NC ms 3009

A unified approach to building and controlling spiking attractor networks

Chris Eliasmith

23rd September 2004

Abstract

Extending work in Eliasmith and Anderson (2003), I employ a general framework to construct biologically plausible simulations of the three classes of attractor networks relevant for biological systems: static (point, line, ring, and plane) attractors; cyclic attractors; and chaotic attractors. I discuss these attractors in the context of the neural systems that they have been posited to help explain: eye control, working memory, and head direction; locomotion (specifically swimming); and olfaction, respectively. I then demonstrate how to introduce control into these models. The addition of control shows how attractor networks can be used as subsystems in larger neural systems, demonstrates how a much larger class of networks can be related to attractor networks, and makes it clear how attractor networks can be exploited for various information processing tasks in neurobiological systems.

1 Introduction

Persistent activity has been thought to be important for neural computation at least since Hebb (1949), who suggested that it may underly short-term memory. Amit (1989), following work on attractors in artificial neural networks (e.g., that of Hopfield 1982), suggested that persistent neural activity in biological networks is a result of dynamical attractors in the state space of recurrent networks. Since then, attractor networks have become a mainstay of computational neuroscience, and have been used in a wide variety of models (see, e.g., Zhang 1996; Seung et al. 2000; Touretzky and Redish 1996; Laing and Chow 2001; Hansel and Sompolinsky 1998; Eliasmith et al. 2002).

Despite a general agreement amongst theoretical neuroscientists that attractor networks form a large and biologically relevant class of networks, there is no general method for constructing and controlling the behavior of such networks. In this paper, I present such a method and explore several examples of its application, significantly extending work described in Eliasmith and Anderson (2003). I argue that this framework can both unify the current use of attractor networks and show how to extend the range of applicability of attractor models. Most importantly, perhaps, I describe in detail how complex control can be integrated with standard attractor models. This allows us to begin to answer the kinds of pressing questions now being posed by neuroscientists, including, for example, how to account for the dynamics of working memory (see, e.g., Brody et al. 2003; Fuster 2001; Rainer and Miller 2002).

In the remainder of this paper, I briefly summarize the general framework and then demonstrate its application to construction of spiking networks that exhibit line, plane, ring, cyclic, and chaotic attractors. Subsequently, I describe how to integrate control signals into each of these models, significantly increasing the power and range of ap-

plication of these networks.

2 Framework

This section briefly summarizes the methods described in Eliasmith and Anderson (2003), which I will refer to as the Neural Engineering Framework (NEF). Subsequent sections discuss the application of these methods to the construction and control of attractor networks. The following three principles describe the approach:

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 control theoretic state variables. Thus, the dynamics of neurobiological systems can be analyzed using control 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.

Each of the next three sections describes one of the principles, in the context of the addendum, in more detail. For detailed justifications of these principles, please see

Eliasmith and Anderson (2003). They are presented here to make clear both the terminology and assumptions in the subsequent network derivations. In some ways, the successes of the subsequent models helps to justify the adoption of these principles.

2.1 Representation

Consider a population of neurons whose activities $a_i(\mathbf{x})$ encode some vector, $\mathbf{x}$. These activities can be written

$$a_i(\mathbf{x}) = G_i\left[J_i(\mathbf{x})\right], \tag{1}$$

where G_i is the nonlinear function describing the neuron's response function, and $J_i(\mathbf{x})$ is the current entering the soma. The somatic current is given by

$$J_i(\mathbf{x}) = \alpha_i \left\langle \mathbf{x} \cdot \tilde{\boldsymbol{\phi}}_i \right\rangle + J_i^{bias} \tag{2}$$

where α_i is a gain and conversion factor, $\mathbf{x}$ is the vector variable to be encoded, $\tilde{\boldsymbol{\phi}}_i$ determines the 'preferred stimulus' of the neuron, and J_i^{bias} is a bias current that accounts for background activity. This equation provides a standard description of the effects of a current arriving at the soma of neuron i as a result of presenting a stimulus $\mathbf{x}$.

The nonlinearity G_i which describes the neuron's activity as a result of this current can be left undefined for the moment. In general, it should be determined experimentally, and thus based on the intrinsic physiological properties of the neuron(s) being modeled. The result of applying G_i to the soma current $J_i(\mathbf{x})$ over the range of stimuli gives the neuron's tuning curve $a_i(\mathbf{x})$. So, $a_i(\mathbf{x})$ defines the *encoding* of the stimulus into neural activity.

Given this encoding, the original stimulus vector can be estimated by decoding

those activities, e.g.

$$\hat{\mathbf{x}} = \sum_i a_i(\mathbf{x})\boldsymbol{\phi}_i. \tag{3}$$

These decoding vectors, $\boldsymbol{\phi}_i$, can be found by a least-squares method (see 5; (Salinas and Abbott 1994; Eliasmith and Anderson 2003)). Together, the nonlinear encoding in (1) and the linear decoding in (3) define a neural 'population code' for the representation of $\mathbf{x}$.

To incorporate a temporal code into this population code, we can draw on work that has shown that most of the information in neural spike trains can be extracted by linear decoding (Rieke et al. 1997). Let us first consider the temporal code in isolation by taking the neural activities $a_i(t)$ to be decoded spike trains, i.e.

$$a_i(t) = \sum_n h_i(t) * \delta_i(t - t_n) = \sum_n h_i(t - t_n), \tag{4}$$

where $\delta_i(\cdot)$ are the spikes at times $t_n$for neuron i, and $h_i(t)$ are the linear decoding filters which, for reasons of biological plausibility, we can take to be the (normalized) post-synaptic currents (PSCs) in the subsequent neuron. Elsewhere it has been shown that the information loss under this assumption is minimal, and can be alleviated by increasing population size (Eliasmith and Anderson 2003). As before, the encoding on which this linear decoding operates is defined as in (1), where G_i is now taken to be a spiking nonlinearity.

We can combine this temporal code with the previously defined population code to give a general population-temporal code for vectors:

$$\begin{aligned} &\delta(t - t_{in}) = G_i\left[\alpha_i \left\langle \mathbf{x} \cdot \tilde{\boldsymbol{\phi}}_i \right\rangle + J_i^{bias}\right] && \text{Encoding} \\ &\hat{\mathbf{x}} = \textstyle\sum_{i,n} h_i(t - t_n)\boldsymbol{\phi}_i && \text{Decoding} \end{aligned}$$

For notational simplicity, subsequent derivations include the PSC filtering in the

encoding process, unless otherwise necessary. This gives:

$$a_i(\mathbf{x}) = G_i\left[\alpha_i \left\langle \mathbf{x} \cdot \tilde{\boldsymbol{\phi}}_i \right\rangle + J_i^{bias}\right] \quad \text{Encoding}$$
$$\hat{\mathbf{x}} = \textstyle\sum_i a_i(\mathbf{x})\boldsymbol{\phi}_i \quad \text{Decoding}$$

2.2 Transformation

For such representations to be useful they must be used to define *transformations* (i.e. functions of the vector variables). Fortunately, we can again find (least-squares optimal) decoders $\boldsymbol{\phi}_i^{f(\mathbf{x})}$ to perform a transformation $f(\mathbf{x})$. So, instead of finding the optimal decoders $\boldsymbol{\phi}_i$ to extract the originally encoded variable $\mathbf{x}$ from the encoding, we can 're-weight' the decoding to give some function $f(\mathbf{x})$ other than identity (see Appendix 5). To distinguish the 'representational' decoders $\boldsymbol{\phi}_i$ from $\boldsymbol{\phi}_i^{f(\mathbf{x})}$, I refer the latter as 'transformational' decoders.

Given this characterization, it is a simple matter to re-write the encoding and decoding equations for estimating some function of the vector variable:

$$\delta(t - t_{in}) = G_i\left[\alpha_i \left\langle \mathbf{x} \cdot \tilde{\boldsymbol{\phi}}_i \right\rangle + J_i^{bias}\right] \quad \text{Encoding}$$
$$\hat{f}(\mathbf{x}) = \textstyle\sum_{i,n} h_i(t - t_n)\boldsymbol{\phi}_i^{f(\mathbf{x})} \quad \text{Decoding}$$

Notably, both linear and nonlinear functions of the encoded variable can be computed in this manner (see Eliasmith and Anderson (2003) for further discussion).

2.3 Dynamics

Dynamics of neural systems can be described using the previous characterizations of representation and transformation by employing modern control theory. Specifically, we can allow the 'higher-level' vector variables represented by a neural population to be control theoretic state variables.

Let us first consider linear, time-invariant (LTI) systems. Recall that the state equation in modern control theory describing LTI dynamics is

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t). \tag{5}$$

The input matrix $\mathbf{B}$ and the dynamics matrix $\mathbf{A}$ completely describe the dynamics of the LTI system, given the state variables $\mathbf{x}(t)$ and the input $\mathbf{u}(t)$. Taking the Laplace transform of (5) gives:

$$\mathbf{x}(s) = h(s)\left[\mathbf{A}\mathbf{x}(s) + \mathbf{B}\mathbf{u}(s)\right],$$

where $h(s) = \frac{1}{s}$. Any LTI control system can be written in this form.

In the case of the neural system, the transfer function $h(s)$ is not $\frac{1}{s}$, but is determined by the intrinsic properties of the component cells. Because it is reasonable to assume that the dynamics of the synaptic PSC dominate the dynamics of the cellular response as a whole (Eliasmith and Anderson 2003), it is reasonable to characterize the dynamics of neural populations based on their synaptic dynamics, i.e. using $h_i(t)$ from (4).

A simple model of a PSC is given by

$$h'(t) = \frac{1}{\tau}e^{-t/\tau}, \tag{6}$$

where τ is the synaptic time constant. The Laplace transform of this filter is:

$$h'(s) = \frac{1}{1 + s\tau}.$$

Given the change in filters from $h(s)$ to $h'(s)$, we now need to determine how to change $\mathbf{A}$ and $\mathbf{B}$ in order to preserve the dynamics defined in the original system (i.e.

the one using $h(s)$). In other words, letting the neural dynamics be defined by $\mathbf{A}'$ and $\mathbf{B}'$, we need to determine the relation between matrices $\mathbf{A}$ and $\mathbf{A}'$ and matrices $\mathbf{B}$ and $\mathbf{B}'$ given the differences between $h(s)$ and $h'(s)$. To do so, we can solve for $\mathbf{x}(s)$ in both cases and equate the resulting expressions for $\mathbf{x}(s)$. Doing so gives

$$\mathbf{A}' = \tau\mathbf{A} + \mathbf{I} \tag{7}$$

$$\mathbf{B}' = \tau\mathbf{B}. \tag{8}$$

This procedure assumes nothing about $\mathbf{A}$ or $\mathbf{B}$, so we can construct a neurobiologically realistic implementations of any dynamical system defined using the techniques of modern control theory applied to LTI systems (constrained by the neurons' intrinsic dynamics). Note also that this derivation is independent of the spiking nonlinearity, since that process is both very rapid, and dedicated to encoding the resultant somatic voltage (not filtering it in any significant way). Importantly, the same approach can be used to characterize the broader class of time-varying and nonlinear control systems, which I provide examples of below.

2.4 Synthesis

Combining the preceding characterizations of representation, transformation, and dynamics results in the generic neural subsystem shown in figure 1. With this formulation, the synaptic weights needed to implement some specified control system can be directly computed. Note that the formulation also provides for simulating the same model at various levels of description (e.g., at the level of neural populations or individual neurons, using either rate neurons or spiking neurons, and so on). This is useful for fine-tuning a model before introducing the extra computational overhead involved

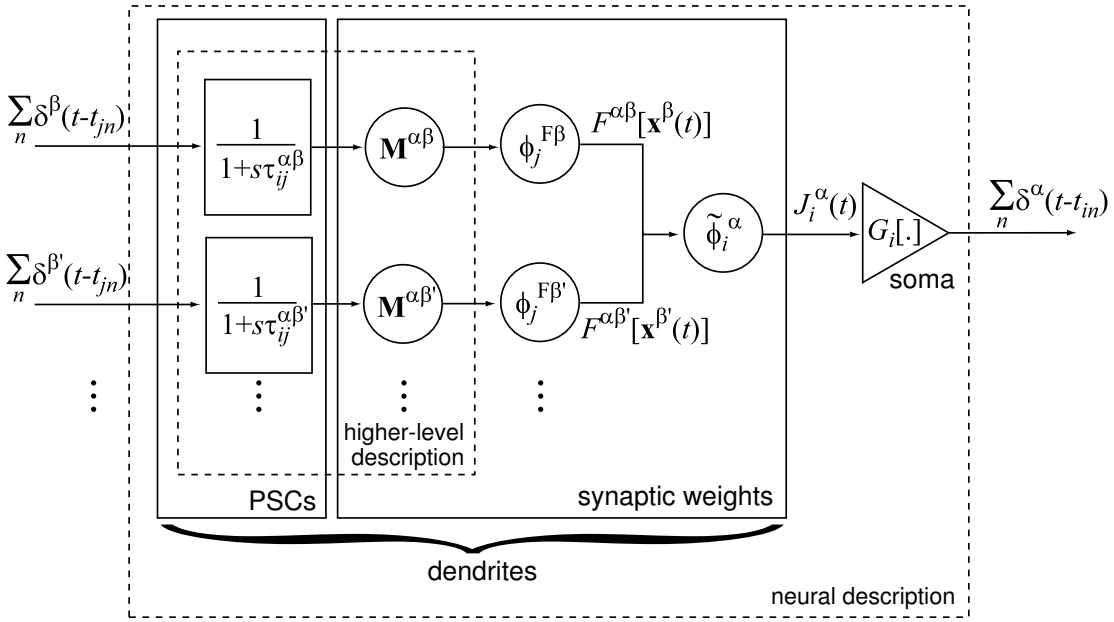

Figure 1: A generic neural population systems diagram. This figure is a combination of a higher-level (control theoretic) and a neural-level system description (denoted by dotted lines). The solid boxes highlight the dendritic elements. The rightmost solid box decomposes the synaptic weights into the relevant matrices. See text for further discussion. Adapted from Eliasmith and Anderson (2003).

with modeling complex spiking neurons.

In figure 1, the solid lines highlight the dendritic components, which have been separated into postsynaptic filtering by the PSCs (i.e., temporal decoding), and the synaptic weights (population decoding/encoding). The weights themselves are determined by three matrices: 1) the decoding matrix whose elements are the decoders $\phi_i^{F\beta}$ for some (nonlinear) function F of the signal $\mathbf{x}^{\beta}(t)$ that comes from a preceding population β; 2) the encoding matrix whose elements are the encoders $\tilde{\phi}_i^{\alpha}$ for this population; and 3) the generalized transformation matrix $\mathbf{M}^{\alpha\beta}$ that defines the transformations necessary for implementing the desired control system.

Essentially, this diagram summarizes the three principles and their interrelations.

The generality of the diagram hints at the generality of the underlying framework. In the remainder of the paper, however, I focus only on its application to the construction and control of attractor networks.

3 Building attractor networks

The most obvious feature of attractor networks is their tendency towards dynamic stability. That is, given a momentary input, they will 'settle' on a position, or a recurring sequence of positions, in state space. This kind of stability can be usefully exploited by biological systems in a number of ways. For instance, it can help a system react to environmental changes on multiple time scales. That is, stability permits systems to act on longer time scales than they might otherwise be able to, which is essential for numerous behaviors including prediction, navigation, and social interaction. In addition, stability can be used as an indicator of task completion, such as in the case of stimulus categorization (Hopfield 1982). As well, stability can make the system more robust; i.e., more resistant to undesirable perturbations. Because these networks are constructed so as to have only a certain set of stable states, random perturbations to nearby states can quickly dissipate to a stable state. As a result, attractor networks have been shown to be effective for noise reduction (Pouget et al. 1998). Similarly, attractors over a series of states (e.g. cyclic attractors) can be used to robustly support repetitive behaviors such as walking, swimming, flying, or chewing.

Given these kinds of useful computational properties, and their natural analogs in biological behavior, it is unsurprising that attractor networks have become a staple of computational neuroscience. More than this, as the complexity of computational models continues to increase, attractor networks are likely to form important sub-networks

in larger models. This is because the ability of attractor networks to, for example, categorize, filter noise, and integrate signals, makes them good candidates for being some of the basic building blocks of complex signal processing systems. As a result, the networks described here should prove useful for a wide class of more complex models.

To maintain consistency, all of the results of subsequent models were generated using networks of leaky integrate-and-fire neurons with absolute refractory periods of 1ms, membrane time constants of 10ms, and synaptic time constants of 5ms.[1] Intercepts and maximum firing rates were chosen from even distributions. The intercept intervals are normalized over $[-1, 1]$ unless otherwise specified. For a detailed discussion of the effects of changing these parameters, see Eliasmith and Anderson (2003).

For each example presented below, the presentation focuses specifically on the *construction* of the relevant model. As a result, there is minimal discussion of the justification for mapping particular kinds of attractors onto various neural systems and behaviors, although references are provided.

3.1 Line attractor

The line attractor, or 'neural integrator', has recently been implicated in decision making (Shadlen and Newsome 2001), but most extensively explored in the context of oculomotor control (Fukushima et al. 1992; Seung 1996; Askay et al. 2001). It is interesting to note that the terms 'line attractor' and 'neural integrator' actually describe different aspects of the network. In particular, the network is called an 'integrator' because the low-dimensional variable (e.g., horizontal eye position) $x(t)$ describing the network's output reflects the integration of the input signal (e.g., eye movement veloc-

[1] Note that this choice of a very short synaptic time constant makes it more difficult to achieve stability in recurrent networks. Many models assume synaptic time constants on the order of 100ms or longer.

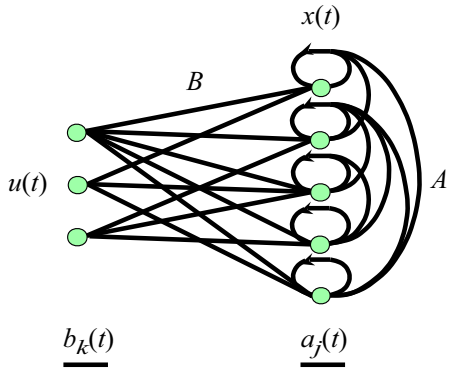

Figure 2: Line attractor network architecture. The underline denotes variables that are part of the neuron-level description. The remaining variables are part of the higher-level description.

ity) $u(t)$ to the system. In contrast, the network is called a 'line attractor' because in the high-dimensional activity space of the network (where the dimension is equal to the number of neurons in the network), the organization of the system collapses network activity to lie on a one-dimensional subspace (i.e. a line). As a result, only input that moves the network along this line changes the network's output.

In a sense, then, these two terms reflect a difference between what can be called 'higher-level' and 'neuron-level' descriptions of the system (see figure 2). As modelers of this system, we need a method that allows us to integrate these two descriptions. Adopting the principles outlined earlier does precisely this. Notably, the resulting derivation is extremely simple, and is similar to that already presented in Eliasmith and Anderson (2003). However, all of the steps need to generate the far more complex circuits discussed later are described here, so it is a useful introduction (and refered to for some of the subsequent derivations).

We can begin by describing the higher-level behavior as integration, which has the state equation

$$\dot{x} = Ax(t) + Bu(t) \tag{9}$$

$$x(s) = \frac{1}{s}\left[Ax(s) + Bu(s)\right], \tag{10}$$

where $A = 0$ and $B = 1$. Given principle 3, we can determine the A' and B', which are needed to implement this behavior in a system with neural dynamics defined by $h'(t)$ (see (6)). The result is

$$B' = \tau$$

$$A' = 1,$$

where τ is the time constant of the PSC of neurons in the population representing $x(t)$.

To use this description in a neural model, we must define the representation of the state variable of the system, i.e., $x(t)$. Given principle 1, let us define this representation using the following encoding and decoding:

$$a_j(t) = G_j\left[\alpha_j \left\langle x(t)\tilde{\phi}_j \right\rangle + J_j^{bias}\right] \tag{11}$$

and

$$\hat{x}(t) = \sum_j a_j(t)\phi_j^x. \tag{12}$$

Note that the encoding weight $\tilde{\phi}_j$ plays the same role as the encoding vector in (2), but is simply ± 1 (for 'on' and 'off' neurons) in the scalar case. Figure 3 shows a population of neurons with this kind of encoding. Let us also assume an analogous representation for $u(t)$.

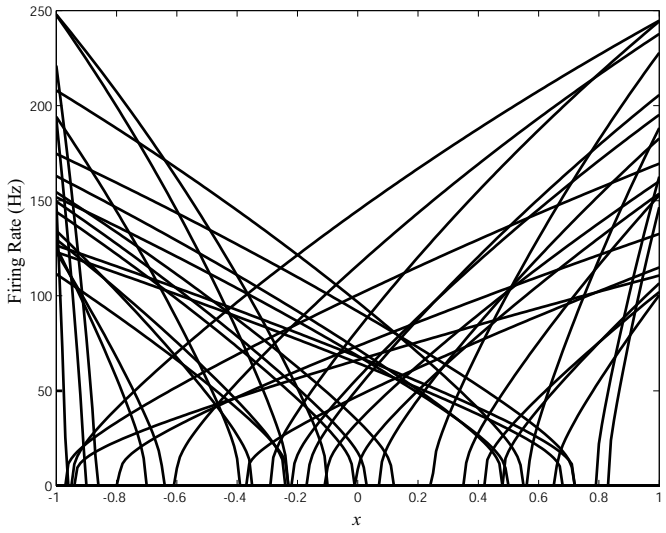

Figure 3: Sample tuning curves for a population of neurons used to implement the line attractor. These are the equivalent steady state tuning curves of the spiking neurons used in this example. They are found by solving the differential equations for the LIF neuron assuming a constant input current, and are described by: $a_j(x) = \frac{1}{\tau_j^{ref} - \tau_j^{RC} \ln\left(1 - \frac{J_{threshold}}{\alpha_j x + J^{bias}}\right)}$.

Working in the time domain, we can take our description of the dynamics,

$$x(t) = h'(t) * [A'x(t) + B'u(t)]$$

and substitute it into (11), to give

$$a_j(t) = G_j \left[\alpha_j \left\langle \tilde{\phi}_j h'(t) * [A'x(t) + B'u(t)] \right\rangle + J_j^{bias}\right]. \tag{13}$$

Substituting our decoding (12) into (13) for both populations gives

$$\begin{aligned} a_j(t) &= G_j \left[\alpha_j \left\langle h'(t) * \tilde{\phi}_j \left[A' \sum_i a_i(t)\phi_i^x + B' \sum_k b_k(t)\phi_k^u\right] \right\rangle + J_j^{bias}\right] && (14) \\ &= G_j \left[h'(t) * \left[\sum_i \omega_{ji} a_i(t) + \sum_k \omega_{jk} b_k(t)\right] + J_j^{bias}\right] && (15) \end{aligned}$$

where $\omega_{ji} = \alpha_j A' \phi_i^x \tilde{\phi}_j$ and $\omega_{jk} = \alpha_j B' \phi_k^u \tilde{\phi}_j$ are the recurrent and input connection weights respectively. Note that i is used to index population activity at the previous 'time step'[2] and G_i is a spiking nonlinearity. It is important to keep in mind that the temporal filtering is only done once, despite this notation. That is, $h'(t)$ is the same filter as that defining the decoding of both $x(t)$ and $u(t)$. More precisely, this equation should be written as

$$\begin{aligned} \sum_n \delta_j(t - t_n) &= G_j \left[\sum_{i,n} \omega_{ji} h_i'(t) * \delta_i(t - t_n) + \ldots \right. \\ &\qquad \left. \sum_{k,n} \omega_{jk} h_k'(t) * \delta_k(t - t_n) + J_j^{bias}\right]. \end{aligned} \tag{16}$$

[2] In fact, there are no discrete time steps since this is a continuous system. However, the PSC effectively acts as a time step, as it determines the length of time that previous information is available.

The dynamics of this system are as written in (13) expected when $h_i'(t) = h_k'(t)$, which is the case of most interest as it best approximates a true integrator. Neverthless, they do not have to be equal and model a broader class of dynamics when this is included in the higher-level analysis.

For completeness, we can write the sub-threshold dynamical equations for an individual LIF neuron voltage $V_j(t)$ in this population as follows:

$$\begin{aligned} \frac{dV_j}{dt} &= -\frac{1}{\tau_j^{RC}} \left(V_j - R_j \left[\sum_{i,n} \omega_{ji} h_i'(t) * \delta_i(t - t_n) + \ ... \right. \right. \\ &\qquad \left. \left. \sum_{k,n} \omega_{jk} h_k'(t) * \delta_k(t - t_n) + J_j^{bias} \right] \right) \end{aligned} \tag{17}$$

where $\tau_j^{RC} = R_j C_j$, $R_j$is the membrane resistance and $C_j$is the membrane capacitance. As usual, the spikes are determined by choosing a threshold voltage for the LIF neuron (V_{th}) and placing a spike when $V_j > V_{th}$. In our models, we also include a refractory time constant τ_j^{ref}, which captures the absolute refractory period observed in real neurons. Figure 4 shows a brief sample run for this network.

To gain insight into the network's function, both as an attractor and an integrator, it is important to derive measures of the network's behavior. This has already been done to some extent for line attractors, so I will not discuss such measures in here (Seung et al. 2000; Eliasmith and Anderson 2003). What these analyses make clear, however, is how higher-level properties, such as the effective time constant of the network, are related to neuron-level properties, such as membrane and synaptic time constants. Because the previous derivation is part of a general method for building more complex attractor networks (as I discuss next), it becomes evident how these same analyses can apply in the more complex cases. This is a significant benefit of generating models with a

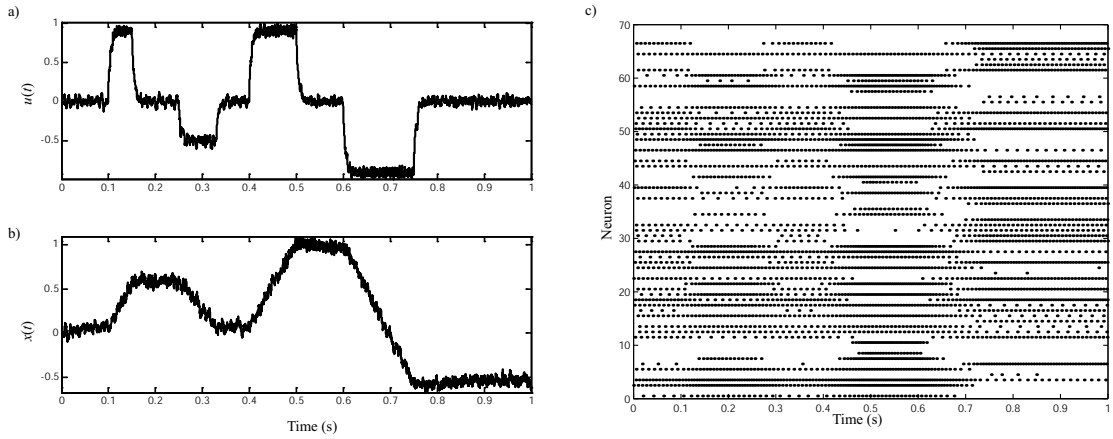

Figure 4: a) The decoded input $u(t)$, b) the decoded integration $x(t)$ of a spiking line attractor with 200 neurons under 10% noise, and c) spike rasters of a third of the neurons in the population.

set of unified principles. More importantly from a practical standpoint, constructing this network by employing control theory makes it evident how to control some of the high-level properties, such as the effective network time constant (see section 4). It is this kind of control that begins to make clear how important such simple networks are for understanding neural signal processing.

3.2 Plane attractor

Perhaps the most obvious generalization of a line attractor is to a plane attractor. That is, from a one-dimensional to a multi-dimensional attractor. In this section I perform this generalization. However, to demonstrate the representational generality of the NEF, I consider plane attractors in the context of function representation (rather than vector representation). This has the further benefit of demonstrating how the ubiquitous 'bump' attractor networks relate to the methods described here.

Networks that have sustained Gaussian-like bumps of activity have been posited in

various neural systems including frontal working memory areas (Laing and Chow 2001; Brody et al. 2003), the head direction system (Zhang 1996; Redish 1999), visual feature selection areas (Hansel and Sompolinsky 1998), and arm control systems (Snyder et al. 1997). The prevalence of this kind of attractor suggests that it is important to account for such networks in a general framework. However, it is not immediately obvious how the stable representation of functions relates to either vector or scalar representation as I have so far described it.

Theoretically, continuous function representation demands an infinite number of degrees of freedom. Neural systems, of course, are finite. As a result, it is natural to assume that understanding function representation in neural systems can be done using a finite basis for the function space accessible by that system. Given a finite basis, the finite set of coefficients of such a basis determine the function being represented by the system at any time. As a result, function representation can be treated as the representation of a *vector* of coefficients over some basis. That is,

$$x(\nu; \mathbf{A}) = \sum_{m=1}^{M} x_m \Phi_m(\nu) \tag{18}$$

where $\nu$is the dimension the function is defined over (e.g., spatial position), $\mathbf{x}$is the vector of M coefficients x_m, and Φ_m are the set of M orthonormal basis functions needed to define the function space. Notably, this basis does not need to be accessible in any way by the neural system itself, it is merely a way for us to conveniently write the function space that is represented by the system.

The neural representation depends on an overcomplete representation of this same space, where the overcomplete basis is defined by the tuning curves of the relevant neurons. More specifically, we can define the encoding of a function space analogously

to that for a vector space by writing

$$a_i(x(\nu;\mathbf{x})) = a_i(\mathbf{x}) = G_i\left[\alpha_i \left\langle x(\nu;\mathbf{x})\tilde{\phi}_i(\nu)\right\rangle_\nu + J_i^{bias}\right]. \tag{19}$$

Here, the encoded function $x(\nu;\mathbf{x})$ (e.g., a Gaussian-like bump) and the encoding function $\tilde{\phi}_i(\nu)$ which is inferred from the tuning curve, determine the activity of the neuron (see figure 6 for an example). Because of the integration over ν in this encoding, it is only changes in the coefficients $\mathbf{x}$ that affect neural firing – again making it clear that we can treat neural activity as encoding a vector of coefficients. The decoding for function representation is as expected:

$$\hat{x}(\nu;\mathbf{x}) = \sum_i a_i(\mathbf{x})\phi_i(\nu) \tag{20}$$

where the decoders $\phi_i(\nu)$ can be determined similarly to the vector decoders discussed earlier.

Having defined function representation in this way, we are in a position to relate it to an equivalent vector representation. This is important because it allows us to use the control theoretic techniques discussed in section 2.3 to define the dynamics of the representation. Let us begin by writing the decoders $\phi_i(\nu)$ using the orthonormal basis that defines the function space $x(\nu;\mathbf{x})$:

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

where q_{im} are elements of the matrix of coefficients defining each of the i encoding functions with respect to the basis $\Phi_m(\nu)$. This ensures that the representation of a given function will not lie outside the original function space. Similarly, the encoding

functions $\tilde{\phi}_i(\nu)$ should only encode functions $x(\nu;\mathbf{x})$ in such a way that they can be decoded by these decoders, so we may assume

$$\tilde{\phi}_i(\nu) \;=\; \sum_m^M \tilde{q}_{im}\Phi_m(\nu). \tag{21}$$

Together, these definitions determine the equivalent of the function representation in a vector space. In particular, the encoding is given by

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

and the decoding is given by

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

Essentially, these equations simply convert the encoding and decoding *functions* into their equivalent encoding and decoding vectors in the function space whose dimensions are determined by Φ_m. So, the description of a neural system in the vector space has the same properties as it did in the original function space. The advantage, as I have mentioned, is that control theory is more easily applied to finite vector spaces. When I introduce control in section 4, I demonstrate this advantage in more detail.

To see the utility of this formulation, let us consider two different neural systems in parallel: working memory in lateral intraparietal (LIP) cortex and the ring attractor in

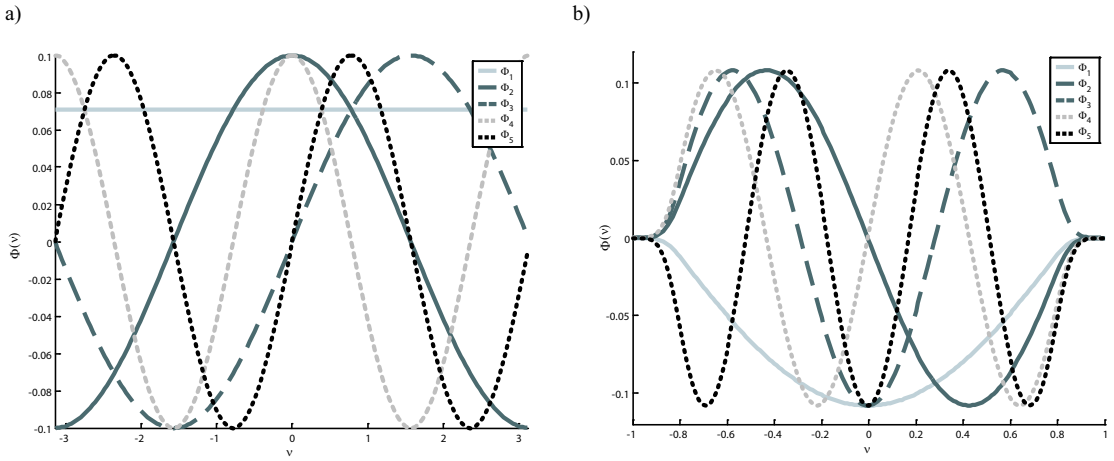

Figure 5: The orthonormal bases for the representational spaces for a) the ring attractor in the head direction system and b) LIP working memory. Note that the former is cyclic and the later is not.

the head direction system.[3] These can be considered in parallel because the dominant dynamics in both systems are the same. That is, just like the integrator, LIP working memory and the ring attractor maintain a constant value with no input, i.e. $\dot{\mathbf{x}} = 0$.

The difference between these systems is that LIP working memory is sometimes taken to have a bounded domain of representation (e.g., between ± 60 degrees from midline), whereas the ring attractor has a cyclic domain of representation. Given equation (18), this difference will show up as a difference in the orthonormal basis $\Phi(\nu)$ that spans the representational space. Figure 5 shows this difference.

In fact, these orthonormal basis functions can be inferred from the neural data. To do so, we first need to construct the neural encoding functions $\tilde{\phi}_i(\nu)$ by looking at the experimentally determined tuning curves of the neurons. That is, we can generate a population of neurons with tuning curves that have the same distribution of widths and heights as observed in the system we are modeling, and then use the encoding

[3]The first of these is considered in Eliasmith and Anderson (2003), although without the integration of the Platt and Glimcher (1998) data as described here.

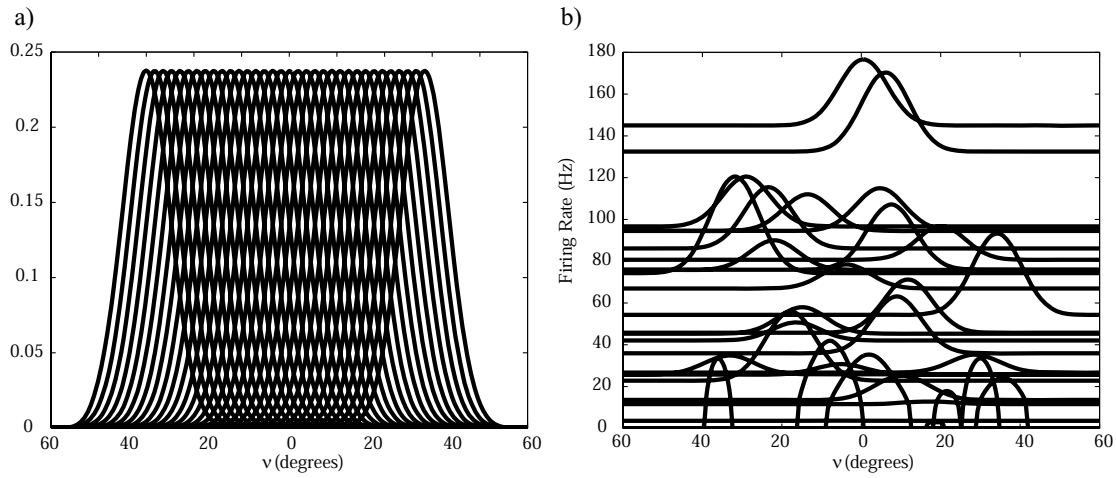

Figure 6: Samples of the a) encoding functions and b) tuning curves for LIP neurons based on the data provided in Platt and Glimcher (1998).

functions necessary for generating that population as an overcomplete basis for the representational space. We can then determine, using singular value decomposition, the orthonormal basis that spans that same space, and use it as our orthonormal basis, $\Phi(\nu)$. For example, using data from Platt and Glimcher (1998), I have applied this method to get the tuning curves shown in figure 6a, the encoders shown in figure 6b, and thus arrive at an orthonormal basis similar to that shown in figure 5b for LIP representation.

Given an orthonormal basis, we now need to determine which set of coefficients $\mathbf{x}$ on that basis are relevant for the neural system of interest. The standard representations in both LIP working memory and the ring attractor are generally characterized as Gaussian-like bumps, at any position. However, in LIP there is evidence that this bump can be further parameterized by non-positional dimensions (Sereno and Maunsell 1998). This kind of 'parametric' working memory has also been found in frontal systems (Romo et al. 1999). So, the representation in LIP will be Gaussian-like bumps,

but of varying heights.

Ideally, we need to specify some probability density $\rho(\mathbf{x})$ on the coefficients that appropriately picks out just the Gaussian-like functions centered at every value of ν (and those of various heights for the LIP model). This essentially specifies the range of functions that are permissible in our function space. It is clearly undesirable to have all functions that can be represented by the orthonormal basis as candidate representations. Given $\rho(\mathbf{x})$ we can use the methods described in appendix 5 to find the decoders. In particular, the average over $\mathbf{x}$ in equation (31) is replaced by an average over $\rho(\mathbf{x})$. In practice, it can be difficult to compactly define $\rho(\mathbf{x})$, so it is often convenient to use a Monte Carlo method for approximating this distribution when performing the average, which I have done for these examples.

Having fully defined the representational space for the system, we can apply the methods described earlier for the line attractor to generate a fully recurrent spiking model of these systems. The resulting behavior of these two models are shown in figure 7. Note that although it is natural to interpret the behavior of this network as a function attractor (as demonstrated by the activity of the population of neurons), the model can also be understood as implementing a (hyper-)plane attractor in the $\mathbf{x}$ vector space.

Being able to understand the models of two different neural systems with one approach can be very useful. This is because we can transfer analyses of one model to the other with minimal effort. As well, it highlights possible basic principles of neural organization (in this case integration). These observations provide a first hint that the NEF stands to unify diverse descriptions of neural systems and help identify general principles underlying the dynamics and representations in those systems.

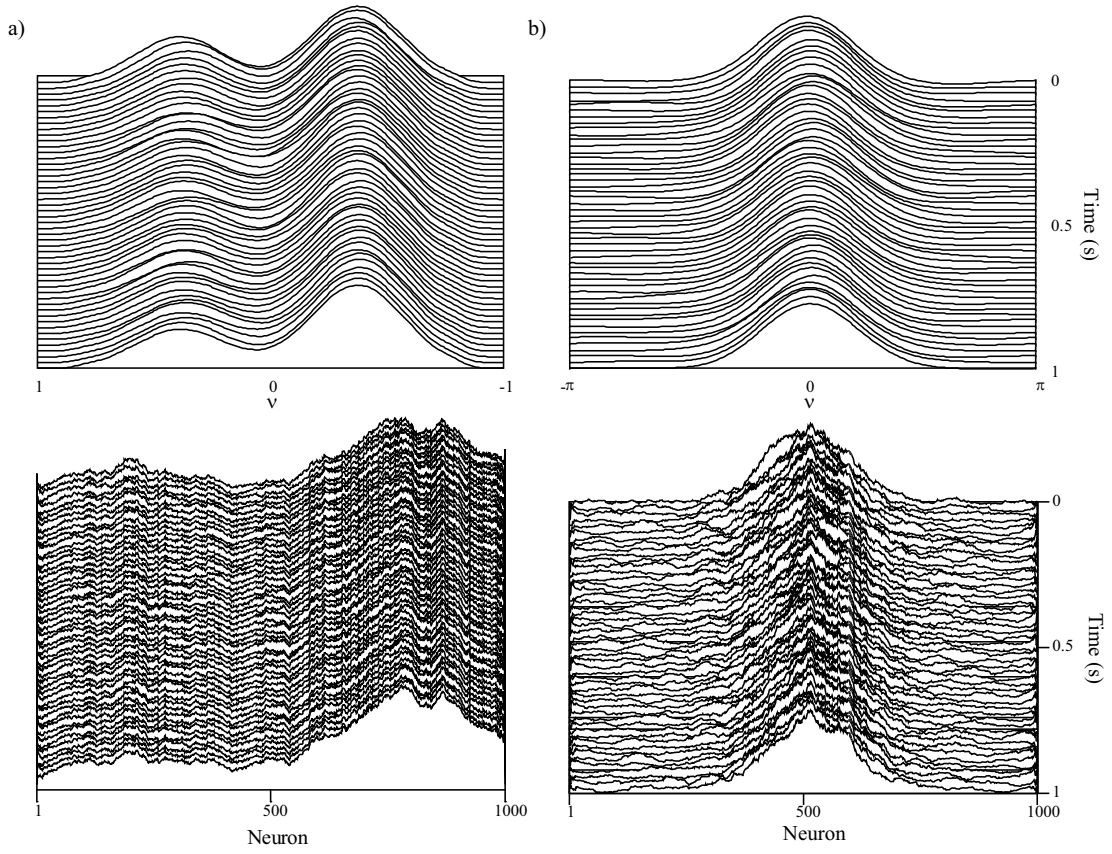

Figure 7: Simulation results for a) LIP working memory encoding two different bump heights at two locations and b) a head-direction ring attractor. The top graph shows the decoded function representation, and the bottom graph shows the activity of the neurons in the population (the activity plots are calculated from spiking data using a 20ms time window with background activity removed, and are smoothed by a 50 point moving average). Both models use 1000 spiking LIF neurons with 10% noise added.

3.3 Cyclic attractor

The kinds of attractors I have presented to this point are, in a sense, static because once the system has settled to a stable point, it will remain there unless perturbed. However, there is another broad class of attractors that have dynamic, periodic stability. In such cases, settling into the attractor results in a cyclic progression through a closed set of points. The simplest example of this kind of attractor is the ideal oscillator.

Because cyclic attractors are used to describe oscillators, and many neural systems seem to include oscillatory behavior, it is natural to use cyclic attractors to describe oscillatory behavior in neural systems. Such behavior may include any repetitive motion such as walking, swimming, flying, chewing, and so on. The natural mapping between oscillators and repetitive behavior is at the heart of most work on central pattern generators (CPGs; Selverston 1980; Kopell and Ermentrout 1988). However, this work typically characterizes oscillators as interactions between only a few neighboring neurons. In contrast, the NEF can help us in understanding cyclic attractors at the network level. Comparing the results of a NEF characterization with that of the standard approach to CPGs shows that there are advantages to the higher-level characterization. To effect this comparison, let us extend a previously described model of lamprey swimming (for more detail of the mechanical model, see Eliasmith and Anderson 2000; Eliasmith and Anderson 2003).[4] Later, I extend this model by introducing control.

When the lamprey swims, the resulting motion resembles a standing wave of one period over the lamprey's length. The tensions T in the muscles needed to give rise to

[4]Unlike the previous model, this one includes noisy spiking neurons. Parameters for these neurons and their distribution are based on data in Manira et al. (1994). This effectivley demonstrates that the bursting observed in lamprey spinal cord is observed in the model as well.

this motion can be described by:

$$T(z,t) = \kappa(\sin(\omega t - kz) - \sin(\omega t)), \tag{22}$$

where $\kappa = \frac{\gamma\eta\omega A}{k}$, $k = \frac{2\pi}{L}$, $A = 1$ is the wave amplitude, $\eta = 1$ is the normalized viscosity coefficient, $\gamma = 1$ is the ratio of intersegmental and vertebrae length, $L = 1$ is the length of the lamprey, and ω is the swimming frequency.

As for the LIP model, we can define an orthogonal representation of the dynamic pattern of tensions in terms of the coefficients $x_n(t)$ and the harmonic functions $\Phi_n(z)$:

$$\hat{T}(z,t;\mathbf{x}) = \kappa\left(x_0 + \sum_{n=1}^{N} x_{2n-1}(t)\sin(2\pi nz) + x_{2n}(t)\cos(2\pi nz)\right).$$

The appropriate $\mathbf{x}$ coefficients are found by setting the MSE between $\hat{T}(z,t;\mathbf{x})$ and $T(z,t)$ to be zero. Doing so, we find that $x_0(t) = -\sin(\omega t)$, $x_1(t) = -\cos(\omega t)$, $x_2(t) = \sin(\omega t)$, and for $n > 2$, $x_n(t) = 0$. This defines the representation in a higher-level function space, whose dynamics we can implement by describing the dynamics of the coefficients, $\mathbf{x}$.

In this case, it is evident that the coefficients $x_0$and $x_1$implement a standard oscillator. The coefficient x_2 is an additional counter-phase sine wave. This additional term simply tilts the 2-dimensional cyclic attractor in phase space, so we essentially have just a standard oscillator. We can write the control equations as usual

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$$

where

$$\mathbf{A} = \begin{bmatrix} 0 & \omega & 0 \\ -\omega & 0 & 0 \\ 0 & -\omega & 0 \end{bmatrix} \tag{23}$$

for some frequency ω.

Before we embed this control description into a neural population, it makes sense to take into account the known anatomical structure of the system we are describing. In the case of the lamprey, we know that the representation of the tension T is spread down the length of the animal in a series of 100 or so segments (Grillner et al. 1991). As a result, we can define a representation that is intermediate between a neural representation and the orthogonal representation that captures this structure. In particular, let us define an overcomplete representation along the length z with Gaussian-like encoding functions $\tilde{\phi_j}(z)$. The encoding into this intermediate representation is thus

$$b_j(t) = \left\langle \tilde{\phi}_j(z) T(z,t) \right\rangle_z$$

and the decoding is

$$\hat{T}(z,t) = \sum_j b_j(t) \phi_j(z).$$

This representation is not essential, but has one very useful property: it allows us to simulate some parts of the model at the neural level and other parts at this intermediate level, resulting in significant computational savings while *selectively* simplifying the model. Of course, to use this representation, we need to associate the intermediate representation to both the neural and orthogonal representations. The relation to the neural representation is defined by the standard neural representation described in section 2.1,

with the encoding given by

$$\delta(t - t_{in}) = G_i \left[\alpha_i \left\langle b_j \tilde{\phi}_i \right\rangle + J_i^{bias} \right]$$

and the decoding by

$$\hat{b}_j = \sum_{i,n} h_{ij}(t - t_n)\phi_i.$$

Essentially, these equations describe how the intermediate population activity b_j is related to the actual neurons, indexed by i, in the various populations along the length of the lamprey.[5]

To relate the intermediate and orthogonal spaces, we can use the projection operator $\Theta = \left[\tilde{\phi}\Phi\right]$. That is, we can project the high-level control description into the intermediate-level space as follows:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} \\ \Theta\Theta^{-1}\dot{\mathbf{b}} &= \Theta\mathbf{A}\Theta^{-1}\mathbf{b} \\ \dot{\mathbf{b}} &= \mathbf{A_b}\mathbf{b} \end{aligned}$$

where $\mathbf{A_b} = \Theta\mathbf{A}\Theta^{-1}$.

Having provided these descriptions, we can now selectively convert segments of the intermediate representation into spiking neurons to see how single cells perform in the context of the whole, dynamic spinal cord. Figure 8a shows single cells that burst during swimming, and figure 8b shows the average spike rate of an entire population of

[5] Because the lamprey spinal cord is effectively continuous, assignment of neurons to particular populations is somewhat arbitrary, although constrained by the part of the lamprey over which they encode muscle tension. So, the resulting model is similarly continuous as well.

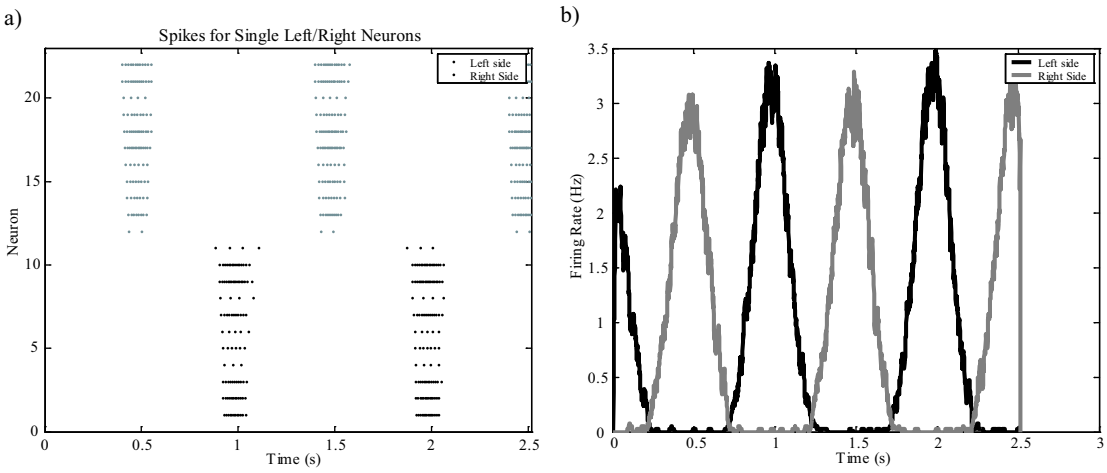

Figure 8: Results from the lamprey modeled as a cyclic attractor. The middle segment of the lamprey was modeled at the single cell level by a population of 200 neurons under 10% noise. The spikes from twenty (10 left, 10 right) single cells in the population are shown in a), and the average rate in the two (left and right) subpopulations is shown in b).

neurons in a segment. Given this characterization of neural representation, these graphs reflect different levels of description of the neural activity during the same simulation.

It should be clear that this model does not adopt the standard CPG approach to modeling this system. As a result, the question arises as to whether the results of this simulation match the known neuroanatomy as well as the CPG approach. While there is much that remains unknown about the lamprey, these three constraints on the anatomy are well-established:

1. connectivity is mostly local, but spans several segments;

2. connectivity is asymmetric; and

3. individual neurons code muscle tension over small regions.

By introducing the intermediate level of representation, we have enforced the third

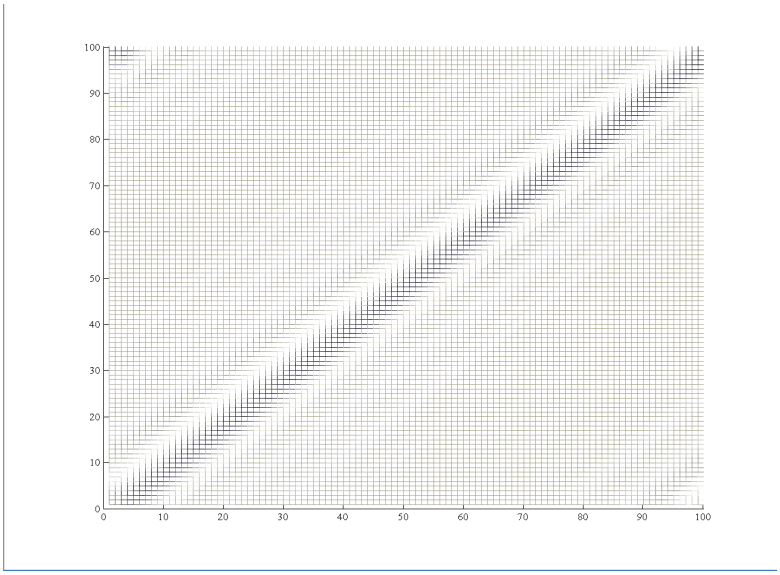

Figure 9: The connectivity matrix between segments in the lamprey model. Connectivity is asymmetric and mostly local, in agreement with the known anatomy. The darker the image, the stronger the weights. Zero weights are the large grey hatched areas.

constraint explicitly. Looking at the intermediate-level weight matrix for this system shown in figure 9, we can see that 2 clearly holds and that 1 is approximately satisfied.

So, the neural engineering framework (NEF) can be used to embed high-level descriptions of cyclic attractors into biologically plausible networks that are consistent with the relevant anatomical constraints. Undoubtedly, different systems will impose different anatomical constraints, but the NEF methods are clearly not determined by the particular constraints found in the lamprey.

Equally important, as I discuss in section 4, because the NEF allows the integration of control into the high-level description it is straightforward to characterize (and enforce) essential high-level properties like stability and controllability in the generated models. This has often proved a daunting task for the standard bottom-up, CPG approach (Marder et al. 1997). So, again, it is the ease with which this model can be

extended to account for important, but complex behaviors that demonstrates the utility of the NEF.

3.4 Chaotic attractor

The final class of attractors I consider are also dynamic attractors, but they, unlike the cyclic attractors, are not periodic. Instead, any nearby trajectories in chaotic (or 'strange') attractors diverge exponentially over time. Nevertheless, they are attractors in that there is a bounded subspace of the state space towards which trajectories, regardless of initial conditions, tend over time.

In the context of neurobiological systems, there have been some suggestions that chaos or chaotic attractors can be useful for describing certain neural systems (Matsugu et al. 1998; Kelso and Fuchs 1995; Skarda and Freeman 1987). For example, Skarda and Freeman (1987) suggest that the olfactory bulb, before odor recognition, rests in a chaotic state. The fact that the state is chaotic rather than merely noisy permits more rapid convergence to limit cycles that aid in the recognition of odors. These kinds of information processing effects themselves are well-documented. For instance, there a number of practical control problems that can be more efficiently solved if a system can exploit chaotic attractors effectively (Bradley 1995). However, the existence of chaos in neural systems is subject to much debate (Lai et al. 2003; Biswal and Dasgupta 2002).

As a result, I consider chaotic attractors here largely for the purposes of completeness. That is, to show that this approach is general enough to capture such phenomena, should they exist. For this example, I have chosen to use the familiar Lorenz attractor,

described by:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} -a & a & 0 \\ b & -1 & -x_1 \\ x_2 & 0 & -c \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{24}$$

If $a = 10$, $b = 28$, and $c = 8/3$ this system of equations gives the well-known 'butterfly' chaotic attractor. It is clear from this set of equations, that the system to be considered is nonlinear. So, unlike the previous examples, we need to compute nonlinear functions of the state variables, meaning this is not an LTI system. As discussed in more detail in section 4.1, there are various possible architectures for computing the necessary transformations.

Here, we can compute the necessary cross-terms by extracting them directly from the population representing the vector space $\mathbf{x}$. Specifically, we can find decoding vectors for x_1x_3 (i.e., $\phi^{x_1x_3}$) and x_1x_2(i.e., $\phi^{x_1x_2}$) using the method discussed in appendix 5 where, e.g., $f(\mathbf{x}) = x_1x_3$. These decoding vectors can be used to provide an expression for the recurrent updating of the population's activity:

$$a_i(\mathbf{x}) = G_i\left[\alpha_i \left\langle \tilde{\phi}_i l(\mathbf{x}) \right\rangle + J_i^{bias}\right] \tag{25}$$

where the vector function $l(\mathbf{x})$is defined by the Lorenz equations in (24). Substituting the appropriate neural-level characterizations of this transformation into (25) gives

$$a_i(\mathbf{x}) = G_i\left[\sum_j \left(\omega_{ij}^{ax_1} - \omega_{ij}^{ax_2} + \omega_{ij}^{bx_1} - \omega_{ij}^{x_2} - \omega_{ij}^{x_1x_3} + \omega_{ij}^{x_1x_2} - \omega_{ij}^{cx_3}\right) a_j(\mathbf{x}) + J_i^{bias}\right]$$

where $\omega_{ij}^{ax_1} = \alpha_i\tilde{\phi}_{i,1}a\phi_j^{x_1}$, $\omega_{ij}^{ax_2} = \alpha_i\tilde{\phi}_{i,1}a\phi_j^{x_1}\omega_{ij}^{bx_1} = \alpha_i\tilde{\phi}_{i,2}b\phi_j^{x_1}$, $\omega_{ij}^{x_2} = \alpha_i\tilde{\phi}_{i,2}\phi_j^{x_2}$, $\omega_{ij}^{x_1x_3} = \alpha_i\tilde{\phi}_{i,2}\phi_j^{x_1x_3}$, $\omega_{ij}^{x_1x_2} = \alpha_i\tilde{\phi}_{i,3}\phi_j^{x_1x_2}$, and $\omega_{ij}^{cx_3} = \alpha_i\tilde{\phi}_{i,3}c\phi_j^{x_3}$. So, as usual, the

connection weights are found by combining the encoding and decoding vectors as appropriate. Note that despite implementing a higher-level nonlinear system, there are no multiplications between neural activities in the neural-level description. This demonstrates that the neural nonlinearities alone result in the nonlinear behavior of the network. That is, no additional nonlinearities (e.g., dendritic nonlinearities) are needed to give rise to this behavior.

Running this simulation in a spiking network of 2000 LIF neurons under noise gives the results shown in figure 10. Because this is a simulation of a chaotic network under noise, it is essential to demonstrate that a chaotic system is in fact being implemented, and the results are not just noisy spiking from neurons. Applying the noise titration method (Poon and Barahona 2001) on the decoded spikes verifies the presence of chaos in the system (P-value $< 10^{-15}$ with a noise limit of 54%, where the noise limit indicates how much more noise could be added before the nonlinearity was no longer detectable). Notably, the noise titration method is much better at detecting chaos in time series than most other methods, even in highly noisy contexts. As a result, we can be confident that the simulated system is implementing a chaotic attractor as expected. This can also be qualitatively verified by plotting the state space of the decoded network activity, which clearly preserves the 'butterfly' look of the Lorenz attractor (figure 10c).

4 Controlling attractor networks

To this point I have demonstrated how three main classes of attractor networks can be embedded into neurobiologically plausible systems, and I have indicated, in each case, which specific systems might be well-modeled by these various kinds of attractors. However, in each case, I have not demonstrated how a neural system could *use* this

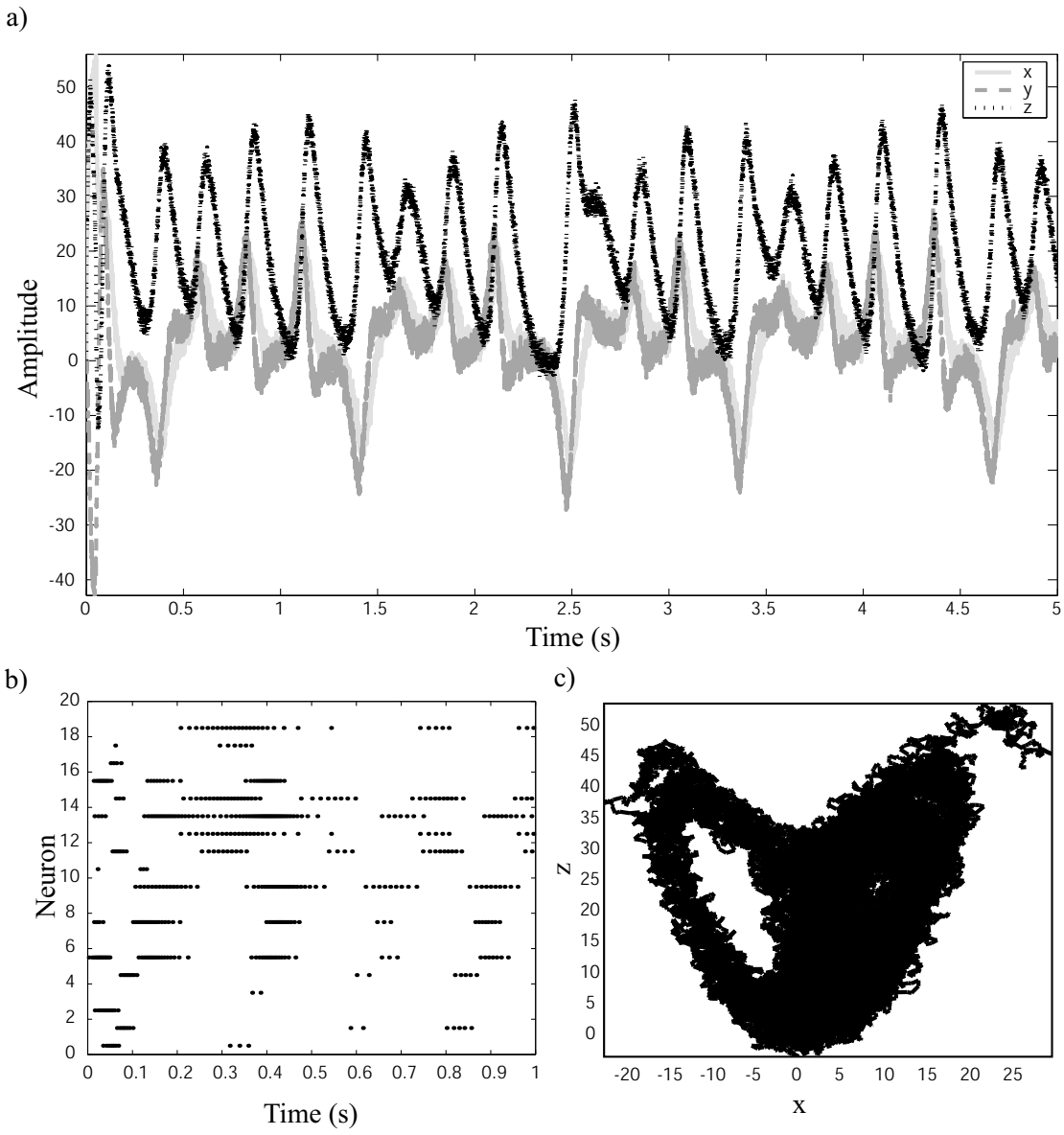

Figure 10: The Lorenz attractor implemented in a spiking network of 2000 LIF neurons under 10% noise. a) The decoded output from the three dimensions. b) Spike trains from 20 sample neurons in the population for the first second of the run display irregular firing. c) Typical Lorenz attractor-type motions (i.e. the 'butterfly' shape) are verified by plotting the state space. For clarity, only the last 4.5s are plotted, removing the startup transients.

kind of structure effectively. Merely having an attractor network in a system is not itself necessarily useful unless the computational properties of the attractor can be taken advantage of. Taking advantage of an attractor can be done by moving the network either into or out of an attractor, moving between various attractor basins, or destroying and creating attractors within the network's state space. Performing these actions means controlling the attractor network in some way. Some of these behaviors can be effected by simply changing the input to the network. But, more generally, we must be able to control the parameters defining the attractor properties.

In what follows, I focus on this second, more powerful, kind of control. Specifically, I revisit examples from each of the three classes of attractor networks and show how control can be integrated into these models. For the neural integrator I show how it can be turned into a more general circuit that acts as a controllable filter. For the ring attractor, I demonstrate how to build a nonlinear control model that moves the current head direction estimate given a vestibular control signal, and which does not rely on multiplicative interactions at the neural level. In the case of the cyclic attractor, I construct a control system that permits variations in the speed of the orbit. And finally, in the case of the chaotic attractor I demonstrate how to build a system that can be moved between chaotic, cyclic, and point attractor regimes.

4.1 The neural integrator as a controllable filter

As described in section 3.1, a line attractor is implemented in the neural integrator in virtue of the dynamics matrix A' being set to 1. While the particular output value of the attractor depends on the input, the dynamics of the attractor are controlled by A'. Hence, it is natural to inquire as to what happens as A' varies over time. Since A' is unity feedback, it is fairly obvious what the answer to this question is: as A' goes

over 1, the resulting positive feedback will cause the circuit to saturate; as A' becomes less than one, the circuit begins to act as a low-pass filter, with the cutoff frequency determined by the precise value of A'. Thus, we can build a tunable filter by using the same circuit and allowing direct control over A'.

To do so, we can introduce another population of neurons d_l that encode the value of $A'(t)$. Because A' is no longer static, the product $A'x$ must be constantly recomputed. This means that our network must support multiplication at the higher level. The two most obvious architectures for building this computation into the network are shown in figure 11. Both architectures are implementations of the same high-level dynamics equation

$$x(t) = h'(t) * (A'(t)x(t) + \tau u(t)) \tag{26}$$

which is no longer LTI, as it is clearly a time-varying system. Notably, while both architectures demand multiplication at the higher level, this does not mean that there needs to be multiplication between activities at the neural level. This is because, as mentioned in section 2.2 and demonstrated in section 3.4, nonlinear functions can be determined using only linear decoding weights.

As described in Eliasmith and Anderson (2003), the first architecture can be implemented by constructing an intermediate representation of the vector $\mathbf{c} = [A', x]$ from which the product is extracted using linear decoding. The result is then used as the recurrent input to the $a_i$population representing x. This circuit is successful, but performance is improved by adopting the second architecture.

In the second architecture, the representation in a_i population is taken to be a 2D representation of $\mathbf{x}$ in which the first element is the integrated input and the second

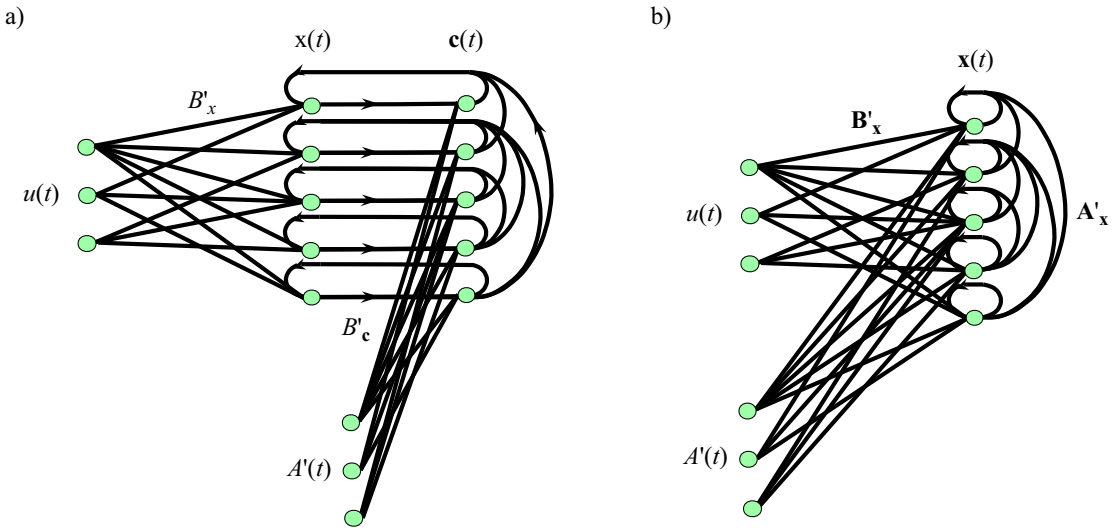

Figure 11: Two possible network architectures for implementing the controllable filter. The B variables modify the inputs to the populations representing their subscripted variable. The A variables modify the relevant recurrent connections. The architecture in a) is considered in Eliasmith and Anderson (2003), the more efficient architecture in b) is considered here.

element is A'. The product is extracted directly from this representation using linear decoding and then used as feedback. This has the advantage over the first architecture of not introducing extra delays and noise.

Specifically, let $\mathbf{x} = [x_1, x_2]$ (where $x_1 = x$ and $x_2 = A'$ in (26)). So, a more accurate description of the higher-level dynamics equation for this system is

$$\begin{aligned} \mathbf{x} &= h' * (\mathbf{A}'\mathbf{x} + \mathbf{B}'\mathbf{u}) \\ \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= h' * \left(\begin{bmatrix} x_2 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} \tau & 0 \\ 0 & \tau \end{bmatrix} \begin{bmatrix} u \\ A' \end{bmatrix} \right) \end{aligned} \tag{27}$$

which makes the nonlinear nature of this implementation explicit. Notably, here the desired A' is provided as input from a preceding population, as is the signal to be integrated, u. To implement this system, we need to compute the transformation

$$\hat{p}(t) = \sum_i a_i(t)\phi_i^p,$$

where $p(t)$ is the product of the elements of $\mathbf{x}$. Substituting this transformation into (14) gives

$$\begin{aligned} a_j &= G_j \left[\alpha_j \left\langle h' * \tilde{\phi}_j \left[\sum_i a_i(t)\phi_i^p + B' \sum_k b_k(t)\phi_k^u \right] \right\rangle + J_j^{bias} \right] \\ &= G_j \left[\sum_i \omega_{ij} a_i(t) + \sum_k \omega_{kj} b_k(t) + J_j^{bias} \right] \end{aligned} \tag{28}$$

where $\omega_{ij} = \alpha_j \tilde{\phi}_j \phi_i^p$, $\omega_{kj} = \alpha_j \tilde{\phi}_j B' \phi_k^u$, and

$$a_i(t) = h' * G_i \left[\alpha_i \left\langle \mathbf{x}(t)\tilde{\boldsymbol{\phi}}_i \right\rangle + J_i^{bias} \right].$$

The results of simulating this nonlinear control system are shown in figure 12. This run demonstrates a number of features of the network. In the first tenth of a second, the control signal $1 - A'$ is non-zero, helping to eliminate any drift in the network for zero input. The control signal then goes to zero, turning the network into a standard integrator over the next two-tenths of a second when a step input is provided to the network. The control signal is then increased to .3, rapidly forcing the integrated signal to zero. The next step input is then filtered by a low pass filter, since the control signal is again non-zero. The third step input is also integrated, as the control signal is zero. Like the first input, this input is forced to zero by increasing the control signal, but this time the decay is much slower because the control signal is lower (.1). These behaviors show how the control signal can be used as a reset signal (by simply making it non-zero), or as a means of determining the properties of a tunable low-pass filter.

So, the introduction of control into the system gave us a means of radically altering the attractive properties of the system. It is only while $A' = 1$ that we have an approximate line attractor. For positive values less than one, the system no longer acts as a line attractor, but rather as a point attractor, whose basin properties (e.g., steepness) vary as the control signal.

As can be seen in (28), there is no multiplication of neural activities. There is, of course, significant debate about whether, and to what extent, dendritic nonlinearities might be able to support multiplication of neural activity (see, e.g., Koch and Poggio 1992; Mel 1999; Mel 1994; Salinas and Abbott 1996; von der Heydt et al. 1991). As a result, it is useful to demonstrate that it is possible to generate circuits without multiplication of neural activities that support network level nonlinearities. If dendritic nonlinearities are discovered in the relevant systems, these networks would become much simpler (essentially we would not need to construct the intermediate $\mathbf{c}$ population).

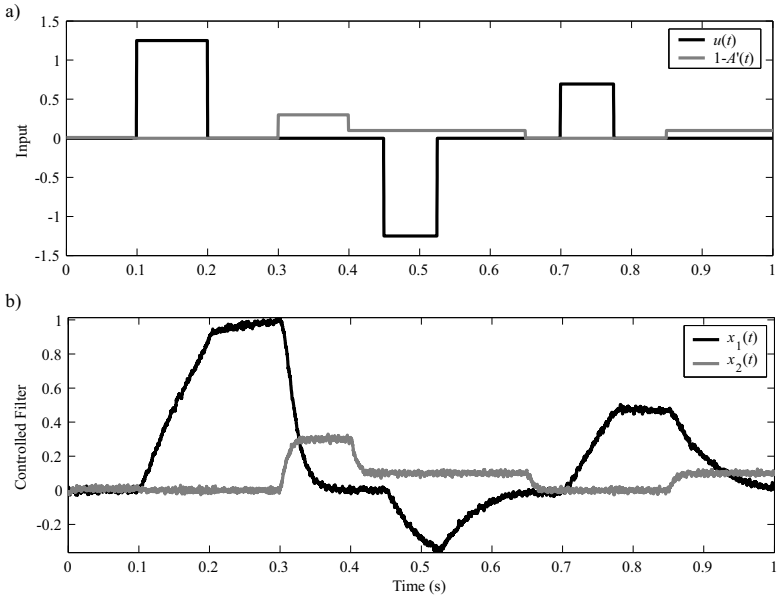

Figure 12: The results of simulating the second architecture for a controllable filter in a spiking network of 2000 neurons under 10% noise. a) The input signals to the network. b) The high-level decoded response of the spiking neural network. The network encodes both the integrated result and the control signal directly, to efficiently support the necessary nonlinearity. See text for a description of the behavior.

4.2 Controlling bump movement around a ring attractor

Of the two models presented in section 3.2, the role of control is most evident for the head direction system. In order to be useful, the head direction system must be able to update its current estimate of the heading of the animal given vestibular information about changes in the animal's heading. In terms of the ring attractor, this means that the system must be able rotate the bump of activity around the ring in various directions and at various speeds given simple left/right angular velocity commands.

To design a system that behaves in this way, we can first describe the problem in the function space $x(\nu)$ for a velocity command δ/τ and then write this in terms of the coefficients $\mathbf{x}$ in the orthonormal Fourier space:

$$\begin{aligned} x(\nu; t+\tau) &= x(\nu+\delta; t) \\ &= \sum_m x_m e^{im(\nu+\delta)} \\ &= \sum_m x_m e^{im\nu} e^{im\delta}. \end{aligned}$$

So, rotation of the bump can be effected by applying the matrix $\mathbf{E}_m = e^{im\delta}$, where δ determines the speed of rotation. Written for real-valued functions, $\mathbf{E}$ becomes

$$\mathbf{E} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(m\delta) & \sin(m\delta) & 0 \\ 0 & -\sin(m\delta) & \cos(m\delta) & \vdots \\ 0 & 0 & \cdots & \ddots \end{bmatrix}.$$

To derive the dynamics matrix $\mathbf{A}$ for the state equation (5), it is important to note that

$\mathbf{E}$ defines the new function at $t + \tau$, not just the change, $\delta\mathbf{x}$. As well, we would like to control the speed of rotation, so we can introduce a scalar on $[-1, 1]$ that changes the velocity of rotation, with δ defining the maximum velocity. Taking these considerations into account, gives

$$\mathbf{A} = C(t)\,(\mathbf{E} - \mathbf{I})$$

where $C(t)$ is the left/right velocity command.[6]

As in section 4.1, we can include the time-varying scalar variable $C(t)$ in the state vector $\mathbf{x}$ and perform the necessary multiplication by extracting a nonlinear function that is the product of that element with the rest of the state vector. Doing so again means that there is no need to multiply neural activities. Constructing a ring attractor without multiplication is a problem that was only recently solved by Goodridge and Touretzky (2000). That solution, however, is specific to a one-dimensional single bump attractor, does not use spiking neurons, and does not include noise. As well, the solution posits single left/right units that together project to every cell in the population, necessitates the setting of normalization factors, and demands numerical experiments to determine the appropriate value of a number of the parameters. In sum, the solution is somewhat non-biological and very specific to the problem being addressed. In contrast, the solution I have presented here is subject to none of these concerns: it is both biologically plausible and based on general principles. The behavior of the fully

[6]There is more subtlety than one might think to this equation. For values of $C(t) \neq 1$, the system does not behave as one might expect. For negative values, two bumps are created: one negative bump in the direction opposite the desired direction of motion; and the other at the current bump location. This results in the current bump being effectively 'pushed away' from the negative bump. For values less than one, a proportionally scaled bump is created in the location as if $C(t) = 1$, and a proportionally scaled bump is subtracted from the current position, resulting in proportionally scaled movement. There are two reasons that this equation works as expected. The first is that the movements are very small, so the resulting bumps in all cases are approximately Gaussian (though subtly bi-modal). The second is that the attractor dynamics built into the network 'clean-up' any non-Gaussianity of the resulting states. The result is a network that displays bumps moving in either direction proportional to $C(t)$, as desired.

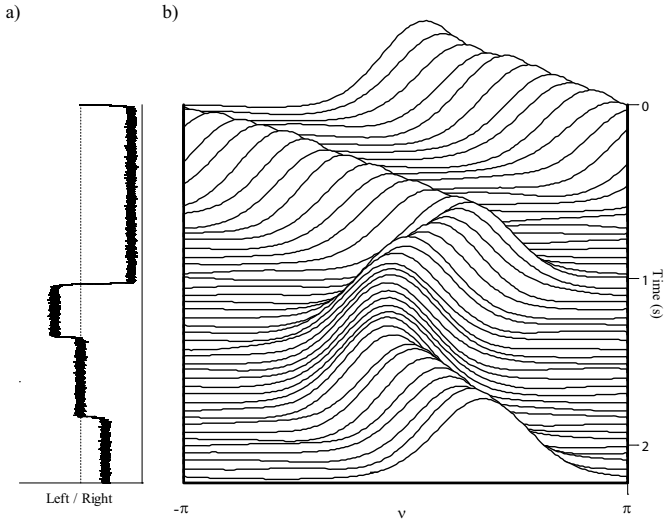

Figure 13: The controlled ring attractor in a spiking network of 1000 neurons with 10% noise. The left/right velocity control signal is shown on the left and the corresponding behavior of the bump is shown on the right.

spiking version of this model is shown in figure 13.

In this case, the control parameters simply move the system through the attractor's phase space, rather than altering the phase space itself as in the previous example. However, to accomplish the same movement using the input $u(t)$, we would have to provide the appropriate high-dimensional vector input (i.e., the new bump position minus the old bump position). Using the velocity commands in this nonlinear control system, we need only provide a scalar input to appropriately update the system's state variables. In other words, the introduction of this kind of control greatly simplifies updating the system's current position in phase space.

4.3 Controlling the speed of a cyclic attractor

As mentioned in section 3.3, one advantage of our synthesis of top-down and bottom-up data is that it permits the inclusion of strong top-down constraints on model building. In

the context of lamprey locomotion, introducing control over swimming speed and guaranteeing stable oscillation to CPG-based models were problems that had to be tackled separately, took much extra work, and resulted in solutions specific to this kind of network (Marder et al. 1997). In contrast, stability, control, and other top-down constraints can be included in the cyclic attractor model directly.

In this example, I consider control over swimming speed. Given the two previous examples, we know that this kind of control can be characterized as the introduction of non-linear or time-varying parameters into our state equations. For instance, we can make the frequency term ω in (23) a function of time. For simplicity, I will consider the standard oscillator, although it is closely related to the swimming model as discussed earlier (see Kuo and Eliasmith ress for an anatomically and physiologically plausible model of zebrafish swimming with speed control).

To change the speed of an oscillator we need to implement:

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & \omega(t) \\ -\omega(t) & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \mathbf{B}\mathbf{u}(t).$$

For this example, we can construct a nonlinear model, but unlike equation (27), we do not have to increase the dimension of the state-space. Instead, we can increase the dimension of the input space, so the third dimension carries the time-varying frequency signal, giving

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & u_3(t) \\ -u_3(t) & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \mathbf{B}\mathbf{u}(t). \tag{29}$$

As before, this can be translated into a neural dynamics matrix using (7) and implemented in a neural population using the methods analogous to those in section 4.1. In fact, the architecture used for the neural implementation is the same despite the al-

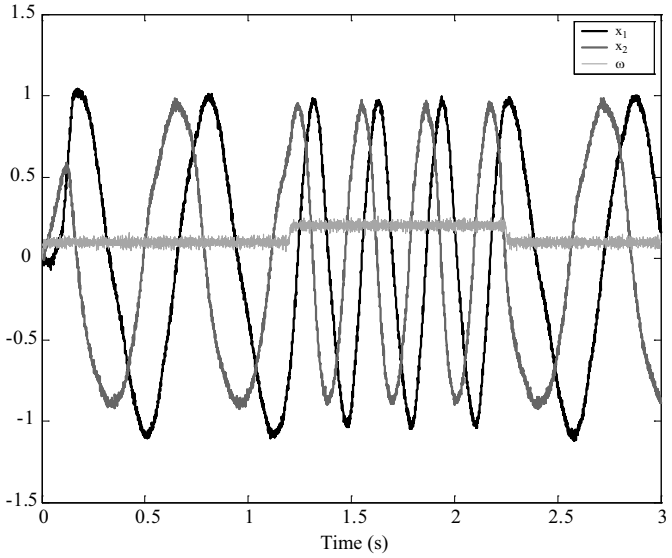

Figure 14: The controlled oscillator in a spiking network of 800 neurons with 10% noise. The control signal varies ω, causing the oscillator to double its speed and then slow down to the original speed.

ternate way of expressing the control system in (29). That is, in order to perform the necessary multiplication, the dimensionality of the space encoded by the population is increased by one. In this case, the extra dimension is assigned to the input vector rather than the state vector. It should not be surprising, then, that we can successfully implement a controlled cyclic attractor (see figure 14).

In this example, control is used to vary a property of the attractor, namely the period of the orbit. Because the previous two examples are static attractors, this kind of control does not apply to them. However, it should be clear that this example adds nothing new theoretically. However, it helps to demonstrate that the methods introduced earlier apply broadly. Additionally, the introduction of this kind of control into a neural model of swimming results in connectivity that matches the known anatomy (Kuo and Eliasmith ress).

4.4 Moving between chaotic, cyclic, and point attractors

Both Skarda and Freeman (1987) and Kelso and Fuchs (1995) have suggested that being in a chaotic attractor may help to improve the speed of response of a neural system to various perturbations. In other words, they suggest that if chaotic dynamics are to be useful to a neural system it must be possible to move into and out of the chaotic regime. Conveniently, the bifurcations and attractor structures in the Lorenz equations (24) are well-characterized, making them ideal for introducing the kind of control needed to enter and exit the chaotic attractor.

For instance, changing the b parameter over the range $[1, 300]$ causes the system to exhibit point, chaotic and cyclic attractors. So, constructing a neural control system with an architecture analogous to that of the controlled integrator discussed earlier would allow us to move the system between these kinds of states.

However, an implementational difficulty arises. As suggested by figure 10, the mean of the $x_3$variable is roughly equal to b. Thus, for largely varying values of b, the neural system will have to represent a large range of values, necessitating a very wide dynamic range with a good signal to noise ratio. This can be achieved with enough neurons, but it is more efficient to re-write the Lorenz equations to preserve the dynamics, but eliminate the scaling problem. To do so, we can simply subtract and add b as appropriate to remove the scaling effect. This gives

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{array}{l} a(x_2 - x_1) \\ bx_1 - x_2 - x_1(x_3 + b) \\ x_1 x_2 - c(x_3 + b) - b \end{array}$$

$$= \begin{bmatrix} -a & a & 0 \\ 0 & -1 & -x_1 \\ x_2 & 0 & -c \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -(c+1) & 0 & 0 \end{bmatrix} \begin{bmatrix} b \\ 0 \\ 0 \end{bmatrix}$$

Given this characterization of the Lorenz system, it is evident that, conveniently, introduction of the controlled signal b no longer requires multiplication, making the problem simpler than the previous control examples. Implementing these equations in a spiking neural population can be done as in section 3.4.

The results of simulating this network under noise are shown in figure 15. After the startup transient, the network displays chaotic behavior, as with no control (see figure 10). However, in this case it is the value of the control signal $b(t)$ that forces the system to be in the chaotic regime. After six seconds, the control signal changes, moving the system to a stable limit cycle. At eight seconds, the control signal changes again, moving the system to a fixed point attractor. To verify that these different regimes are as described, data from each of the regimes was titrated as before. The noise titration during the chaotic regime verified the presence of the nonlinearity (p-value $<10^{-15}$; noise limit 39%). During the limit cycle and the point attractor regimes, noise titration did not detect any nonlinearity, as expected.

Interestingly, despite the highly nonlinear nature of the system itself, the kind of control that might be useful for information processing turns out to be quite simple. Unlike the previous examples, this one demonstrates how non-multiplicative control can be very powerful. It serves to move a non-linear system through very different attractor regimes – some linear and some not. However, unlike previous examples, it is unclear how useful this kind of network is for better understanding neural systems. Nevertheless, it serves to illustrate the generality of the NEF for understanding a wide

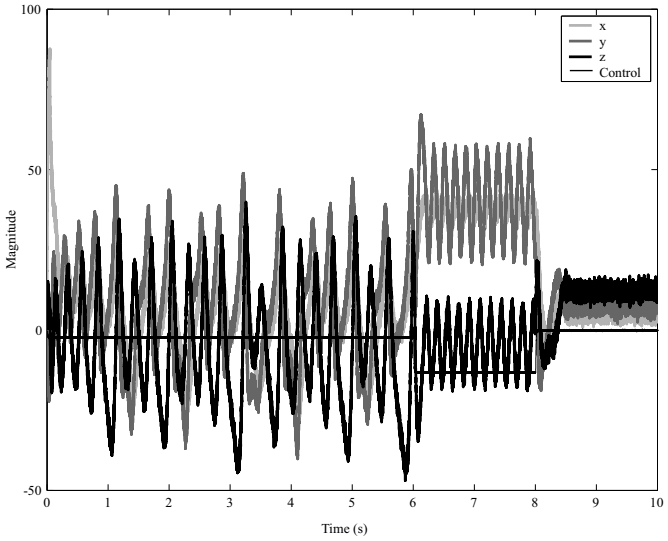

Figure 15: The controlled chaotic attractor in a spiking network of 1000 neurons. The system clearly moves from the chaotic regime to an oscillatory one, and finally to a point attractor. Noise titration verifies this result (see text for discussion).

variety of attractors and control systems in the context of constraints specific to neural systems.

5 Conclusion

I have presented several examples of biologically plausible attractor networks that cover a wide variety of attractors. These examples employ a variety of representations, including scalars, vectors and functions. And, they also exemplify a variety of control systems, including linear, time-varying and non-linear. As mentioned during the discussion, some of the examples on their own are not particularly novel. However, what is novel is demonstrating how they relate to more complex (and novel) models. In other words, it is this wide diversity of examples that shows that the NEF can help us systematize the functionality of neural systems. That is, despite differences in tuning curves,

kinds of representation, neural response properties, intrinsic dynamics, and so on, it is possible to classify various networks as variations on themes of integration, filtering, oscillation, etc. all of which can be derived from simple, general principles. Furthermore, characterizing high-level dynamics, and how lower-level properties affect those dynamics, can provide a window into the purpose of neural subsystems. Compiling this knowledge should aid in more quickly understanding the likely functions of larger, more complex, and less familiar systems.

This kind of systematization can be useful in a number of ways. For one, it suggests that perhaps the kinds of networks I have described here can serve as functional parts of larger networks – networks which we can construct using these same methods. One example of this is presented in Eliasmith et al. (2002), where an integrator is one of nine subnetworks used to estimate the true translational velocity of an animal given the responses of semicircular canals and otoliths to a variety of acceleration profiles. So, while I have used the NEF here to construct a specific class of networks, the same methods are more broadly applicable.

A second benefit of systematization, as mentioned earlier, is that it supports the transfer of knowledge and analyses regarding well-understood neural systems to lesser-understood ones. So, for instance, understanding a useful control structure for the ring attractor suggests a useful control structure for path integration, a lesser-studied and more complex neural system (Conklin and Eliasmith 2004). More generally, if we can see the close relation between two high-level descriptions of different neural systems (e.g., working memory and the neural integrator, or path integration and the head-direction system), what we have learned about one may often be 'translated' into implications for the other. This can greatly speed up the development of novel models and focus our attention on the important characteristics of new systems (be they

similarities to, or differences from, other known systems).

Finally, by being able to provide the high-level characterizations of neural systems upon which such systematization depends, we can carefully introduce new complexities into existing models. For instance, the recent surge of interest in the observed dynamics of working memory (Miller et al. 2003; Romo et al. 1999; Brody et al. 2003), can be captured by simple extensions of the models described earlier (Singh and Eliasmith 2004). Again, this can greatly aid the construction of novel models – models which may be able to address more complicated phenomena than otherwise possible (Eliasmith et al. (2004) presents a neuron-level model of a well-studied deductive inference task (the Wason card selection task) with 14 subsystems).

So, while this discussion has focused on characterizing a class of networks that is clearly important for understanding neural systems, the methods underlying this approach have much broader application. That is, they can help us begin to better understand the general control and routing of information through the brain in a way responsible to neural constraints. With continuing improvement in experimental techniques for examining large-scale networks, theoretical tools for understanding networks on a the same scale become essential. I would like to suggest that the examples provided here hint that the NEF may be a useful attempt at beginning to develop such tools.

Acknowledgments

I would like to express special thanks to John Conklin, Charles H. Anderson and Valentin Zhigulin and for helpful discussions. I would like to further thank John Conklin for generating the figures related to the ring attractor. The noise titration code for detecting chaos was graciously provided by Chi-Sang Poon. This work is supported

by grants from the National Science and Engineering Research Council of Canada, the Canadian Foundation for Innovation, the Ontario Innovation Trust and the McDonnell Project in Philosophy and the Neurosciences.

Appendix A

Consider determining the optimal linear decoders ϕ_i in (3) under noise (see also Salinas and Abbott 1994). To include noise, we introduce the noise term η_i which is drawn from a Gaussian, independent, identically distributed, zero mean distribution. The noise is added to the neuron activity a_i resulting in a decoding of

$$\hat{\mathbf{x}} = \sum_{i=1}^{N} \left(a_i(\mathbf{x}) + \eta_i\right) \phi_i. \tag{30}$$

To find the least squares optimal ϕ_i, we construct and minimize the mean square error, averaging over the expected noise and the vector $\mathbf{x}$:

$$\begin{aligned} E &= \frac{1}{2} \left\langle \left[\mathbf{x} - \sum_{i=1}^{N} \left(a_i(\mathbf{x}) + \eta_i\right) \phi_i \right]^2 \right\rangle_{\mathbf{x},\eta} \\ &= \frac{1}{2} \left\langle \left[\mathbf{x} - \left(\sum_{i=1}^{N} a_i(\mathbf{x}) \phi_i - \sum_{i=1}^{N} \eta_i \phi_i \right) \right]^2 \right\rangle_{\mathbf{x},\eta} \end{aligned} \tag{31}$$

where $\langle \cdot \rangle_{\mathbf{x}}$indicates integration over the range of $\mathbf{x}$. Because the noise is independent on each neuron, the noise averages out except when $i = j$. So, the average of the $\eta_i \eta_j$ noise is equal to the variance σ^2 of the noise on the neurons. Thus, the error with noise becomes

$$E = \frac{1}{2} \left\langle \left[\mathbf{x} - \sum_{i=1}^{N} a_i(\mathbf{x}) \phi_i \right]^2 \right\rangle_{\mathbf{x}} + \frac{1}{2} \sigma^2 \sum_{i=1}^{N} \phi_i^2. \tag{32}$$

Taking the derivative of the error gives

$$\begin{aligned}\frac{\delta E}{\delta \phi_i} &= -\frac{1}{2}\left\langle 2\left[\mathbf{x} - \sum_j^N a_j(\mathbf{x})\phi_j\right] a_i(\mathbf{x})\right\rangle_{\mathbf{x}} + \sigma^2\phi_j\delta_{ij} \\ &= -\left\langle a_i(\mathbf{x})\mathbf{x}\right\rangle_{\mathbf{x}} + \left\langle \sum_j^N a_i(\mathbf{x})a_j(\mathbf{x})\phi_j\right\rangle_{\mathbf{x}} + \sigma^2\phi_j\delta_{ij}. \end{aligned} \tag{33}$$

Setting the derivative to zero gives

$$\left\langle a_i(\mathbf{x})\mathbf{x}\right\rangle_{\mathbf{x}} = \sum_j^N \left(\left\langle a_i(\mathbf{x})a_j(\mathbf{x})\right\rangle_{\mathbf{x}} + \sigma^2\delta_{ij}\right)\phi_j \tag{34}$$

or, in matrix form,

$$\Upsilon = \Gamma\phi.$$

The decoding vectors $\phi_i$are given by

$$\phi = \Gamma^{-1}\Upsilon$$

where

$$\begin{aligned}\Gamma_{ij} &= \left\langle a_i(\mathbf{x})a_j(\mathbf{x})\right\rangle_{\mathbf{x}} + \sigma^2\delta_{ij} \\ \Upsilon_i &= \left\langle \mathbf{x}a_i(\mathbf{x})\right\rangle_{\mathbf{x}}.\end{aligned}$$

Notice that the Γ matrix will be non-singular because of the noise term on the diagonal.

Much of the power of this framework comes from the simple observation that this same procedure can be followed to find the optimal linear decoders $\phi^{f(\mathbf{x})}$ for some

arbitrary function $f(\mathbf{x})$. The error to be minimized then becomes

$$E = \frac{1}{2} \left\langle \left[f(\mathbf{x}) - \sum_{i=1}^{N} \left(a_i(\mathbf{x}) + \eta_i \right) \phi_i^{f(\mathbf{x})} \right]^2 \right\rangle_{\mathbf{x},\eta} .$$

The minimization is analogous, with only a differing result for Υ:

$$\Upsilon_i = \left\langle f(\mathbf{x}) a_i(\mathbf{x}) \right\rangle_{\mathbf{x}} .$$

This extension provides the impetus for analyses of the functions that are possible to compute given a specific neural population (i.e., set of $a_i(\mathbf{x})$ curves). While not reproduced here, these analyses demonstrate that polynomial nonlinearities are well-supported by neurons with broad tuning curves (Eliasmith and Anderson 2003, chp. 7). Thus we can be assured that computing products, as done in the networks presented here, can be done with sufficient accuracy. It is this ability that makes the inclusion of both nonlinear and time-varying control systems within this framework a possibility.

References

Amit, D. J. (1989). *Modeling brain function: The world of attractor neural networks*. New York, NY: Cambridge University Press.

Askay, E., G. Gamkrelidze, H. S. Seung, R. Baker, and D. Tank (2001). In vivo intracellular recording and perturbation of persistent activity in a neural integrator. *Nature Neuroscience 4*, 184–193.

Biswal, B. and C. Dasgupta (2002). Neural network model for apparent deterministic chaos in spontaneously bursting hippocampal slices. *Physical Review Letters 88*,

088102.

Bradley, E. (1995). Autonomous exploration and control of chaotic systems. *Cybernetics and Systems 26*, 299–319.

Brody, C. D., R. Romo, and A. Kepecs (2003). Basic mechanisms for graded persistent activity: discrete attractors, continuous attractors, and dynamic representations. *Current Opinion in Neurobiology 13*, 204–211.

Conklin, J. and C. Eliasmith (2004). A controlled attractor network model of path integration in the rat. *unpublished*.

Eliasmith, C. and C. H. Anderson (2000). Rethinking central pattern generators: A general approach. *Neurocomputing 32*, 735–740.

Eliasmith, C. and C. H. Anderson (2003). *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. Cambridge, MA: MIT Press.

Eliasmith, C., J. Conklin, and R. Singh (2004, March 24–28). A neurobiological simulation of deductive reasoning. In *Computational and Systems Neuroscience 2004*, Cold Spring Harbour, NY.

Eliasmith, C., M. B. Westover, and C. H. Anderson (2002). A general framework for neurobiological modeling: An application to the vestibular system. *Neurocomputing 46*, 1071–1076.

Fukushima, K., C. R. S. Kaneko, and A. F. Fuchs (1992). The neuronal substrate of integration in the oculomotor system. *Progress in Neurobiology 39*, 609–639.

Fuster, J. M. (2001). The prefrontal cortex – an update: time is of the essence. *Neuron 30*, 319–333.

Goodridge, J. P. and D. S. Touretzky (2000). Modeling attractor deformation in the rodent head-direction system. *Journal of Neurophysiology 83*, 3402–3410.

Grillner, S., P. Wallén, L. Brodin, and A. Lansner (1991). The neuronal network generating locomotor behavior in lamprey: Circuitry, transmitters, membrane properties, and simulation. *Annual Review of Neuroscience 14*, 169–199.

Hansel, D. and H. Sompolinsky (1998). Modeling feature selectivity in local cortical circuits. In C. Koch and I. Segev (Eds.), *Methods in neuronal modeling*. Cambridge, MA: MIT Press.

Hebb, D. O. (1949). *The organization of behavior*. New York, NY: Wiley.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences 79*, 2554–2558.

Kelso, J. A. S. and A. Fuchs (1995). Self-organizing dynamics of the human brain: Critical instabilities and silnikov chaos. *Chaos 5*, 64–69.

Koch, C. and T. Poggio (1992). Multiplying with synapses and neurons. In T. McKenna, J. Davis, and S. F. Zornetzer (Eds.), *Single neuron computation*. Boston, MA: Academic Press.

Kopell, N. and G. B. Ermentrout (1988). Coupled oscillators and the design of central pattern generators. *Mathematical Biosciences 90*, 87–109.

Kuo, D. and C. Eliasmith (in press). Understanding interactions between networks controlling distinct behaviors: Escape and swimming in larval zebrafish. *Neurocomputing*.

Lai, Y.-C., M. A. Harrison, M. G. Frei, , and I. Osorio (2003). Inability of lyapunov

exponents to predict epileptic seizures. *Physical Review Letters 91*, 068012.

Laing, C. R. and C. C. Chow (2001). Stationary bumps in networks of spiking neurons. *Neural Computation 13*, 1473–1494.

Manira, A. E., J. Tegnér, and S. Grillner (1994). Calcium-dependent potassium channels play a critical role for burst termination in the locomotor network in lamprey. *Journal of Neurophysiology 4*, 1852–1861.

Marder, E., N. Kopell, and K. Sigvardt (1997). How computation aids in understanding biological networks. In P. Stein, S. Grillner, A. Selverston, and D. Stuart (Eds.), *Neurons, networks, and motor behavior*. Cambridge, MA: MIT Press.

Matsugu, M., J. Duffin, and C. Poon (1998). Entrainment, instability, quasi-periodicity, and chaos in a compound neural oscillator. *Journal of Computational Neuroscience 5*, 35–51.

Mel, B. (1994). Information processing in dendritic trees. *Neural Computation 6*, 1031–1085.

Mel, B. (1999). Why have dendrites? a computational perspective. In G. Stuart, N. Spruston, and M. Häusser (Eds.), *Dendrites*. New York, NY: Oxford University Press.

Miller, P., C. Brody, R. Romo, and X.-J. Wang (2003). A recurrent network model of somatosensory parametric working memory in the prefrontal cortex. *Cerebral Cortex 13*, 1208–1218.

Platt, M. L. and G. W. Glimcher (1998). Response fields of intraparietal neurons quantified with multiple saccadic targets. *Experimental Brain Research 121*, 65–75.

Poon, C.-S. and M. Barahona (2001). Titration of chaos with added noise. *Proceedings of the National Academy of Science 98*, 7107–7112.

Pouget, A., K. Zhang, S. Deneve, and P. E. Latham (1998). Statistically efficient estimation using population coding. *Neural Computation 10*, 373–401.

Rainer, G. and E. K. Miller (2002). Timecourse of object-related neural activity in the primate prefrontal cortex during a short-term memory task. *European Journal of Neuroscience 15*, 1244–1254.

Redish, A. D. (1999). *Beyond the cognitive map*. Cambridge, MA: MIT Press.

Rieke, F., D. Warland, R. R. de Ruyter van Steveninick, and W. Bialek (1997). *Spikes: Exploring the neural code*. Cambridge, MA: MIT Press.

Romo, R., C. D. Brody, A. Hernández, and L. Lemus (1999). Neuronal correlates of parametric working memory in the prefrontal cortex. *Nature 399*, 470–473.

Salinas, E. and L. F. Abbott (1994). Vector reconstruction from firing rates. *Journal of Computational Neuroscience 1*, 89–107.

Salinas, E. and L. F. Abbott (1996). A model of multiplicative neural responses in parietal cortex. *Proceedings of the National Academy of Sciences USA 93*, 11956–11961.

Selverston, A. I. (1980). Are central pattern generators understandable? *Behavioral and Brain Sciences 3*, 535–571.

Sereno, A. B. and J. H. R. Maunsell (1998). Shape selectivity in primate lateral intraparietal cortex. *Nature 395*, 500–503.

Seung, H. S. (1996). How the brain keeps the eyes still. *Proceedings of the National Academy of Sciences, USA 93*, 13339–13344.

Seung, H. S., D. Lee, B. Reis, and D. Tank (2000). Stability of the memory of eye position in a recurrent network of conductance-based model neurons. *Neuron 26*, 259–271.

Shadlen, N. N. and W. T. Newsome (2001). Neural basis of a perceptual decision in the parietal cortex (area lip) of the rhesus monkey. *Journal of Neurophysiology 86*, 1916–1936.

Singh, R. and C. Eliasmith (2004, March 24–28). A dynamic model of working memory in the pfc during a somatosensory discrimination task. In *Computational and Systems Neuroscience 2004*, Cold Spring Harbour, NY.

Skarda, C. A. and W. J. Freeman (1987). How brains make chaos in order to make sense of the world. *Behavioral and Brain Sciences 10*, 161–195.

Snyder, L. H., A. P. Batista, and R. A. Andersen (1997). Coding of intention in the posterior parietal cortex. *Nature 386*, 167–170.

Touretzky, D. S. and A. D. Redish (1996). Theory of rodent navigation based on interacting representations of space. *Hippocampus 6*, 247–270.

von der Heydt, R., E. Peterhans, and M. Dursteler (1991). Grating cells in monkey visual cortex: Coding texture? In B. Blum (Ed.), *Channels in the visual nervous system: Neurophysiology, psychophysics, and models*. London: Freund.

Zhang, K. (1996). Representation of spatial orientation by the intrinsic dynamics of the head-direction cell ensemble: A theory. *Journal of Neuroscience 16*, 2112–2126.