynamics, a high-level hypothesis about the system function is needed. All of these considerations can vary widely depending on which brain areas are being considered.

I mentioned several examples of the broad application of the NEF at the beginning of section 2.3. As noted, these applications have focussed on detailed neural models. A benefit of this focus has been that the NEF has successfully accounted for changes in single cell tuning curves under different circumstances (Conklin and Eliasmith, 2005), has been able to better account for the variety of single cell

[7]Some notable exceptions are retinal processing, some dendro-dendtric interactions, and gap junctions. An extension to the generic neural subsystem as drawn here can capture such interactions by incorporating optional elements that include these other potential sources of dendritic current. This added complexity obscures the simplicity and broad generality of the subsystem, however.

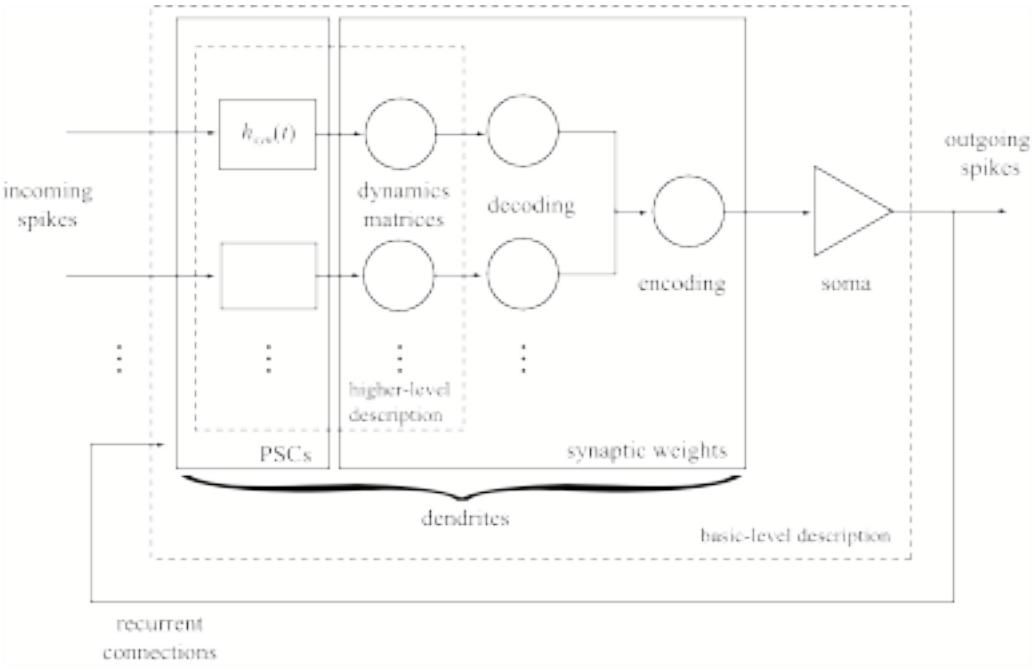

Figure 2.10: A generic neural subsystem. A synthesis of the preceding characterizations of representation (encoding/decoding), computation (biased decoding), and dynamics (captured by $h_{syn}(t)$ and the dynamics matrices). Dotted lines distinguish neural and state space (i.e., mathematical objects with units) descriptions.

dynamics during working memory tasks than other models (Singh and Eliasmith, 2006), and has provided a model of eye position control that proposes a novel synaptic learning rule for adapting individual synapses that is able to account for recovery from ablation of single cells, the effects of systematic inputperturbation, and normal functioning in a more biologically plausible manner than past models (ref macneill???). In short, the credentials of the NEF as a method for generating models with close connections to detailed neural data are good.

As well, these applications make it clear that the NEF bears little resemblance to traditional connectionism: NEF neurons spike, they are highly heterogeneous, they have a variety of different dynamics, they are differentially affected by different neurotransmitters – they are not 'units' or 'nodes', but *neurons*. In addition, the NEF does not rely on learning to design models (although learning is often included if appropriate to the system of interest), consequently it can be used to construct arbitrarily 'deep' (i.e. with any number of layers), and arbitrarily connected models. This is because the networks can be designed both top-down (given a functional hypothesis, how might neurons be organized), and bottom-up (given a network structure and a learning rule, what functions can be realized).

However, the drawback of these focussed applications is that the kinds of functions addressed are simple, making less than obvious how the NEF might be relevant for understanding complex behavior. It has been suggested that a detailed understanding of neural implementation is completely irrelevant for understanding cognition (Fodor and Pylyshyn, 1988). I won't argue that point here.[8] Nevertheless, it is not obvious what the relation *is* between these simple neural systems and cognition: presumably we want to understand the cognitive forest as well as the neural trees.

Before confronting the task of applying the NEF to cognitive systems in the next chapter, I believe it is conceptually important to clarify a notion I will be using throughout the remainder of the book. A notion I have been using somewhat loosely to this point: the notion of 'levels'.

2.4 Levels

At one point in the development of science, it was widely thought that the sciences could be identified with the level of nature which they characterized. Physics was the most fundamental, followed closely by chemistry, and subsequently biology, psychology, sociology and so on (ref??? read bechtel?). The suggestion was that lower-level sciences could reductively describe the higher-level sciences. Soci-

[8] Though I have elsewhere (???cite BBS paper).

eties, after all, are composed of people, who are made up of biological parts, which are largely driven by chemical processes, which can be understood as interactions between the fundamental parts of nature. This idyllic view has not withstood the test of time. However, the notion that there are 'levels' of some kind has proven too useful to discard with this view. Instead it has become a perennial problem to determine what, exactly, the relation between the sciences is.

Jerry Fodor has famously suggested that, since the reduction of one science to another has failed, the sciences must be independent (ref fodor). Consequently, he has been argued that to understand cognitive systems, which lie in the domain of psychology, appeal to lower-level sciences is useless. The only thing such disciplines, including neuroscience, can do is to provide an implementational story that bears out whatever psychological theory has been independently developed. There are many difficulties with such a view (refs???), but the one that stands out in the context of developing cognitive theories is that cleaving neuroscience from psychology arbitrarily throws out an enormous amount of empirical data about the system we are attempting to develop our theory about. If we were already certain that we had a perfect psychological-level description of the system, this strategy might be defensible. However, we are in no such position.

There are many possible relations that may exist between levels other than the extremes of their being reducible to one another, or their being independent. In fact, there are many different things we might mean by 'levels' in the first place. And, different notions of 'levels' may require different characterization of the relations between them. I suspect that the reason both the reducibility and the independence views seem implausible is because they share a very strong interpretation of the term 'level' in nature. Specifically, both assume that such levels are *ontological* (Oppenheim and Putnam, 1958). That is, that the levels themselves are intrinsic divisions in the structure of the natural world. The reason we might assume we can reduce people to particles, is because people are ultimately best described as a bunch of particles stuck together in some (perhaps very complex) way. The reason we might assume theories about people are independent of theories about particles, is because reduction does not work, but both are equally real and so both kinds of theories are equally scientific.

If, instead, we think of levels as different *descriptions* of a single underlying system, neither of these views seem plausible. Sometimes we may describe a system as a person without much consideration of their relationship to individual particles. Other times we may be concerned about the decomposition a biological person, or that person's parts, which may ultimately lead in the direction of a reductive description. The reasons for preferring one such description

over another can be largely practical: in one case we need to predict the overall behavior of the whole system; in another we need to explain how changes in a component influence complex interactions. In either case, we need not take these different perspectives on the underlying system as something we need reify (i.e., make real). Rather, we can understand levels to be descriptions that are chosen for purposes.

Crucially, this does not mean that levels are in any problematic sense 'made up': they can still be right and wrong (depending on whether they agree with empirical evidence), and they can still be about real things (i.e., things whose existence does not depend on our existence). Instead, they are flexible in a manner that acknowledges our limited intellectual capacities, or maybe simply our limited knowledge. Like any tool, theories must be useable. Consequently, it is sometimes important to make assumptions or simplifications in order to make predictions. In psychology, we may talk about people as if they were unchanging sets of particles because most of our predictions are sufficiently accurate regardless of that assumption.

I have painted a picture of levels as pragmatically driven descriptions of underlying phenomena. Perhaps it is useful to call this position 'descriptive pragmatism' for short. This view will no doubt be unappealing to those wanting ontological levels for science, but descriptive pragmatism does a better job of capturing the many competing ideas at work in talk of 'levels' in scientific practice.[9] Specifically, it explains why levels are related to complexity: more complex systems require more resources to characterize.[10] Consequently, a simpler high-level descriptions of such a system requires more assumptions than a more complete, but lower-level description. Nevertheless, such assumptions are warranted if they do not prevent us from adequately explaining or predicting the system of interest.

Descriptive pragmatism also help us understand why levels, spatial scale, and complexity are inter-related: it is often the case that if we have two objects of a given size, they take up about twice the space of one such object, and they interact only with each other. As we increase the number of objects, we need to increase the amount of space necessary to encapsulate the system, and the number of possible interactions goes up exponentially (???or faster) as well. So, typically, larger spatial scales allow for more complexity and concurrently demand higher

[9]See Bechtel (???ref) for a discussion of several failed past attempts to adequately describe an ontological notion of levels.

[10]By many definitions, this is precisely what complexity is (Kolmogorov, 1968; Cun and Denker, 1992)???also cite J. M. Carlson, J. Doyle, Proc. Natl. Acad. Sci. U.S.A. 99, 2538 (2002).???

level descriptions.

Finally, this view of levels is consistent with intuitions that part/whole relations, especially in mechanisms, often coincide with levels. Most systems composed of parts have parts that interact. If the parts of a system are organized in such a way that their interaction results in some regular phenomena, they are called 'mechanisms' Machamer et al. (2000). Mechanisms are often analyzed by decomposing them into their simpler parts, which may also be mechanisms. So, within a given mechanism, we again see a natural correspondence between spatial scale, complexity and levels. Note, however, that even if two distinct mechanisms can be decomposed into lower level mechanisms, this does not suggest that the levels *within* each mechanism can be mapped *across* mechanisms. This, again, suggests that descriptive pragmatism better captures scientific practice than more traditional ontological views ???ref bechtel's new book???.

Descriptive pragmatism about levels may initially seem to be a more arbitrary characterization of levels because the levels are not set by nature. However, we can still demand systemmatic identification and quantification of the relation between levels. The descriptions we employ can still be mathematical, after all. In fact, mathematical descriptions, being abstract, can help clarify the assumptions made when identifying levels. We should suspect that precisely where the description is abstracted from empirical evidence is where practical considerations are at work. We might talk in terms of mathematical objects that represent concepts instead of a mathematical description of individual neuron function because successful explanation or prediction of certain behavior can proceed in terms of concepts without detailed consideration of individual neuron function. The nature of 'without detailed consideration' is specified by assumptions that underly our mathematical theory concept representation.

With this background in mind let me return to specific consideration of the NEF. As described in section 2.3.1 on the principle of representation, the principle applies to the representation of all mathematical objects. Since such objects can be ordered by their complexity, we have a natural and well-defined meaning of a representational hierarchy. Table 2.1 provides the first levels of such a hierarchy.

This hierarchy maps well to notions of levels as spatial scale, complexity, or part/whole relations. If we hold the precision of the representation constant, then the higher levels in the hierarchy will require more neurons, and hence typically be ascribed larger areas of cortex (relative few cells are needed to encode light intensity at a single spatial position, compared to encoding light intensity over the visual field). Relatedly, these higher-level representations will be able to underwrite more complex behaviors (object recognition will often require detailed in-

Table 2.1: A representational hierarchy. Each row has the same type of encoding/decoding relations with neurons. Higher rows can be constructed out of linear combinations of the previous row.

Mathematical Object	Dimension	Example
Scalar (x)	1	Light intensity at x, y
Vector ($\mathbf{x}$)	N	Light intensity, depth, color, etc. at x, y
Function ($x(v)$)	∞	Light intensity at all spatial positions
Vector Field ($x(\mathbf{r}, v)$)	$N \times \infty$	Light intensity, depth, color, etc. at all spatial positions
$\vdots$	$\vdots$	$\vdots$

tensity information across large portions of the visual field, whereas orienting does not). Finally, the hierarchy clearly defines how the higher-levels are built up out of the lower levels. Hence mechanisms trafficking in one level of representation will often be decomposable into mechanisms trafficking in lower-level representations.

Given the tight relationship between principles 1 and 2 (the latter is a simple variation of the former), it should not be surprising that what goes for representations also goes for transformations. High-level computations will be carried out over larger spatial scales, can be more complex, and relate to high-level mechanisms. Furthermore, because principles 1 and 2 are used to implement principle 3 (dynamics are defined by transformations of representations), the same can be said for dynamics.

Notice also that all of the levels of this hierarchy are described using the the same encoding/decoding relationship with individual neurons. For this reason, I will often talk of neuron-level descriptions as being lower-level than these representational descriptions. In other words, the neuron space/state space distinction is also a distinction between levels of description. The representational hiearchy itself lies within the state space half of this distinction. The relation between the neuron and state space levels is different than that between levels within the representational hierarchy. But, both are quantitatively defined, and the increase of levels with spatial scale, complexity, and part/whole relations still obtains. Consequently, it still makes sense to talk of all of these as levels of the same system.

The fact that all of these levels can be described quantitatively and in a standard form suggests that this characterization provides a unified way of understanding neurobiological systems. In addition, it makes clear how we can 'move between' levels, and precisely how these levels are *not* independent. They are not independent because empirical data that constrains one description is about *the*

same system described at a different leve. So, when we move between levels, we must either explicitly assume away the relevance of lower (or higher) level data, or we rely on the hypothesized relationship between the levels to relate that data to the new level. Either way, the data influences our characterization of the new level (since allowable assumptions are part and parcel of identifying a level).

Before leaving this consideration of levels, one final subtlety of levels in my subsequent description of neural systems is worth discussing. That is, different descriptions of neural function can 'move' the complexity of the systems into different parts of the description. That is, in general, what we may naturally call high-level neural representations (e.g., the representation of phonemes), may be relatively simple mathematical objects (i.e., a scalar carrying only one of forty possible values). On the face of it, this characterization of phonemes as simple mathematical objects seems to contradict my earlier claim that, in general, higher-level representations are more complex mathematical objects. However, that concern misses the fact that the complexity of this representation has been moved to the units of that object. That is, phonemes themselves are very complicated objects that require many sophisticated computations to extract (they depend not only on complex interactions of frequencies, but auditory and motor contexts). So the the fact that the unit of the object is 'phoneme' captures enormous amounts of complexity. Consequently, the representational hierarchy applies in cases when the units of the component representations stay constant (as in the examples provided in table 2.1). If we change units, we may still proceed 'up' levels even though the mathematical objects get simpler. Of course, the relationship between units at different levels must also be specified to properly characterized the relationship between these levels. This relationship is most often a transformational one, and hence not one that the NEF specifies in general (recall from section 2.3 that the NEF is about *how*, not *what,* neural systems compute).

So, the NEF provides a consistent, precise way to talk about levels in the behavioral sciences. Some, but not all of the inter-level relations are defined by the principles of the framework. This seems appropriate given the current state of ignorance about how best to decompose the highest-level neural systems. Nevertheless, the NEF still provides systemmatic and quantitive tools for carefully exploring such decompositions.

2.5 Nengo: Neural representation

In this tutorial, we examine two simple examples of neural representation. These examples highlight two aspects of how principle 1 in section 2.3.1 characterizes neural representation. First, the examples make evident how the activity of neural populations can be thought of as representing a mathematical *variable*. Second, we examine a simple case of moving up the representational hierarchy to see how the principle generalizes from simple to more complex cases.

Representing a scalar

We begin with a network that represents the simplest mathematical object, a scalar value. As in the last tutorial, we will use the interactive mode to examin the behavior of the running model. Let us begin.

- In an empty Nengo world, right-click anywhere on the background and choose *New Network.* Set the *Name* of the network to 'Scalar Representation' and click *OK*.

- Right-click inside the network, select *Create new->NEFEnsemble*

Here the basic features of the ensemble can be configured.

- Set *Name* to 'x', *Number of nodes* to 100, *Dimensions* to 1, *Node factory* to 'LIF Neuron', and *Radius* to 1.

The name is a way of referening to the population of neurons you are creating. The number of nodes is the number of neurons you would like to have in the popuation. The dimension is the number of different elements in the vector you would like the population to represent (a 1 dimensional vector is a scalar). The node factory is the kind of single cell model you would like to use. The default is a simple leaky integrate-and-fire (LIF) neuron. Finally, the radius determines size of the *n*-dimensional hypersphere that the neurons will be good at representing. A 1-dimensional hypersphere with a radius of 1 is the range from -1 to 1 on the real number line. A 2-dimensional hypersphere with the same radius is a unit circle.

- Click the *Set* button. In the panel that appears, you can leave the defaults (*tauRC* is 0.02, *tauRef* is 0.001, *Max rate* low is 200 high is 400, *Intercept* is low -1.0 and high is 1.0).

Clicking on *Set* allows the parameters for generating neurons in the population to be configured. Briefly, *tauRC* is the RC time constant for the neuron membrane, usually 20ms, *tauRef* is the absolute refractory period (the period during which a neuron cannot spike after having spiked), *Max rate* has a high and low value, in Hertz, which determines the range of firing rates neurons will have at the extent of the radius (the maximum firing rate for a specific neuron is chosen randomly from a uniform distribution between low and high), *Intercept* is the range of possible values along the represented axis where a neuron 'turns off' (for a given neuron its intercept is chosen randomly from a uniform distribution between low and high).

- Click *OK*. Click *OK* again, and the neurons will be created.

If you double-click on the population of neurons, each of the individual cells will be displayed.

- Right-click on the population of neurons, select *Plot->Constant Rate Responses*.

The 'activities' graph which is now displayed shows the 'tuning curve' of all the neurons in the population. This shows that there are both 'on' and 'off' neurons in the population, that they have different maximum firing rates at $x = \pm 1$, and that there is a range of interecepts between $[-1, 1]$. These are the heterogeneous properties of the neurons that will be used to represent a scalar value.

- Right-click on the population of neurons, select *Plot->Plot distortion: X*.

This plot is an overlay of two different plots. The first, in red and blue compares the ideal representation over the range of x (red) to the representation by the population of neurons (blue). If you look closely, you can see that blue does not lie exactly on top of red, though it is close. To emphasize the difference between the two plots, the green plot is the distortion (i.e. the difference between the ideal and the neural representation). Essentially the green plot is the error in the neural representation, blown up to see its finer structure. At the top of this graph, in the title bar, the error is summarized by the MSE (mean squared error) over the range of x. Importantly, MSE decreases as the square of the number of neurons (so RMSE is proportional to 1/N), so more neurons will represent x better.

- Right-click on the population and select *Configure*. Any of these properties can be changed for the population.

There are too many properties here to discuss them all. But, for example, if you click on the arrow beside *i neurons*, double-click the current value. You can now set it to a different number and the population will be regenerated with the new number of neurons.

Let us now examine the population running in real time.

- Right-click inside the Scalar Representation network and select *Create new->Function input*.
- Set *Name* to 'input', make sure *Output Dimensions* is 1 and click *Set Functions*.
- Select *Constant Function* from the drop down menu click *Set* and set *Value* to 0. Click *OK* three times.

The function input will appear. We will now connect the function input to the neural population as in the tutorial in section 1.5. This essentially means that the output from the function will be injected into the soma of each of the neurons in the population, driving its activity.

- Right-click the 'x' population and select *Add decoded termination*. Set *Name* to 'input', *Weights Input Dim* to 1, and *tauPSC* to 0.02. Click *Set Weights*, double-click the value and set it to 1. Click *OK* twice.
- Click and drag the 'origin' on the input function you created to the 'input' on the 'x' population.
- Right-click in the Scalar Representation network and select *Interactive Plots*.

This plot should be familiar from the previous tutorial. It allows us to interactively change the input, and watch various output signals generated by the neurons.

- Right-click 'x' and select *value*. Right-click 'x' and select *spike raster*. Right-click 'input' and select *value*. Right-click 'input' and select *control*.

Change the layout to something you prefer by dragging items around. If you would like the layout to be remembered in case you close and re-open these plots, click the small triangle in the middle of the bottom of the window (this expands the view), then click the disk icon under *layout.*

- Click the play button. Grab the control and move it up and down. You will see changes in the neural firing pattern, and the value graphs of the input and the population.

- Note that the value graph of the 'x' population is the linearly decoded estimate of the input, as per the first principle of the NEF.
- Note also that the spike raster graph is displaying the encoding of the input signal into spikes.

The spiking of only 10% of the neurons are shown by default. To increase this proportion:

- Right-click on the spike raster graph and select a larger percentage of neurons to show.

The population of neurons does a reasonably good (if slightly noisy) job of representing the input. However, neurons can not represent arbitrary values well. To demonstrat this do the following.

- Right-click the control and select *increase range*. Do this again. The range should now be ± 4.
- Centre the controller on zero. The population should represent zero. Slowly move move the controller up, and watch the value graph from the 'x' population.

Between 0 and 1, the graph will track the controller motions well. Notice that many of the neurons are firing very quickly at this point. As you move the controller past 1, the neural representation will no longer linearly follow your movement. All the neurons will become 'saturated,' that is, firing at their maximum possible value. As you move the controller past 2 and 3, the decoded value will almost stop moving altogether.

- Move the controller back to zero. Notice that changes around zero cause relatively large changes in the neural activity compared to changes outside of the radius (which is 1).

These effects make it clear why the neurons do a much better job of representing information within the defined radius: changes in the neural activity outside the radius no longer accurately reflect the changes of the input. This becomes especially true under noisy conditions, where small changes are easily masked by noise.

Notice that this population, representing a scalar value, does not in any way store that value. Instead, the activity of the cells act as a momentary representation

of the current value of the incoming signal. That is, the population acts together like a variable, which can take on many values over a certain range. The particular value it takes on is represented by its activity, which is constantly changing over time. This conception of neural representation is very different from that found in many traditional connectionist networks which assume that the activation of a neuron or a population of neurons represents the activation of a specific 'concept'. Here, the same population of neurons, *differently activated*, can represent different 'concepts'.[11] I return to this issue in sections 9.3 and 10.3.1.

Representing a vector

A single scalar value is the simplest neural representation, and hence at the bottom of the representational hierarchy. Combining two or more scalars into a representation, and moving up one level in the hierarchy, results in a vector representation. In this tutorial, I consider the case of two-dimensional vector representation, but the ideas naturally generalize to any dimension. Many parts of cortex are best characterized as using vector representations. Most famously, Apostolos Georgopoulos and his colleagues have demonstrated vector representation in motor cortex (Georgopoulos et al., 1984, 1986, 1993).

In these experiments, a monkey moves its arm in a given direction while the activity of a neuron is recorded in motor cortex. The response of a single neuron to forty different movements is shown in figure 2.11. As can be seen from this figure, the neuron is most active for movements in a particular direction. This direction is called the 'preferred direction vector' for the neuron. Georgopoulos' work has shown that over the population of motor neurons, these preferred direction vectors are evenly distributed around the unit circle in the plane of movement(Georgopoulos et al., 1993).

So to construct a model of this kind of representation, we can do exactly the same steps as for the scalar representation, but with a two-dimensional representation.

- In an empty Nengo world, right-click anywhere on the background and choose *New Network*. Set the *Name* of the network to 'Vector Representation' and click *OK*.

[11] This is true even if we carve the populations of neurons up differently. That is, there is clearly a range of values (perhaps not the whole range in this example) over which exactly the same neurons are active, but different values are represented.

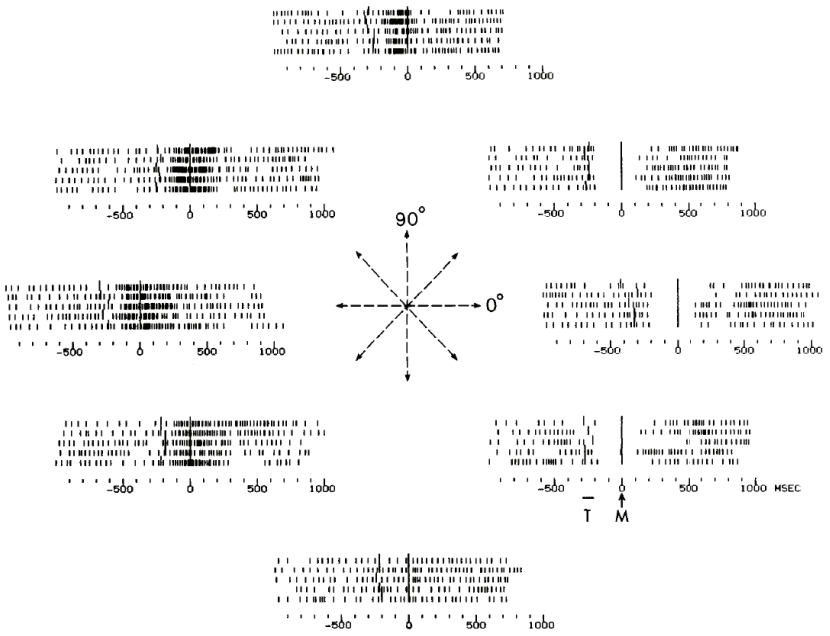

Figure 2.11: The response of a single neuron in motor cortex to different directions of movement. M indicates the time of movement onset. T indicates the time that the target appears. This data shows five trials in each direction for this neuron. The neuron is most active when the target is between 135 and 180 degrees. The direction of the peak of this response is called the neuron's 'preferred direction vector'. (Reproduced from Georgopoulos et al. (1986), no permission yet???)

- Right-click inside the network, select *Create new->NEFEnsemble*
- Set *Name* to 'x', *Number of nodes* to 100, *Dimensions* to 2, *Node factory* to 'LIF Neuron', and *Radius* to 1.
- Click the *Set* button. In the panel that appears, you can leave the defaults (*tauRC* is 0.02, *tauRef* is 0.001, *Max rate* low is 200 high is 400, *Intercept* is low -1.0 and high is 1.0).

The only interpretational difference for vector representations of these parameters is that the *Max rate* and *Intercept* are defined along the preferred direction vector (which is always one-dimensional). You can think of the scalar representation as having only two possible 'preferred directions', positive and negative. The preferred direction vector generalizes that notion to higher-dimensional spaces.

- Click *OK*. Click *Advanced*. This expands the window. Ensure that the *Encoding Distribution* slider is all the way to the left (over 0.0 - *Evenly Distributed*).

The previous step gives a motor cortex-like distribution of preferred direction vectors.

- Click *OK*, and the neurons will be created.

Given the default settings, preferred direction vectors will be randomly chosen from an even distribution around the unit n-dimensional hypersphere (i.e. the unit circle in two dimensions). If you plot the constant rate responses, the neuron responses will be plotted along the preferred direction vector of the cell. Consequently, the plot is generated as if all neurons had the same preferred direction vector.

You can now create an input function, and run the network.

- Right-click inside the Vector Representation network and select *Create new->Function input*.
- Set *Name* to 'input', make sure *Output Dimensions* is 2 and click *Set Functions*.
- Select *Constant Function* from the drop down menu, click *Set* and set *Value* to 0 for both functions. Click *OK* three times.

- Right-click the 'x' population and select *Add decoded termination*. Set *Name* to 'input', *Weights Input Dim* to 2, and *tauPSC* to 0.02. Click *Set Weights*, double-click the top-right value and set it to 1, do the same to the bottom-left value. Click *OK* twice.

Notice that you have a matrix of weights to set. This is because the input functions can be projected to either of the dimensions represented by the neural population. For simplicity, we have told the first function to go only to the first dimension and the second function only to the second dimension.

- Click and drag the 'origin' on the input function you created to the 'input' on the 'x' population.
- Right-click in the Vector Representation network and select *Interactive Plots*.

We can begin by looking at the analogous displays to the scalar representation.

- Right-click 'x' and select *value*. Right-click 'x' and select *spike raster*. Right-click 'input' and select *value*. Right-click 'input' and select *control*.
- Press play to start the simulation. Move the controls to see the effects on the spike raster.

You can attempt to find a neuron's preferred direction vector, but it will be difficult because you have to visualize where in the 2D space you are because the *value* plot is over time.

- Right-click the 'input' *value* plot and select *hide*.
- Right-click 'input' and select *XY plot*. Attempt to determine a neuron's preferred direction vector.

This should be easier because you can now see the position of the input vector in 2D space. There is a trail to the plot to indicate where it has recently been. The easiest way to estimate a neuron's preferred direction vector is to essentially replicate the Georgopolous experiments.

- Move the input so both dimensions are zero. The move one input to it's maximum and minimum values. If a neuron does not respond (or responds minimally), that is not it's preferred direction.
- Repeat the process of moving the input to a point in the unit circle, and then to the opposite point (flipping both signs) on the circle.

The preferred direction will be the direction which has the greatest difference between these two inputs. Keep in mind that different neurons have different gains, meaning they may 'ramp' up and down at different rates even with the same preferred direction.

- Right-click the 'x' population and select *preferred directions* to show the neuron activity plotted along their preferred directions in the 2D space.
- Put one input at an extreme value, and slowly move the other input between extremes.

It should be clear that something like the average activity of the population of neurons moves with the input. If you take the mean of the input (a straight line through the middle of the blob of activity), it will give you an estimate of the input value. That estimate is like the linear decoding defined in the first principle, but for a vector space.

- Right-click the 'x' population and select XY plot. This shows the actual decoded value in 2D space.

Another view of the activity of the neurons can be given by looking the neurons plotted in a pseudo-cortical sheet.

- Right-click the 'x' population and select *voltage grid*.

This graph shows the subthreshold voltage to the neurons in the population in grey. Yellow boxes indicate that a spike is fired.

- Right-click on the voltage grid and select *improve layout* to organize the neurons so that ones with similar preferred direction vectors will be near each other, as in motor cortex.
- Move the sliders and observe the change in firing pattern.

Using the same kind of exploration of inputs as before, it is reasonably evident in this view which parts of the grid have which preferred direction vectors. This summarizes approximately how motor cortex is thought to encode a movement to a target by Georgopolous et al.

Recent work has challenged the idea that there is such a clean mapping between neural activity and target location in general (ref shenoy lab???). Nevertheless, this provides a useful introduction to vector representation in cortex.[12] It

[12]Indeed, this same characterization of vector representation can be used to better understand these exact challenges (ref???).

is important to keep in mind that some aspects of this tutorial on vector representation are specific to this motor cortical example. For instance, the assumption of an even distribution of preferred direction vectors, and that the vectors are two-dimensional. However, many aspects are more general, including the identification of preferred direction vectors and the use of a linear decoding to get an estimate of the population representation.

It is worth highlighting briefly that the represented vectors in this characterization of neural function are not the same as the activity vectors commonly discussed in artificial neural networks. Activity vectors are typically a set of neuron firing rates. If we have three neurons in our population, and they are active at 50, 100, and 20 Hz respectively, the 3D activity vector would be [50, 100, 20], and it defines a point in a space where each axis is associated with a specific neuron. In the example above, the activity vector would be in a 100D space because there are 100 neurons in the population. In contrast, the 2D space represented above has axes determined by an externally measured variable.[13]

Recall from section 2.3.1 that we can call the 2D space in this example the 'state space', where as the 100D space is the 'neuron space'. Consequently, the 2D state space, we can think of as a standard Cartesian space, where two values uniquely specify a single object as compactly as possible. In contrast, the 100D space, while specifying the same object, takes many more resources (i.e. values) to do so. Of course, if there is no uncertainty in any of these values this seems like a simple waste of resources. However, in the much more realistic situation where there is uncertainty (due to noise of receptors, or noise in the channels sending the signals, etc.) this redundancy can make specifying that object much more reliable. And, interestingly, it can make the system much more flexible in how well it represents different parts of that space. For example, we could use 10 of those neurons to represent the first dimension, or we could use 50 neurons to do so. The second option would give a much more accurate representation of that dimension than the first. Being able to redistribute these resources to respond to task demands is one of the foundations of learning (see section ???).

More extensive tutorials on neural representation are available on the CRNG website at `http://compneuro.uwaterloo.ca/cnrglab/?q=node/2`.

[13] These axes do not need to be externally measurable, as well shall see. However, they are in many of the simplest cases of neural representation.