Dynamical Systems in Spiking Neuromorphic Hardware

by

Aaron Russell Voelker

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2019

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner	Adrienne Fairhall Professor, Department of Physiology and Biophysics, University of Washington
Supervisor	Chris Eliasmith Professor, Department of Philosophy and Department of Systems Design Engineering, University of Waterloo
Internal Member	Jeff Orchard Associate Professor, David R. Cheriton School of Computer Science, University of Waterloo
Internal Member	Ali Ghodsi Professor, Department of Statistics and Actuarial Science and David R. Cheriton School of Computer Science, University of Waterloo
Internal-External Member	Sue Ann Campbell Professor, Department of Applied Mathematics, University of Waterloo

Author's Declaration

Statement of Contributions

Yang Voelker provided the artwork for Figure 2.3 as well as the brain illustration at the end of this front matter (the winning T-shirt design for the 2018 Telluride Neuromorphic Workshop). Figure 5.2 was designed in collaboration with Ken E. Friedl. Derivations from Lemma 5.1.1, equation 5.38, and section 5.1.4, extend work in collaboration with my co-authors, Kwabena Boahen and Terrence C. Stewart, both of whom provided important steps in accounting for pulse-extended double-exponential synapses with time-constant mismatch. My advisor, Chris Eliasmith, helped author and edit several publications that have been adapted and extended throughout; references are included at the top of each applicable section.

Abstract

Dynamical systems are universal computers. They can perceive stimuli, remember, learn from feedback, plan sequences of actions, and coordinate complex behavioural responses. The Neural Engineering Framework (NEF) provides a general recipe to formulate models of such systems as coupled sets of nonlinear differential equations and compile them onto recurrently connected spiking neural networks – akin to a programming language for spiking models of computation. The Nengo software ecosystem supports the NEF and compiles such models onto neuromorphic hardware. In this thesis, we analyze the theory driving the success of the NEF, and expose several core principles underpinning its correctness, scalability, completeness, robustness, and extensibility. We also derive novel theoretical extensions to the framework that enable it to far more effectively leverage a wide variety of dynamics in digital hardware, and to exploit the device-level physics in analog hardware. At the same time, we propose a novel set of spiking algorithms that recruit an optimal nonlinear encoding of time, which we call the Delay Network (DN). Backpropagation across stacked layers of DNs dramatically outperforms stacked Long Short-Term Memory (LSTM) networks—a state-of-the-art deep recurrent architecture—in accuracy and training time, on a continuous-time memory task, and a chaotic time-series prediction benchmark. The basic component of this network is shown to function on state-of-the-art spiking neuromorphic hardware including Braindrop and Loihi. This implementation approaches the energy-efficiency of the human brain in the former case, and the precision of conventional computation in the latter case.

Acknowledgements

This thesis would not have been possible if not for my interactions with a few key people. Most notably, my advisor, Chris Eliasmith, who gave me the freedom to swim into the deep end, and the structure, inspiration, counsel, and courage, to ask difficult, but relevant questions. His active engagement and guidance has been instrumental throughout the development of this thesis. Next, Terry Stewart, who programmed the Nengo backend for Braindrop, has been an invaluable source of support when it comes to his technical prowess, breadth of knowledge, and intuition behind constructing functional spiking networks. In my earlier days, Trevor Bekolay, Travis DeWolf, Daniel Rasmussen, and Xuan Choo, all played critical roles as academic mentors, by facilitating my exploration of the Neural Engineering Framework and Nengo. Likewise, Eric Hunsberger, Jan Gosmann, Peter Duggins, and Andreas Stöckel, have each given feedback and provided foundational research that has deeply influenced my work.

Special mention is owed to Kwabena Boahen, who, throughout all of our interactions, has indescribably imparted invaluable streams of insight in his critical and uncompromising quest for answers. Many of his questions, contributions, and comments on papers, have inevitably influenced large portions of this thesis in a subtly-pervasive but welcome manner. This extends to memorable moments with all of his students, but especially Sam Fok, Alex Neckar, and Eric Kauderer-Abrams – it has been a privilege to take part in these collaborations.

Briefly speaking with Mike Davies, hearing his eloquent characterization of neuromorphic computing, and having his genuine support in getting Loihi into the hands of researchers, has all been integral to propelling this work. Wilten Nicola has provided a few influential communications on double-exponential synapses and FORCE learning, while Jamie Knight and Michael Hopkins have each provided important references involving SpiNNaker and high-dimensional sampling. The words of many others have shaped this thesis throughout.

At the institutional level, I would like to acknowledge the support that I've received from ABR (for Nengo and NengoLib development and experimentation with Loihi), Intel (for access to Loihi), Stanford's Brains in Silicon Lab (for access to and support with Braindrop), and the University of Manchester (for access to SpiNNaker). Likewise, my acceptance into the neuromorphic workshops of Telluride, Capo Caccia, Intel's Neuromorphic Research Community (INRC), and the Nengo Summer School, have all proven to be constructive experiences and motivational fuel. The Centre for Theoretical Neuroscience (CTN), Computational Neuroscience Research Group (CNRG), David R. Cheriton School of Computer Science, and the University of Waterloo, have all contributed to providing a supportive academic environment over the past 11 years.

The experiments throughout this thesis make extensive use of Python, NumPy, SciPy, Jupyter, Matplotlib, Seaborn, Pandas, Nengo, PyStorm, Hyperopt, Scitkit-learn, Keras, and TensorFlow – for this, we are incredibly grateful to its thousands of contributors. A repeated mention is in order for Trevor, who provided this modified LaTeX template and facilitated my contributions to the Nengo software ecosystem. Likewise, Daniel, Eric, and Jan, for their help with Nengo development, provisioning of useful code, and feedback.

Working backwards in time, I would also like to thank Mani Fazeli for his many letters of support and words of wisdom at the beginning of graduate school, as well as Carol Norman for her recurring words of encouragement. And the enriched education that I received from my teacher Jon Devlin, at Bell High School, was monumental in my decision to study computer science at the University of Waterloo.

At a personal level, Chris has been an outstanding role model for how to live a balanced and meaningful life, as an academic, parent, and human being, all in ways that I can only aspire towards. Jason Batten-Carew has been the source of many extended philosophical discussions centered on the nature of reality, perception, stories, games, and cycles, all of which have undoubtedly rooted their way into my most basic pursuits for meaning.

I also thank my mom and dad for exciting my curiosity, and enabling my studies, throughout my entire life. And finally I thank my wife, Yang, for providing much-needed perspective and support throughout it all.

Dedication

To Yang – the love of my life. And Gunter – for he is a dog.

Table of Contents

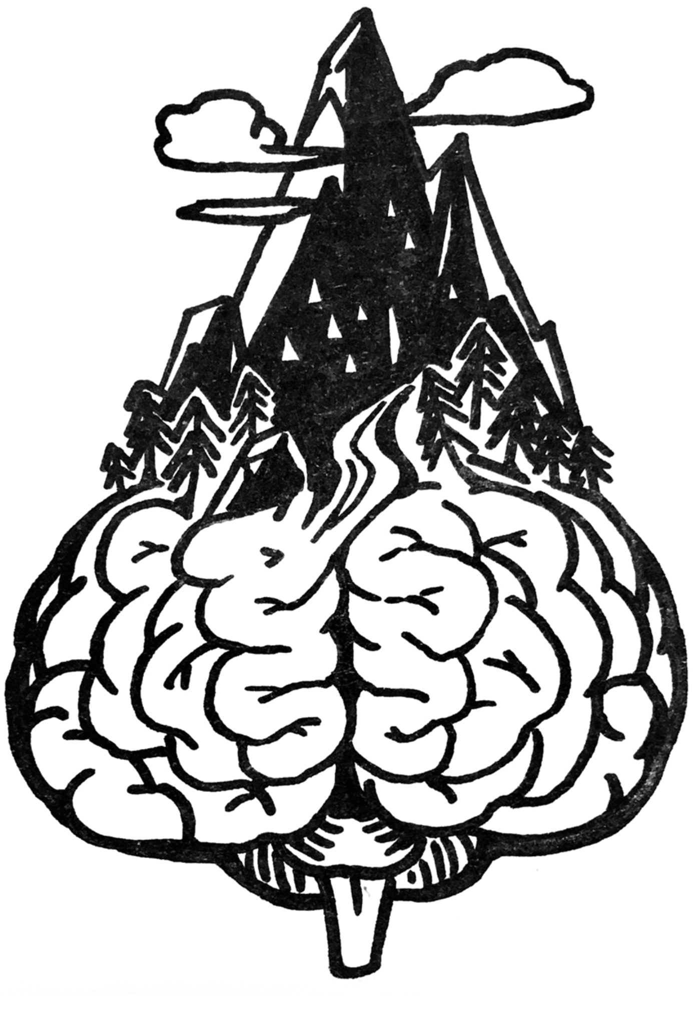
List of Figures

List of Tables

List of Theorems

Mathematical Conventions

This thesis follows standard mathematical conventions of using lower-case italic variables ($u(t)$) to represent scalar quantities, lower-case bold variables ($\mathbf{x}(t)$) to represent vector quantities, upper-case italic variables (A) to represent matrices, and Greek letters (τ, θ) to represent quantities with a unit of time. Bars are included above variables ($\bar{A}$) to indicate discretization in time. Depending on context, $F(s)$ or $F(z)$ are used to represent complex values in the Laplace s-domain, or digital z-domain, respectively. Big $\mathcal{O}(\cdot)$ notation is used informally to denote the growth rate of functions.



You would say to God, "How did you make the mountains?" and he would say... "Well, you're asking me to describe in words how I made the mountains, and there are no words which can do this. Words cannot tell you how I made the mountains any more than I can drink the ocean with a fork... It would take millions of years, and you would be bored with my description, long before I got through it... because I didn't create the mountains with words, I just did it. Like you open and close your hand. You know how you do this, but can you describe in words how you do it? ... You're conscious, aren't you. Don't you know how you manage to be conscious?" ~ Alan Watts

Chapter 1

Introduction

Computation cannot be physically realized without time. Whether we consider the discrete switching of voltages across transistors in a digital computer, or the continuous flow of charges across ion channels within the brain – all physical systems, that we currently know of, make progress on their computations by changing over time. Digital computers pipeline their computations through multiple cores with energetically-expensive access to distant memory, while our brains consume only twenty watts of power by harnessing the dynamics of a biophysical substrate. A leitmotif of this thesis is a computational paradigm in between these two extremes: neuromorphic computing. Here, researchers draw inspiration from how the brain dynamically performs computations to engineer low-power digital and analog circuits that emulate its fundamental principles.

The emerging field of neuromorphic computing has already witnessed many successes backed by large companies, small start-ups, and new research programs being established at various institutions all around the world. Applied Brain Research (ABR), Intel, IBM, and many others (QY Research, 2018), are establishing themselves as key players in the field, while a collaborative landscape is being fostered across a diverse set of academic communities attempting to use neuromorphic hardware, including the Telluride Neuromorphic Cognition Engineering Workshop (Cohen et al., 2001), Capo Caccia Cognitive Neuromorphic Engineering Workshop, Nengo Summer School (ABR's "Brain Camp"), and Intel's Neuromorphic Research Community (INRC).

Despite growing excitement, there exist many challenges when it comes to using neuromorphic hardware to perform some desired task. Generally speaking, neuromorphic programming forces us to rethink our traditional, Von Neumann (1958), understanding of computation, and redesign our algorithms to leverage distributed networks of noisy, heterogeneous units, that locally process their inputs and communicate via spikes – also known as Spiking Neural Networks (SNNs). These networks serve as powerful models of neural computation, capable in theory of everything that we can do, while consuming extraordinarily low amounts of energy. Biological nervous systems already leverage the computational power of SNNs quite successfully. Thus, our approach is to borrow existing principles and models from neuroscience, exploit their properties using tools from mathematics and engineering, and use software methods to synthesize the resulting networks *in silico*. Moreover, the frameworks and methods that we use to accomplish this should be extensible—able to incorporate increasingly-sophisticated levels of biological detail—so long as such details are computationally useful at a functional level, or shown to benefit some resource-power-accuracy trade-off.

To what end? By virtue of emulating core principles of brain function, we hope to realize algorithms that consume far less power, and scale more readily to massively-parallelized computing architectures running in real-time. But such algorithms must also be practically useful. To flip this on its head: we require frameworks that describe what some hardware is doing at the lowest level, so we may systematically harness its capabilities to perform useful computations at the highest level. Analogous to how a single compiler can translate the same C++ code to dozens of physically distinct digital architectures, we strive to systematically describe how the same dynamical computations can be flexibly mapped onto distinct spiking neuromorphic architectures.

The primary goal of this thesis is to explore the above characterization of neuromorphics. In doing so, we elucidate a general recipe for implementing efficient dynamical computations on neuromorphic hardware, empower theorists and practitioners with deeper insights into subtle relationships between network configuration and function, and demonstrate these methods on state-of-the-art neuromorphic chips: Braindrop (Neckar et al., 2019) and Loihi (Davies et al., 2018). Ultimately, this unveils a class of computations where neuromorphic hardware excels, and further automates the process of engineering algorithms for such hardware.

Researchers have already learned a great deal through the co-design of neuromorphic hardware and spiking algorithms. Theoretical frameworks help guide design decisions, while hardware constraints demand relevant extensions to theoretical frameworks and software tools; both processes give-and-take in a dynamic feedback loop. The general approach taken by this thesis adopts a software stack and nonlinear dynamical systems framework for describing computations within vector spaces. This approach is known as the Neural Engineering Framework (NEF; Eliasmith and Anderson, 2003). The general-purpose neural simulator, Nengo (Bekolay et al., 2014), natively supports NEF networks and a variety of neuromorphic backends. This software is extended by the methods described in this thesis. The ultimate goal of this approach is to develop a unified set of tools for constructing artificially intelligent agents, that scale to the same kinds of tasks that humans can perform, while running on neuromorphic hardware consuming several orders of magnitude less power than a traditional computer. Significant progress has been made in this direction over the past three decades. Although we still have a long ways to go, this thesis aims to highlight a promising path for neuromorphics.

When confronted with the ambitious goal of artificially reproducing human intelligence, computer scientists will often point towards the popularity of deep learning (LeCun et al., 2015) and its recent successes such as AlphaGo (Gibney, 2016). First and foremost, we seek to embrace the methods of deep learning, by incorporating its architectures and training methodologies into our toolbox. This is in fact the mandate of Nengo DL (Rasmussen, 2018), a Python package that combines Nengo and TensorFlow into a hybridized framework that automates backpropagation across spiking, non-spiking, and NEF-style recurrent neural networks – broadly referred to as the class of Artificial Neural Networks (ANNs). Deep learning methods have found enormous success, due in part to the black-box nature of the training procedure, which requires a programmer to have virtually no knowledge of the underlying problem being solved. However, we do not believe that deeper networks, better training heuristics, nor larger datasets, will be enough to achieve our aforementioned goal. Some concerns for conventional deep learning approaches include: the amount of power required to scale traditional ANNs to the size of the human brain (Furber, 2012), the difficulties tuning hyperparameters at scale (Bergstra et al., 2015), and the abundance of adversarial attacks on networks trained via backpropagation (Su et al., 2019). More generally, the "no free lunch theorem" informs us that a black-box learning algorithm, on its own, will never be enough (Wolpert, 1996); inevitably, it becomes necessary to incorporate *a priori* knowledge into the structure of ANNs in a manner that biases the solution space of the training procedure towards the problems being solved.

But even more fundamentally, the majority of ANNs in use today, including AlphaGo, are applied to *static* inputs to produce *static* outputs. For a board game such as Go, this is forgivable, as the entire state of the game is encapsulated by the current state of the board itself, and so there is technically no need to consider any history in order to optimize across future states.[1] Nevertheless, this points towards a thematic concern: time is not being treated as a continuous dimension of computation in many of the most successful methods, which may limit their applicability to many real-world problems. For instance, convolutional ANNs are most often deployed in video processing tasks by classifying objects in single frames, sampled at some discrete interval, and independently from one another. Recurrently connected networks are an exception that we embrace, but they come with many challenges that we will discuss. In general, most methods do not respect time as being a physical dimension that underpins the dynamics of the system's input, output, or processing units; in typical ANNs, time is not a variable in the neuron models, synapse models, nor any of the computational elements.

[1]Practically speaking, there could still be a benefit, for a sub-optimal player, to knowing the history of the game in order to better predict the opponent's strategy.

In contrast, the human brain is a physical system that continuously processes information over time. This is crucial for a cognitive agent embodied in a dynamic environment wherein all interactions depend on internally and externally evolving states. The brain naturally learns from interacting with its environment over many different time-scales. For instance, our brains are not programmed to recognize static objects at discrete moments in time. Rather, we have evolved to interact with objects in continuously changing environments, from which the perception of object recognition emerges. From a very young age, children can intuitively infer the physics of moving objects, and learn how to interact with those objects to achieve desired outcomes. This continues through adulthood, up to the level of socially coordinating our thoughts and behaviours, sculpted by our cumulative experience with the world. This occurs so naturally that we often take it for granted, but at a very basic level, our brains are necessarily constructing explicit and implicit dynamical models of reality, and applying these models to perform behaviourally relevant computations in real-time.

Consider a real-world example such as driving a car. We must constantly integrate new information with our existing understanding of where we are, where we are headed, and what might happen along the way. This requires not only an intuitive understanding of the physics of the car, but models of how other drivers behave on the road in the context of changing traffic lights and road conditions, all of which play out in parallel while we flexibly plan and coordinate our actions. Likewise, consider any actively engaging task or hobby that you enjoy. I speculate this activity requires some degree of mentally or physically coordinated interaction with the world, on a similar level of dynamic complexity as in the driving example, such as: playing a video game, participating in a sport, engaging with some form of media, playing music, drawing, dancing, and so forth. Besides such activities being meaningful, fun, and beautiful, they all share the common motif of recruiting a variety of systems in a dynamically coordinated fashion.

To help appreciate how extraordinary such tasks are for the human brain, we should recognize that the mechanisms involved do not have a perfect memory of past inputs. Each neuron locally processes some signal by continuously encoding it into spike trains. These action potentials evoke postsynaptic currents (PSCs) that decay exponentially on the order of milliseconds. Memories emerge from these dynamics, as changes to activation patterns, adaptive states, and synaptic weights, all reflect the histories of representations along a myriad of time-scales. The time-scales over which these mechanisms interface with the world, and interact with each other, determines, and constrains, all of our perceptions, thoughts, and actions. Remarkably, this is all accomplished by a collection of approximately one-hundred billion neurons, connected by over one-hundred trillion[2] synapses, processing these signals sparsely across space and time, while consuming only twenty watts (Koch, 2004) – about the same energy as a typical household CFL bulb.

This "signal processing view" of computation is fundamentally different from conventional views of computation targeted for conventional computers. In the former view, time is intrinsic to the state of all physical "devices" performing the computation (i.e., neurons and synapses), which are themselves necessarily tied to the timing and dynamics of the overall function as a consequence of actually implementing the function themselves. However, in the conventional view, the time that a computation takes is an implementation detail that is mostly decoupled from the input-output relation. As such, there is currently an important distinction between the computational approaches taken by the neuromorphic community and by those who program traditional digital computers. This is not simply a matter of biological detail, but more crucially a matter of thinking about the system that is actually solving these tasks as a collection of signal processors whose dynamics are coupled to the task at hand by the physics of time. If our aim is to build artificial systems that are as useful, robust, capable, and scalable as the human brain, then we should strive to understand computation in such terms.

The approach we take considers time as a fundamental starting point, and thus situates dynamics front-and-center; we bridge the time-scales of the entire system, down from the levels of spikes and postsynaptic currents, up to the levels of reaction times and behaviours. This involves developing an extensible theoretical framework that links together the mechanisms of an SNN, its parameters, and some desired class of computations. We validate this model of computation by deploying these SNNs on low-power neuromorphic hardware designed to emulate key biological principles. Based on these same principles, we develop theory connecting the mathematics of Padé and Legendre, to reveal a novel recurrent architecture that can predict chaotic time-series with greater memory capacity and performance surpassing state-of-the-art recurrent neural networks, including Long Short-Term Memory (LSTM), Reservoir Computing (RC), and "FORCE" networks.

[2]It is difficult to comprehend the sheer magnitude of this number; one-hundred trillion is more than the available estimates of the number of galaxies in the observable universe, trees on the earth, fish in the ocean, and digits of π that have been computed – all combined (at time of writing).

Chapter 2

Background

One of the earliest contributions to the neural network literature can be traced back to Rosenblatt's invention of the perceptron (Rosenblatt, 1958) based on the model of McCulloch and Pitts (1943). The perceptron is equivalent to a single-layer, feed-forward, neural network, where the input vector is multiplied through a weight matrix, and then projected through static nonlinear activation units (e.g., a threshold function, sigmoid function, rectified linear function, or similar).[1] This static nonlinearity is intended to correspond to the abstract behaviour (i.e., average activation level) of an artificial "neuron," while the matrix corresponds to a postsynaptic weighting of activations – hence the term "weight matrix." The output(s) of the network provide a classification of the input vector – the interpretation of which is task-dependent. The weight matrix is learned from example data, in a black-box manner, by applying a simple delta rule that, in effect, performs gradient descent on a loss function across the training data.

A great deal of excitement was fueled by claims that the perceptron would soon "be able to walk, talk, see, write, reproduce itself and be conscious of its existence" (Olazaran, 1996). But soon after, a famous book by Minsky and Papert (1969) pointed out a serious limitation of this architecture: it can only classify linearly-separable patterns – that is, it can only compute functions whose outputs distinguish its inputs based on whether the vector falls onto one side of a hyperplane or the other. The XOR function is perhaps the simplest example of a function that *cannot* be computed in such a manner. As a brief aside, many have credited this book as bringing upon the "AI winter" of the 1970s: neural network architectures were soon abandoned in favour of symbol-based architectures and so-called production systems, in which general aspects of human-like intelligence are reproduced by the manipulation of abstract symbolic representations (Olazaran, 1996; Newell et al., 1972).

[1] A constant bias term is usually added to the weights, to control the activation threshold of each unit.

Fortunately, additional layers permit the computation of more complicated nonlinear functions. That is, multi-layer feed-forward networks do not suffer the severe limitation of requiring linearly-separable output classes. By merely stacking two perceptrons together (i.e., introducing a "hidden layer"), the network becomes able to compute any bounded continuous function—a result known as the universal approximation theorem (Hornik et al., 1989)—assuming the nonlinear activation units are bounded and continuous themselves. This insight, together with the stacking of deeper layers to compose increasingly-complicated functions with fewer neurons, the use of "backpropagation" (Werbos, 1974; Rumelhart et al., 1986)—performing gradient descent across multiple layers via the chain rule—to learn all of the weight matrices, combined with increasingly-powerful GPUs to process large datasets, all coalesced to usher in the revival of AI in the new era of "deep learning" (Sejnowski, 2018).

Deep learning has undoubtedly brought about many rapid and impressive advances to the field of artificial intelligence. Due to its black-box nature, neither domain expertise nor understanding of the network's internal function are required in order to achieve state-of-the-art performance on a large number of important problems, including: image recognition, speech recognition, natural language understanding, question answering, and language translation (LeCun et al., 2015). The basic recipe is conceptually simple: obtain one of the many popular off-the-shelf toolboxes for deep learning, pick your favourite network architecture, set its hyperparameters, and then throw as much training data at the network as your GPU(s) can handle. Deep learning excels at constructing *static* vector functions, that generalize to new examples, by automatically discovering the latent (i.e., hidden) representations that are most relevant to the task at hand. However, the opacity of its optimization procedure comes as a double-edged sword: while it is easy to apply deep learning to many problems with minimal hand-engineering, it is unclear even to experts what effect most hyperparameter changes will have in advance on overall performance (Bergstra and Bengio, 2012). For example, it has been observed that using a rectified linear (ReLU) activation unit for each nonlinearity can dramatically reduce the training time on a variety of benchmarks (Glorot et al., 2011), yet, as far as we know, this observation still does not have any theoretically-grounded framework that could assist in selecting nonlinearities or related hyperparameters for some given task.

Despite its breakthroughs, the field is well-aware that a feed-forward architecture is incapable of learning relationships that span arbitrarily across the input data *in time*, which is necessary for tasks involving video, speech, and language understanding with long-range temporal dependencies (Bengio et al., 1994). Regardless of the depth of the network, a feed-forward network will always have some finite input response, which leaves a finite "memory" of previous inputs within the state of the network. In other words, the functions that are computable with such a network cannot access inputs that go beyond the depth of the network. The most general solution to overcome this problem is to introduce recurrent connections into the network, which transmit current state information back to itself, thus allowing the network to capture information about previous inputs and reuse it in the future. These networks are called Recurrent Neural Networks (RNNs).

We begin with a brief theoretical discussion on the computational power of RNNs, followed by an overview of many of the most popular architectural approaches and methods to building RNNs. This is not intended to provide a comprehensive record of all such findings, but rather to help situate the work in this thesis and shine a spotlight on many of the key ingredients that motivate our eventual approach. Reviews and references that provide more detailed accounts of these methods and its variants are included throughout. We then describe, in detail, the three principles of the Neural Engineering Framework (NEF; Eliasmith and Anderson, 2003) while contrasting this approach with those discussed thus far. This leads into a high-level overview of a few promising neuromorphic architectures that have been made available to us through various collaborations: SpiNNaker (Furber et al., 2014), Braindrop (Neckar et al., 2019), and Loihi (Davies et al., 2018) – all of which support the compilation of NEF networks, and are the targets of many of our example applications.

2.1 Recurrent Neural Networks

The Recurrent Neural Network (RNN) is the most computationally-powerful brand of neural network that we know how to physically implement. By using recurrent connections to persist state information through time, thus endowing the network with an internal memory, RNNs are able to compute functions outside the computational class afforded by deep feed-forward networks: *dynamical systems* – functions whose state evolves nonlinearly according to the history of its inputs. This enables the network to exploit patterns in the input that span time along arbitrary temporal scales.

Specifically, RNNs serve as a universal approximator to any finite-dimensional, causal, discrete-time, dynamical system (Schäfer and Zimmermann, 2006). That is,

$$\begin{aligned} \mathbf{x}[t+1] &= \mathbf{f}(\mathbf{x}[t], \mathbf{u}[t]) \\ \mathbf{y}[t] &= \mathbf{g}(\mathbf{x}[t]), \end{aligned} \tag{2.1}$$

where $\mathbf{x} \in \mathbb{R}^d$ is a finite-dimensional ($d \in \mathbb{N}$) state-vector, $\mathbf{u}$ is some input vector, and $\mathbf{y}$ is some output vector – all indexed at discrete units of time, t.[2] The state updates according to a nonlinear mixing of itself with the input. The output is a static mapping (i.e., "readout") of the

[2]There are similar theorems for continuous time (Funahashi and Nakamura, 1993; Bennett, 1995).

state. The causality constraint implies that these functions do not require knowledge of future inputs (although they are free to make predictions). Crucially, an RNN with sufficiently many nonlinear units can approximate any instantiation of equation 2.1 arbitrarily well, which in turn serves as a universal model of computation (Turing, 1938).

Likewise, the human brain itself is a highly recurrent system (Dayan and Abbott, 2001). Assuming its mechanisms obey the laws of physics, it must also necessarily be a dynamical system (Amit, 1989; McKenna et al., 1994; Port and Van Gelder, 1995). And unless the brain harnesses quantum physics—which seems to be irrelevant to cognitive function (Litt et al., 2006)—is a "hypercomputer" or a "super-Turing" computer—which seems to require unphysical resources (Broersma et al., 2018)—its dynamics must also be finite-dimensional. Causality follows from assuming the brain does not violate the second law of thermodynamics (Evans and Searles, 1996). In essence, we have good reason to believe that the brain is a physical "machine" with finite spatiotemporal resources and limited precision. From this position, it follows that the class of computations made available by RNNs is at least as large as the class of computations that can be performed by the brain. Constraints on the physical devices leveraged by the brain (i.e., constraints on timing, connectivity, precision, and so forth) are likely to impose limitations that make RNNs strictly more powerful in theory.

In summary, the field of neural network theory has progressed from the perceptron, which had the problem of not being very powerful, to the multi-layer feed-forward network, which is a universal approximator to static functions, and finally to the RNN, which is a universal approximator to physically-implementable dynamical systems, one example of which is the human brain. Thus, while feed-forward networks are fundamentally incapable of reproducing human intelligence, RNNs are, at least in theory, up to the task.

In practice, for tasks that involve sequential inputs, such as speech and language, RNNs are often the best choice (LeCun et al., 2015). Recent instances have demonstrated impressive human-like intelligence in generating captions from images (Vinyals et al., 2015), recognizing speech in multiple languages (Amodei et al., 2016), and decoding human emotions from video (Ebrahimi Kahou et al., 2015). However, the largest RNN that we are aware of to be successfully trained is a 6.6 million neuron model of human cognition—just shy of 0.01% the size of the human brain—using the methods of the NEF (Choo, 2018). The next largest (non-NEF) RNNs that we can find with published resource counts are "Google-scale" deep LSTM networks consisting of tens of thousands of units in their largest instantiations (Sak et al., 2014; Wu et al., 2016) – on the order of 10^5–10^7 times smaller than the human brain, depending on how one counts a "neuron" in such a network.

Clearly, there are practical and theoretical challenges in scaling. As informed by the "no free lunch theorem" for RNNs (Wiklicky, 1994), there exists no universal training algorithm that will find the correct solution to all tasks. It is instead necessary to constrain the solution space with prior knowledge. This has motivated the invention of many different types of RNNs and training methods, a selection of which are described in the following sections. For more information we point the reader to a review by Salehinejad et al. (2017).

2.1.1 Backpropagation through time

The traditional approach to build an RNN is to take a single-layer feed-forward network, and recurrently connect the output of each neuron to all other neurons in the same layer with a square weight matrix. To simulate the network, the weighted activations propagated by the recurrent feedback loop are delayed by a single time-step, and added to the weighted inputs on the next step. To train the network, one applies backpropagation through time (BPTT; Werbos, 1990). Intuitively, the idea is to "unroll" the network in time, and treat it as an infinitely-deep feed-forward network where the input at the i^{th} time-step, together with the output from the $(i-1)^{\text{th}}$ layer, are supplied to the i^{th} layer. Errors in the output of the RNN may then be backpropagated along the weights through time. This may optionally be stacked to form multiple layers, akin to a multi-layer feed-forward network where each layer is locally recurrent (Pascanu et al., 2013a).

This approach also extends to more biologically-plausible realizations of the RNN, in which the continuous (i.e., real-valued) neural activations are replaced by spiking neurons that encode signals into binary-valued (i.e., single-bit) events over time, and the weight matrices are replaced by synaptic filters that appropriately weight and exponentially decay incoming spikes over time. These networks are named Spiking Neural Networks (SNNs), and are the subject matter of later sections. The issue in this context is that spikes (i.e., Dirac impulses) are non-differentiable with respect to time. But recently, many research groups have independently demonstrated that SNNs can also be trained with backpropagation by softening the gradient in a variety of creative ways (Esser et al., 2015; Hunsberger and Eliasmith, 2015, 2016; Lee et al., 2016; Marblestone et al., 2016; Neftci et al., 2017; Bellec et al., 2018; Hunsberger, 2018; Huh and Sejnowski, 2018; Severa et al., 2018; Rasmussen, 2018; Shrestha and Orchard, 2018; Neftci et al., 2019)[3] although there are challenges in realizing biologically-plausible implementations that carry this out online (Hunsberger, 2018, submitted). Reviews that focus on feed-forward architectures are given by Pfeiffer and Pfeil (2018); Tavanaei et al. (2018), while examples applied to RNNs are demonstrated by Diehl et al. (2016); Huh and Sejnowski (2018); Kim et al. (2019). It was also recently shown by Chen et al. (2018b) that both RNNs and backpropagation

[3]Many of these are feed-forward examples, but recurrent spiking networks may be unrolled to support back-propagation in the same way.

may be extended to continuous-time (i.e., taking the limit of infinitesimally small time-steps), consistent with the the NEF and Principle 3 (see section 2.2.3).

While these approaches are theoretically sound, BPTT encounters many problems in practice (Bengio et al., 1994). Chief among these are the so-called "vanishing and exploding gradient problems," in which gradients shrink or grow unboundedly as they are propagated backwards in time. These problems are intimately related to the "credit-assignment problem," in which a learner witnesses a combinatorial explosion of points in time to assign credit to for the current error; in the case of storing a single bit of information over time, this causes the gradient to be inherently unstable (Bengio and Frasconi, 1994). This is further connected to the existence of nonlinear chaos in the dynamics of three (or more) recurrently coupled neurons; any correction to the trajectory of such networks are exponentially magnified over time (Strogatz, 2015). Thus, a standard RNN trained by BPTT will often fail to learn tasks that require a long-term memory – a serious problem for the use-cases that RNNs promise to address. Solutions proposed by Pascanu et al. (2013b) that involve clipping and constraining the gradients alleviate these problems to some extent, but the issue still appears in classical architectures for tasks that we know the brain is capable of solving (Lillicrap and Santoro, 2019). Alternatives to BPTT include extended Kalman filter-based learning, second-order Hessian-based optimization methods, Hessian-free optimization, and evolutionary algorithms (Salehinejad et al., 2017) – but none have been found to be as consistently useful in practice as backpropagation. To face the problem of learning across longer time-scales, architectural constraints are often imposed on the RNN, such as those in the following section.

2.1.2 Structured approaches

The Long Short-Term Memory (LSTM; Hochreiter and Schmidhuber, 1997) network addresses the problem of storing information across long intervals of time, by incorporating an explicit memory unit into the network. Rather than a static nonlinearity, the basic computational element is an LSTM "cell": a group of five nonlinearities and three multiplicative gates, dynamically coupled to one another in a specific manner, such that its default behaviour is to "store" its input, by propagating an internal state variable back to itself on the next time-step (i.e., using an internal feedback connection). Multiplicative gating controls whether the input should be integrated, whether the memory is to decay with some leakage, and whether to output the contents of the memory (Gers et al., 1999). Importantly, all of these mechanisms are differentiable, and so BPTT can be applied to automatically learn all of the gating parameters. Thus, the LSTM may be viewed as a particular way of configuring and initializing an RNN in a manner that assists BPTT, by biasing it towards latching onto values and persisting them for long, yet controlled, periods of time. This alleviates the vanishing and exploding gradient problems, described by Bengio et al. (1994), by restructuring the gradient that is involved when integrating over time.

The standard practice is to create layers of LSTM cells, and then stack several of these layers on top of one another, using weight matrices to fully connect each layer to the next (Graves et al., 2013). This enhances the temporal complexity of the network's dynamics through a composition of time-scales, analogous to the manner in which deep feed-forward networks provide richer internal representations via composition of spatial scales. Recently, this has found success in sequence learning, machine translation, and speech (Graves et al., 2013; Sutskever et al., 2014; Cho et al., 2014b; Bahdanau et al., 2014), leading to a widespread adoption of LSTM networks as the RNN of choice (LeCun et al., 2015).

A spiking analog to LSTMs have also been proposed by Bellec et al. (2018), in which adaptive neurons play the functional role of LSTM cells. The intuition behind why this works is that the adaptive state of the neuron is functionally similar to that of a controlled leaky integrator. When BPTT is applied to such an SNN, it has been shown to approach the performance of a non-spiking LSTM network. The structure of LSTM cells have also been compared to that of cortical microcircuits in biology, providing a *post hoc* explanation for the utility of LSTMs, and spiking implementations thereof (Costa et al., 2017; Pozzi et al., 2018).

This success has prompted many variants of LSTMs, including the bidirectional LSTM (Graves and Schmidhuber, 2005), hierarchical S-LSTM (Zhu et al., 2015), grid LSTM (Kalchbrenner et al., 2015), phased LSTM (Neil et al., 2016), and others (Salehinejad et al., 2017). At the same time, this has inspired several related architectures that augment an RNN with differentiable memory cells to facilitate long-range temporal interactions, including the gated recurrent unit (GRU; Cho et al., 2014a; Chung et al., 2014), neural Turing machine (Graves et al., 2014), memory network (Weston et al., 2014), and orthogonal GRU (Jing et al., 2018). The work of Jozefowicz et al. (2015) explored 10,000 related architectures and 230,000 hyperparameter configurations via evolutionary mutations to the LSTM's computational graph and found none to be consistently better than LSTMs.

In section 6.2.3, we propose a fundamentally new type of memory cell based on our work, the Delay Network (DN), and compare its performance in terms of memory capacity to LSTMs. We find that stacked LSTMs are unable to compress continuous-time windows of history. Our results on a chaotic time-series prediction benchmark suggest that LSTMs, when interleaved with the DN, and trained via BPTT, acquire an improved ability to learn dynamical systems described by nonlinear delay-differential equations.

2.1.3 Randomized approaches

Reservoir Computing (RC) is a strategy for training RNNs, that simplifies the optimization problem by keeping the recurrent weights fixed. That is, only the static "readout" weights to a linear output layer are modified, while the recurrent weights are left untrained, thus sidestepping the issues in BPTT. This is referred to as an Echo State Network (ESN; Jaeger, 2001), or Liquid State Machine (LSM; Maass et al., 2002), depending on whether the neural nonlinearities are static or spiking models, respectively. The intuition is that randomized feedback, when initialized in an appropriate manner, causes traces of the input to dynamically reverberate, or "echo," throughout the state of the network; a common metaphor is to compare the neural activity to throwing rocks into a pool of water and observing the ripples interact through time.[4] These traces can then be decoded to reconstruct the output signal in a manner that approximates a nonlinear transfer function of the input signal. Details are provided in section 2.2.4 after introducing the NEF.

There are two great insights driving the adoption of RC: (1) randomized feedback provides an overcomplete basis of nonlinear time-varying signals that can be linearly combined to approximate some target signal, and (2) by leaving the recurrent weights fixed, the optimization procedure is greatly simplified; training the readout weights amounts to a least-squares problem. However, the benefits of RC do not come without limitations involving memory capacity (Wallace et al., 2013). In particular, section 6.2.1 compares the performance of RC to the NEF on a continuous-time memory benchmark, thus highlighting severe scaling problems with RC, while demonstrating how the NEF solves these practical issues.

2.1.4 Engineered approaches

A unique set of approaches to training RNNs is rooted in the application of engineering and control-theoretic methods to modelling neurobiological systems (Eliasmith and Anderson, 1999). We call these top-down methods "engineered approaches". These methods all assume that some model of the system's desired dynamics (i.e., $\mathbf{x}$ in equation 2.1) are known *a priori*. That is, rather than suppose our only knowledge is raw input-output data, many real-world scenarios provide opportunities to model the dynamical system that actually generated the data. In practice, this might be some physical model of the dynamical system under consideration, an approximate model that has been identified (Nelles, 2013), or even some partial observation of the dynamical state. Often, the dynamical system is simply a description of a

[4]To demonstrate the relevance of this metaphor, Fernando and Sojakka (2003) went so far as to actually implement an LSM using a bucket of water.

particular computation that we know to be useful, such as a controller for a robotic arm, or the examples provided in section 4.2.

Instances of this approach include the Neural Engineering Framework (NEF; Eliasmith and Anderson, 2003; Duggins, 2017), first-order reduced and controlled error (FORCE; Sussillo and Abbott, 2009; DePasquale et al., 2018) and its spiking relatives (Thalmeier et al., 2016; DePasquale et al., 2016), balanced spiking networks (Boerlin and Denève, 2011; Boerlin et al., 2013; Schwemmer et al., 2015; Alemi et al., 2018), and closely-related methods (Jaeger, 2014; Gilra and Gerstner, 2017). Reviews of many of these methods are given by Denève and Machens (2016), Abbott et al. (2016), and Nicola and Clopath (2017). In all cases, these methods provide a procedure for taking a description of some desired dynamical state-vector, and then training a recurrent network of spiking or non-spiking neurons, with varying constraints on the target models' level of biological detail, to represent that state. We discuss some of the critical differences below.

It is important to preface our discussion with a reminder that, although all of these approaches are engineered from a top-down modelling perspective, one can still apply BPTT (or any other optimization method) to fine-tune the resulting RNN end-to-end from example input-output pairs – a hybrid approach that is embraced by the NEF, its software implementation Nengo, and related libraries (Rasmussen, 2018; Hazan et al., 2018). In this context, the purpose of these engineered approaches is as follows: incorporate prior knowledge of tasks to dramatically simplify the optimization procedure, provide guarantees regarding scalability and precision, and place appropriate constraints on the initial configuration of the RNN to assist in future rounds of optimization. At the same time, this provides greater transparency into the relationships between network parameters, configuration, and function. This transparency enables theoretical insights into neural computation, and, as we will show in chapter 6, practical advances on RNN benchmarks that would otherwise be quite difficult to obtain with a pure black-box approach.

In section 2.2 we provide a detailed account of the NEF, and in section 2.2.4 we draw architectural comparisons to both RC and FORCE. Its extension, full-FORCE (DePasquale et al., 2018), may be viewed as an instance of the NEF with full-rank weight matrices (Tripp and Eliasmith, 2006), where training is performed online rather than offline, and randomized feedback is thrown in to serve the same purposes as in RC. The crucial difference is that FORCE networks assert that recurrently connecting the estimate of the target variable back into the network during both training and inference will improve performance. The methods of Thalmeier et al. (2016) may be viewed as a means of realizing full-FORCE networks in biologically-plausible spiking networks using a precise spike-time coding mechanism, that is similar to Boerlin et al. (2013), but supported by nonlinearities in the synapses. Likewise, Jaeger (2014) may be viewed as an extension of RC that is essentially an offline variant of FORCE, or a non-spiking NEF network (Aubin, 2017). In section 6.2.2, we find that the NEF outperforms FORCE on a continuous-time memory benchmark, suggesting that FORCE learning may be insufficient for a large class of memory-based tasks.

Balanced spiking networks also share many commonalities with the NEF, primarily a procedure for mapping a dynamical state-vector onto a population of spiking neurons using a low-rank weight matrix that accounts for the dynamics of the neurons and synapses. The main benefit of the balanced network approach is that precision scales as $\mathcal{O}(n)$, compared to $\mathcal{O}\left(\sqrt{n}\right)$ for the NEF, where n is the number of neurons, by carefully coordinating the variance in spike-timing (Boahen, 2017, Figure 11). We discuss some important caveats in section 2.2.4.

Given the subtle distinctions between these overlapping approaches, section 3.2 rigorously analyzes the suitability of the NEF for universal spike-based computation on neuromorphic hardware. The NEF is mathematically correct under fairly weak assumptions, which leads to its scalability, completeness, robustness, and extensibility. These key factors contribute to the NEF being our tool of choice when it comes to training RNNs for the purposes of deployment on the neuromorphic hardware summarized in section 2.3.

2.2 Neural Engineering Framework

In this section, we provide a detailed overview of the Neural Engineering Framework (NEF; Eliasmith and Anderson, 2003) that has been adapted from Voelker and Eliasmith (2018).

The NEF proposes a general approach to model dynamical systems in artificial networks of spiking neurons. It does so by leveraging neural nonlinearities and weighted synaptic filters as computational resources. The NEF has been used to construct a wide variety of neural models, including a functioning 6.6 million neuron model of the human brain, dubbed "Spaun," that is capable of performing perceptual, motor, and cognitive tasks (Eliasmith et al., 2012; Choo, 2018). This model incorporates many kinds of observed neural dynamics, including self-generated oscillations, sustained activity, and point attractor dynamics.

As well, the flexibility of the NEF has lead to it being deployed on mixed-analog-digital neuromorphic chips (Choudhary et al., 2012; Corradi et al., 2014; Voelker et al., 2017a; Voelker and Eliasmith, 2017b; Neckar et al., 2019) and digital architectures (Bekolay et al., 2014; Wang et al., 2014; Mundy et al., 2015; Knight et al., 2016; Berzish et al., 2016; Wang et al., 2017; Blouw et al., 2018; Fischl et al., 2018). Consequently, the NEF provides a practical method for programming neuromorphics, thus helping the field deliver on its promise of a low-energy computing platform that emulates core principles of nervous systems (Boahen, 2017).

Biological brains possess a significant amount of structure, with neuroanatomical constraints that differ considerably between functional areas. We interpret this observation pragmatically, as a call to: (1) incorporate prior knowledge of a desired set of tasks into the network's construction, and (2) constrain a network's function using available models of the underlying biophysical mechanisms. However, the standard formulation of the NEF makes a number of simplifying assumptions regarding these mechanisms. For example, the NEF typically assumes an exponential model of the postsynaptic current (PSC) evoked by an action potential, which has a biologically-implausible, instantaneous, rise-time. This model is also known as a first-order lowpass filter. In contrast, the synapse models in mixed-analog-digital neuromorphic chips are non-ideal, featuring higher-order dynamics due to parasitic capacitances (Voelker et al., 2017a). Similarly, the synapse models commonly used in biological models incorporate distinct rise- and fall-times due to separate time-scales of transmitter binding and unbinding, as well as axonal transmission delays due to the finite-velocity propagation of action potentials (Roth and van Rossum, 2009). In digital hardware, spike-transmission delays are configurable (Davies et al., 2018) and necessary at the time-scale of the simulation time-step (Bekolay et al., 2014). Chapter 5 improves the scope of the NEF, by characterizing the network-level effects of these higher-order synapse models and delays, and harnesses them to implement both discrete-time and continuous-time dynamical systems with improved accuracy. Chapter 6 explores a novel theoretical system that equips any RNN—not just NEF-based—with an enhanced ability to compute nonlinear functions across windows of memory. We find some connections between this system's NEF implementation and the neural responses of "time cells" recorded in the rodent medial prefrontal cortex (mPFC). In this section, we provide the basic building blocks adapted from the original proposal by Eliasmith and Anderson (2003), before analyzing its theoretical merits in chapter 3.

In the context of RNNs, the NEF provides generic machinery for training the weights of a recurrent network of spiking (or non-spiking) neurons in order to implement some particular dynamical system (Voelker and Eliasmith, 2016). This is made efficient through three important steps: (1) weight matrices are factored into $n \times q$ *encoding* and *decoding* matrices, where $q \in \mathcal{O}(1)$ is the dimensionality of the dynamical state-vector, and n is the number of neurons, (2) the optimization problem is recast as a batched least-squares problem that may be solved offline without needing to explicitly simulate the network over time, and (3) the dynamics of the synapses are leveraged as temporal bases for the overall system. These insights lead to procedures for training and simulation that are efficiently implemented—scaling as $\mathcal{O}(n)$ in both time and space—by the Nengo software package (Bekolay et al., 2014). Backpropagation through time can fine-tune the performance of these networks, by use of the Nengo DL extension (Hunsberger, 2018; Rasmussen, 2018; Blouw et al., 2018), which bidirectionally integrates Nengo with Google's machine learning toolkit, TensorFlow (Abadi et al., 2016).

At the same time, this top-down approach provides considerable transparency into systematic relationships between neural nonlinearities, synapse models, time-constants, and network function. For instance, it becomes feasible to determine the class of functions that are most accurately supported by a population (i.e., layer) of neurons (Eliasmith and Anderson, 2003, pp. 185–217). Optimized architectures can therefore be analytically derived for the case of specific functions, such as multiplication (Gosmann, 2015). It also becomes viable to relate the time-constants of the low-level dynamical primitives (i.e., neurons and synapses) to the emergent dynamics of the network (Voelker and Eliasmith, 2018). Furthermore, by virtue of having functional spiking networks grounded in biologically-plausible mechanisms, it becomes possible to investigate connections between cognitive models of psychological phenomena, neural data, and the effects of drugs on behaviour and neurodegenerative disorders (Eliasmith, 2013; Eliasmith et al., 2016; Duggins et al., 2017).

Conceptually, the NEF consists of three mathematical principles used to describe neural computation: (1) representation, (2) transformation, and (3) dynamics. The NEF is most commonly applied to building dynamic (i.e., recurrent) spiking neural networks, but also applies to non-spiking and feed-forward networks. Its primary strengths reside in providing a well-understood and efficient means of engineering spiking neural models, and programming neuromorphic hardware, to approximate mathematically-described computations with precision that scales as $\mathcal{O}\left(\sqrt{n}\right)$ in the spiking case and $\mathcal{O}\left(n\right)$ in the non-spiking case (Eliasmith and Anderson, 2003; Boahen, 2017). We now provide an overview of these methods, applied to training both feed-forward and recurrent connection weights, with an emphasis on mapping linear dynamical systems onto SNNs – although these methods extend to nonlinear dynamical systems as well (Voelker et al., 2017a; Voelker and Eliasmith, 2017b).

2.2.1 Principle 1 – Representation

Let $\mathbf{x}(t) \in \mathbb{R}^q$ denote a q-dimensional continuous-time varying signal, that is to be represented by a population of n spiking neurons. To describe this representation, we define a nonlinear *encoding* and a linear *decoding* that together determine how neural activity relates to the represented vector.

First, we choose encoders $E = [\mathbf{e}_1, \ldots, \mathbf{e}_n]^\top \in \mathbb{R}^{n \times q}$, gains $\alpha_i > 0$, and biases β_i $(i = 1 \ldots n)$, as parameters for the following encoding:

$$\begin{aligned} s_i(t) &= \alpha_i \langle \mathbf{e}_i, \mathbf{x}(t) \rangle + \beta_i \\ a_i(t) &= G_i\left[s_i(t)\right] \\ &= \sum_m \delta\left(t - t_{i,m}\right), \end{aligned} \tag{2.2}$$

where $s_i(t)$ is the input current to the i^{th} neuron, $\langle \cdot, \cdot \rangle$ denotes a dot product, $a_i(t)$ is the neural activity generated by the i^{th} neuron encoding the vector $\mathbf{x}(t)$ at time t, $G_i[\cdot]$ is the nonlinear transfer function modelling the spike-generation of a single neuron, $\delta(\cdot)$ is the Dirac delta, and $\{t\}_{i,m}$ is the sequence of spike-times generated by the neuron model in response to the input current.

A common choice of neuron model for $G_i[\cdot]$ is the leaky integrate-and-fire (LIF) neuron model (Koch and Segev, 1998). This model maintains a one-dimensional voltage trace, $v_i(t)$, by continuously integrating a current-source $s_i(t)$ over time, with some leakage governed by the time-constant[5] τ_{RC} – functionally equivalent to a resistor-capacitor (RC) circuit, a first-order lowpass filter, or a convolution with an exponentially decaying impulse response. An action-potential, or "spike," is generated when the voltage reaches a threshold, in which event the voltage is reset to its resting potential, where it remains clamped for the duration of some refractory period, τ_{ref}. After normalizing the range of $v_i(t)$ to the interval $[0, 1]$, the subthreshold dynamics of the LIF neurons are described by the continuous-time dynamical system:

$$\tau_{\text{RC}} \dot{v}_i(t) + v_i(t) = s_i(t), \quad 0 \le v_i(t) < 1, \tag{2.3}$$

where $v_i(t) \rightarrow 1$ generates a spike at time t, followed by a reset to $0 \rightarrow v_i(t)$ that lasts τ_{ref} seconds. This model is considered to strike a convenient balance between simplicity and biological realism (Eliasmith and Anderson, 2003, pp. 81–82, 88–89), although the NEF does not eliminate the possibility of considering significantly more biologically-detailed neuron models (Duggins, 2017). Likewise, neurons driven by nonlinear conductances, rather than pure current-sources, can be leveraged by the NEF (Stöckel et al., 2018). Nevertheless, a modelling competition centered around fitting biological recordings from real pyramidal neurons, using artificial models of action-potential generation, discovered that simpler models (e.g., non-leaky integration with an adaptive threshold) are often the most successful at reproducing precisely-timed spike-trains (Gerstner and Naud, 2009).

The heterogeneous parameters that define the encoding, $\left\{ \left(\mathbf{e}_i, \alpha_i, \beta_i \right) : i = 1 \ldots n \right\}$, determine the variety of nonlinear responses from the population. Typically the length of each encoder, $\|\mathbf{e}_i\|_2$, is fixed to 1, while $\alpha_i > 0$ controls its length. Then $\mathbf{e}_i$ may be interpreted geometrically as a "preferred direction" vector, such that its dot product similarity with $\mathbf{x}(t)$ determines the relative magnitude of $s_i(t)$. The bias terms β_i effectively determine the sparsity of representation (i.e., which neurons are active for a given $\mathbf{x}(t)$), while the gain terms α_i determine the density of spikes relative to τ_{ref} (i.e., how many spikes are generated by an active neuron). These parameters can be randomly sampled from distributions constrained by the domain of $\mathbf{x}(t)$ and the dynamic range of $G_i[\cdot]$, fit from neuroanatomical data (e.g., known

[5]A time-constant is a physical quantity with a unit of time (e.g., seconds). It is *not* a dimensionless constant.

tuning curves, preferred directions, firing rates, sparsity, etc.; see for instance Friedl et al. (2016)), predetermined using prior knowledge of the desired transformation (Gosmann, 2015), or trained via backpropagation through time (Rasmussen, 2018). By default, we select these parameters such that the neural responses are uniformly distributed with respect to $\mathbf{x}$ across the q-dimensional hyperball (Gosmann, 2018, pp. 51–65). In essence, equation 2.2 defines a high-dimensional nonlinear projection of the vector $\mathbf{x}(t)$, by taking its dot product with an encoding matrix E, and injecting the result into n heterogeneous spike-generators.

Having defined an encoding, we introduce a postsynaptic filter, $h(t)$, which behaves as a model for the synapse by describing the dynamics of a receiving neuron's current. Specifically, this filter models the postsynaptic current (PSC) triggered by action potentials arriving at the synaptic cleft. For now, we set this to be an exponentially decaying PSC with time-constant τ:

$$h(t) = \frac{1}{\tau} e^{-\frac{t}{\tau}}, \quad t \geq 0. \tag{2.4}$$

This is equivalent (in the Laplace domain) to the canonical first-order lowpass filter (also known as a "leaky integrator"), with the same dynamics as equation 2.3. The lowpass synapse is the conventional choice of synapse in the NEF, since it is mathematically tractable while capturing a central feature of biological synapses and approximating the optimal *acausal* filter (Eliasmith and Anderson, 2003, pp. 109–114, 223). In section 5.1, we relax this requirement by considering more general synapse models that are capable of reproducing a much broader variety of PSCs.

We can now characterize the decoding of the neural response, which determines the information extracted from the neural activities encoding $\mathbf{x}(t)$. Let $D = [\mathbf{d}_1, \ldots, \mathbf{d}_n]^\top \in \mathbb{R}^{n \times q}$ be the decoding matrix used to decode a filtered version of $\mathbf{x}(t)$ from the population's activities, $a_i(t)$, at time t, as follows:

$$\begin{aligned} (\mathbf{x} * h)(t) &\approx \sum_{i=1}^{n} (a_i * h)(t) \mathbf{d}_i \\ &= \sum_{i=1}^{n} \sum_{m} h\left(t - t_{i,m}\right) \mathbf{d}_i, \end{aligned} \tag{2.5}$$

where $*$ is the convolution operator used to apply the synaptic filter. Equation 2.5 takes a linear combination of the filtered activities, in order to recover a filtered version of the encoded signal.[6] To complete the characterization of neural representation, we must map the encoders, decoders, and synapse models onto a neural architecture, and solve for the optimal linear decoders D. This optimization is identical for Principles 1 and 2, as discussed below.

[6]It is more accurate to apply the filter to both sides, since in general the (time-invariant) decoders alone cannot compensate for the filter applied to the activities – this instead becomes the responsibility of Principle 3.

2.2.2 Principle 2 – Transformation

The second principle of the NEF addresses the issue of computing transformations of the represented signal. The encoding remains defined by equation 2.2. However, we now decode a filtered version of some function, $\mathbf{f} : \mathbb{R}^q \to \mathbb{R}^q$,[7] by applying an alternate matrix of decoders to the same filtered spike trains:

$$\begin{aligned} D^{\mathbf{f}} &= \left[\mathbf{d}^{\mathbf{f}}_1, \ldots, \mathbf{d}^{\mathbf{f}}_n\right]^{\mathsf{T}} \in \mathbb{R}^{n \times q} \\ (\mathbf{f}(\mathbf{x}) * h)(t) &\approx \sum_{i=1}^{n} (a_i * h)(t)\mathbf{d}^{\mathbf{f}}_i \\ &= \sum_{i=1}^{n} \sum_m h\left(t - t_{i,m}\right)\mathbf{d}^{\mathbf{f}}_i. \end{aligned} \tag{2.6}$$

Now we must solve for $D^{\mathbf{f}}$. For both Principles 1 and 2, we optimize for $D^{\mathbf{f}}$ over the domain of the signal, $S = \{\mathbf{x}(t) : t \geq 0\}$, which is typically the unit q-ball $\{\mathbf{v} \in \mathbb{R}^q : \|\mathbf{v}\|_2 \leq 1\}$ or the unit q-cube $[-1,1]^q$. This can be done, as is, by simulating the population over time and solving a least-squares problem. But, in order to do this as efficiently and scalably as possible, we also reformulate the problem to avoid needing to *explicitly* simulate the spike-generation. We first let $r_i(\mathbf{v})$ be the limiting average firing rate of the i^{th} neuron under the constant input $\mathbf{v} \in S$:

$$r_i(\mathbf{v}) = \lim_{t \to \infty} \frac{1}{t} \int_0^t a_i(t')\, dt'. \tag{2.7}$$

For non-adaptive neuron models, equation 2.7 reduces to encoding $\mathbf{v}$ using a rate model. To provide an example, we solve for the closed-form solution in the case of the LIF model. To start, we solve the differential equation 2.3 while holding $s_i = \alpha_i \langle \mathbf{e}_i, \mathbf{v} \rangle + \beta_i$ fixed:

$$v_i(t) = v_i(0) + (s_i - v_i(0))\left(1 - e^{-\frac{t}{\tau_{\text{RC}}}}\right).$$

[7] We may also consider transformations where the range has a different dimension, but the described framework will suffice for our purposes.

To determine how much time it takes to generate a spike, we set $v_i(0) = 0$, $v_i(t) = 1$, and rearrange for t:

$$\begin{aligned} 1 &= 0 + (s_i - 0)\left(1 - e^{-\frac{t}{\tau_{\mathrm{RC}}}}\right) \\ \frac{1}{s_i} &= 1 - e^{-\frac{t}{\tau_{\mathrm{RC}}}} \\ \ln e^{-\frac{t}{\tau_{\mathrm{RC}}}} &= \ln\left(1 - \frac{1}{s_i}\right) \\ t &= -\tau_{\mathrm{RC}} \ln\left(1 - \frac{1}{s_i}\right). \end{aligned}$$

The total time to spike, including the refractory period, is $\tau_{\mathrm{ref}} + t$. And since frequency is the reciprocal of period, the expected spike-rate in response to s_i becomes:

$$r_i(\mathbf{v}) = \frac{1}{\tau_{\mathrm{ref}} - \tau_{\mathrm{RC}} \ln\left(1 - \frac{1}{s_i}\right)}. \tag{2.8}$$

Equation 2.8 is also known as the rate model for a LIF neuron, as it describes a static nonlinearity whose output is the LIF neuron's spike rate in response to the input.

For adaptive neuron models, other definitions for $r_i(\mathbf{v})$ may be considered, but we limit our discussion here to the (non-adaptive) spiking LIF model. To account for the variance introduced by neural spiking, and other sources of uncertainty, we introduce a white noise process $\eta \sim \mathcal{N}(0, \sigma^2)$. The solution to equation 2.6 becomes:

$$D^{\mathbf{f}} = \underset{D \in \mathbb{R}^{n \times q}}{\arg\min} \int_S \left\| \mathbf{f}(\mathbf{v}) - \sum_{i=1}^{n} \left(r_i(\mathbf{v}) + \eta\right) \mathbf{d}_i \right\|_2^2 d^q\mathbf{v}. \tag{2.9}$$

Note that this optimization only depends on $r_i(\mathbf{v})$ for $\mathbf{v} \in S$, as opposed to depending on the signal $\mathbf{x}(t)$. In other words, the optimization is determined strictly by the distribution of the signal, and not according to its particular dynamics. Since we have replaced the optimization problem involving spikes (equation 2.6) with one involving static nonlinearities, this sidesteps the need to explicitly simulate the neurons over time when training. Furthermore, this is a convex optimization problem, which may be solved by uniformly sampling S (Voelker et al., 2017b) and applying a standard regularized least-squares solver to the sampled data (Bekolay et al., 2014). In the context of machine learning, the samples of S are the example inputs from training data. Nengo also supports alternative decoder solvers that optimize variants of equation 2.9 (e.g., Friedl et al., 2016; Kauderer-Abrams et al., 2017), and randomized singular-value-decomposition (SVD) solvers that take $\mathcal{O}(mn)$ time and space (Halko et al., 2011) – but we do not use them here.

The accuracy of this approach depends on the quality of the following approximation:

$$\mathbf{x}(t) = \mathbf{v} \quad \implies \quad \sum_{i=1}^{n} (a_i * h)(t)\mathbf{d}_i^{\mathbf{f}} \approx \sum_{i=1}^{n} \Big(\big(r_i(\mathbf{v}) + \eta\big) * h\Big)(t)\mathbf{d}_i^{\mathbf{f}}. \tag{2.10}$$

In plain words, this presupposes that substituting a filtered spike-train with its correspondingly filtered (but noisy) rates does not change the optimal set of decoders for a target set of postsynaptic currents. A subtle, yet critical, observation is that we *do not* require the spike rates of *individual* neurons to reflect their idealized instantaneous rates. Rather, we exploit the fact that filtering and weighting spikes across the *entire* population will satisfy such an approximation (Eliasmith and Anderson, 2003, pp. 132–136).

In section 3.1.5, we discuss how our adoption of equation 2.10 has lead many to mischaracterize the NEF as using a "rate code," which comes with significant baggage. We suggest that the NEF should instead by characterized as mapping latent state-variables onto postsynaptic currents, and to label this as a "postsynaptic current code"; this code can exhibit repeatedly precise spike-times, facilitate rapid transmission of information, and does not demand excessive pool sizes or long time-constants, contrary to commonly-held beliefs that have carried over from rate-coding paradigms. In the NEF, individual neurons are free to spike much slower than their ideal rate models, while remaining silent across intervals of time that far exceed the time-constant of the synapse. In section 3.2.1, we provide a novel proof that justifies the correctness of this technique much more generally. Perhaps surprisingly, this optimization procedure is mathematically correct for Poisson-spiking models and simple integrate-and-fire models, even as the frequency of the state-vector goes towards infinity, and as the time-constants of the filters approach zero (see Figure 5.4).

It remains to map these encoding and decoding parameters onto a neural architecture. Equations 2.2 and 2.6 are used to derive a connection weight matrix between layers that implicitly computes the desired function $\mathbf{f}(\mathbf{x})$ within the *latent* state-space $\mathbb{R}^q$. Let ω_{ij} be the "weight," or coupling strength, between the j^{th} presynaptic neuron and the i^{th} postsynaptic neuron. Specifically, the weight matrix $W = [\omega_{ij}] \in \mathbb{R}^{n \times n}$, which maps activities from the j^{th} presynaptic neuron to the i^{th} postsynaptic neuron (disregarding the gain and bias, for notational simplicity), is given by:

$$W = E\left(D^{\mathbf{f}}\right)^{\top}. \tag{2.11}$$

To project activities from one population to another (or back to itself), they are decoded from an n-dimensional neuron space into the q-dimensional state-space, and then encoded back into the n-dimensional neuron space. Thus, by linearity, the process of decoding (equation 2.6) and then encoding (equation 2.2), is equivalent to taking the dot product of the weight matrix W with a vector of neural activations. Consequently, the matrices E and $D^{\mathbf{f}}$ are a low-rank factorization of W. Moreover, we remark that by linearity, the addition of multiple connections has the effect of adding together their transformations in state-space. Similarly, any linear transformation of the nonlinear function $T\mathbf{f}(\mathbf{x})$, $T \in \mathbb{R}^{q \times q}$, may be implemented by the decoders $D^{\mathbf{f}}T$ or by the encoders ET—both are optimal and equivalent to one another—without needing to solve for a new set of weights. Although this assumes linear synapses, the computations of nonlinear conductance-based synapses may be exploited by the NEF to encode nonlinear functions of the decoded variables across multiple pre-populations (Stöckel et al., 2018).

This factorization has additional consequences for the computational efficiency of neural simulations. The crucial difference between the factorized and non-factorized forms is that it takes $\mathcal{O}\left(qn\right)$ operations per simulation time-step to implement this dot product in the factored form of equation 2.11, as opposed to $\mathcal{O}\left(n^2\right)$ operations for a full weight matrix. Since q is typically held constant, this yields a factor $\mathcal{O}\left(n\right)$ improvement in simulation time. Similarly, this factorizations yields an $\mathcal{O}\left(n\right)$ reduction in memory, which significantly improves the scaling of neuromorphics (Mundy et al., 2015). In essence, this factorization provides a way to describe the network's latent state-vector $\mathbf{x}(t)$. This, in turn, enables useful computations that nonlinearly transform the state-vector with $\mathcal{O}\left(\tau\sqrt{n}\right)$ precision in the presence of spike-noise (see Eliasmith and Anderson (2003, p. 47–48) and section 3.2.2).

2.2.3 Principle 3 – Dynamics

Principle 3 is a method of harnessing the dynamics of the synapse model for network-level information processing. We begin by focusing our discussion of NEF dynamics on the neural implementation of continuous, linear time-invariant (LTI) systems (see Figure 2.1):

$$\begin{aligned}\dot{\mathbf{x}}(t) &= A\mathbf{x}(t) + B\mathbf{u}(t) \\ \mathbf{y}(t) &= C\mathbf{x}(t) + D\mathbf{u}(t),\end{aligned} \tag{2.12}$$

where the time-varying signal $\mathbf{x}(t)$ represents the system state, $\dot{\mathbf{x}}(t)$ its time-derivative, $\mathbf{y}(t)$ the output, $\mathbf{u}(t)$ the input, and the time-invariant matrices (A, B, C, D) fully describe the system (Brogan, 1991). This form of an LTI system is commonly referred to as the *state-space model*, but there are many other equivalent forms, which we refer to later.

For LTI systems, the *dynamical primitive*—that is, the source of the dynamics—is the integrator (see Figure 2.1). However, in the model that we consider (see Figure 2.2), the dynamical primitive at our disposal is the *leaky* integrator, given by the canonical first-order lowpass filter modelling the synapse (repeating equation 2.4, for clarity):

$$h(t) = \frac{1}{\tau}e^{-\frac{t}{\tau}} = \mathcal{L}^{-1}\left\{\frac{1}{\tau s + 1}\right\}, \tag{2.13}$$

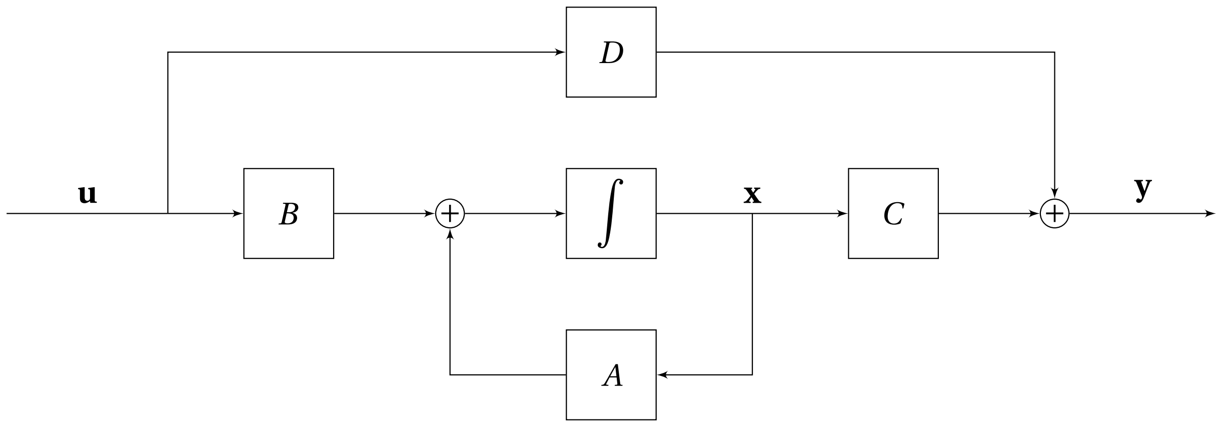


Figure 2.1: Block diagram for an LTI system. The integrator is driven by the signal $\dot{\mathbf{x}}(t)$. Reproduced from Voelker and Eliasmith (2018, Figure 1).

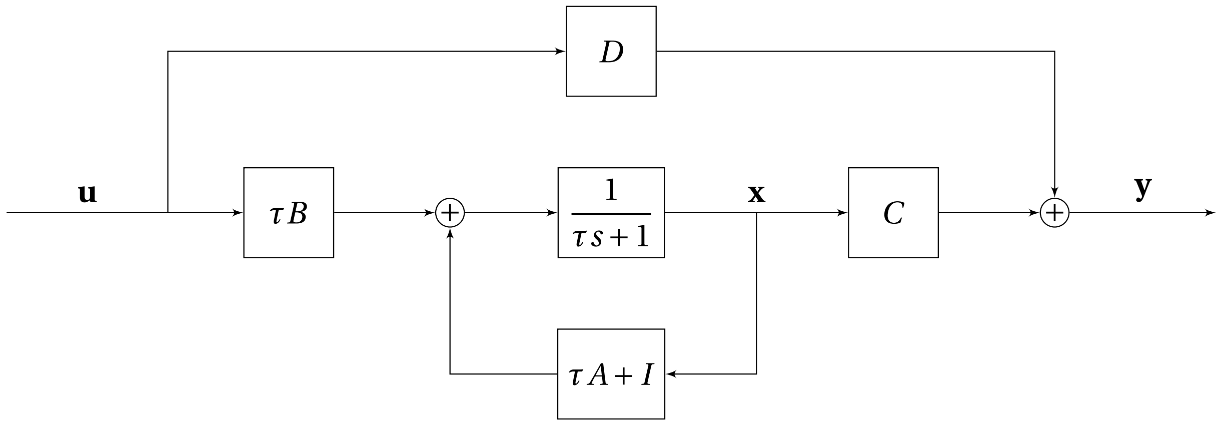


Figure 2.2: Block diagram for an LTI system, equivalent to Figure 2.1, with the integrator replaced by a first-order lowpass filter. The lowpass is driven by the signal $\tau\dot{\mathbf{x}}(t)+\mathbf{x}(t)$ to ensure that it implements the same system as in Figure 2.1. Reproduced from Voelker and Eliasmith (2018, Figure 2).

where $\mathcal{L}^{-1}\{\cdot\}$ denotes the inverse Laplace transform, and the latter representation is referred to as a transfer function.[8] Our approach is to represent the state-vector $\mathbf{x}(t)$ in a population of spiking neurons (Principle 1; equation 2.2), such that *this same* vector is obtained by filtering an appropriate transformation with a leaky integrator (Principle 2; equation 2.6). Thus, the goal of Principle 3 is to determine the transformations required to implement equation 2.12, given that the required $\mathbf{x}(t)$ is obtained by some convolution with a leaky integrator, rather than the perfect integrator depicted in Figure 2.1.

Principle 3 states that in order to implement equation 2.12 in a population of neurons that represents $\mathbf{x}(t)$, we must compensate for the effect of "replacing" the integrator with a leaky integrator (compare Figures 2.1 and 2.2) by driving the synapse with $\tau\dot{\mathbf{x}}(t) + \mathbf{x}(t)$ instead of only $\dot{\mathbf{x}}(t)$. This compensation is achieved as follows: implement the recurrent transformation $(\tau A + I)\mathbf{x}(t)$, and the input transformation $\tau B\mathbf{u}(t)$, but use the same output transformation $C\mathbf{x}(t)$, and the same passthrough transformation $D\mathbf{u}(t)$ (Eliasmith and Anderson, 2003, pp. 221–225). Specifically, this may be implemented in a spiking dynamical network by representing $\mathbf{x}(t)$ via Principle 1 and then using Principle 2 to decode the needed transformations.

The resulting model is summarized in Figure 2.2. Notably, both systems compute the exact same signals $\mathbf{x}(t)$ and $\mathbf{y}(t)$; the difference is the dynamical primitive, and the transformations needed to account for this change. The neural architecture that realizes this model consists of a population of neurons representing $\mathbf{x}(t)$, recurrently coupled to itself, with weights implementing the above transformations determined by equation 2.11, and each synapse filtering its spike-trains via convolution with equation 2.13. The outcome is that the desired state-vector, $\mathbf{x}(t)$, is linearly projected onto the postsynaptic current of each neuron. We depict and compare this architecture to related work in the following section.

The correctness of this "mapping" procedure relies on three assumptions: (1) the synapse model is equation 2.13, (2) the network is simulated in continuous time (or the discrete time-step is sufficiently small), and (3) the necessary representations and transformations are sufficiently accurate, such that the approximation error from equation 2.6 is negligible. In other words, assuming n is sufficiently large, the system depicted in Figure 2.1 may be implemented by a continuous-time SNN that realizes the model of Figure 2.2. In section 5.1, we provide a novel proof of this fact, and extend Principle 3 to relax the first and second assumptions. In section 3.2.1, we provide rigorous criteria for satisfying the third assumption.

Principle 3 is useful for accurately implementing a wide class of dynamical systems—integrators, oscillators, attractor networks, controllers, and even continuous memories (see chapter 6)—to solve specific problems that frequently arise in neural modelling (e.g., Eliasmith

[8] For comparison, the transfer function of an integrator is $\mathcal{L}\{1\} = \frac{1}{s}$.

and Anderson, 2000; Singh and Eliasmith, 2004; Eliasmith, 2005; Singh and Eliasmith, 2006). We provide details on such examples in section 4.2. Furthermore, the class of state-space models is isomorphic to the class of all finite-dimensional causal linear filters, or equivalently all rational (finite-order) proper transfer functions, which is a large and useful class of dynamical systems employed widely in control applications (Brogan, 1991). Given the ability of Principle 2 to compute nonlinear functions (i.e., equation 2.6), Principle 3 also naturally generalizes to nonlinear dynamical systems, as illustrated in section 5.1.3.

2.2.4 Relationships to other approaches

Conventional RNNs are notoriously difficult to train and interpret in practice (Bengio et al., 1994). The paradigm of Reservoir Computing (RC; Jaeger, 2001; Maass et al., 2002) has managed to overcome some of these limitations, by driving a fixed and randomly-connected ensemble of neurons, and learning the desired output via linear decoding. The First-Order Reduced and Controlled Error (FORCE; Sussillo and Abbott, 2009) method improves the performance of RC networks, by simultaneously re-encoding the learned output back into the reservoir, and training this feedback loop online using recursive least-squares (RLS; Haykin, 1991). Despite the success of these approaches, it remains difficult to interpret the high-dimensional representations employed by the reservoir, or to incorporate prior knowledge into the reservoir to improve task performance. Full-FORCE (DePasquale et al., 2018) solves the latter problem, by allowing the desired state-vector to be encoded into the network in the form of "hints" during training, but does so without accounting for the dynamics introduced by the synaptic τ, or carefully attending to the choice of encoding parameters, as in the NEF.

In all of these cases, however, random feedback ultimately leads to an inefficient use of neural resources for dynamical systems that are predominantly low-dimensional. That is, if we know the dynamics of the system that we would like to implement, then the NEF already does just that. Many have argued that dynamics in biological systems are unions of low-dimensional attractor manifolds (Sussillo and Barak, 2013; Cunningham and Byron, 2014; Waernberg and Kumar, 2017). The NEF imposes such a low-dimensional structure on the weight matrices themselves. Learning these dynamics is a somewhat separate matter; the NEF does not *limit* itself to situations where the dynamics are known *a priori*. Techniques from system identification (Nelles, 2013), optimization in the time-domain using a teacher signal (Duggins, 2017), and backpropagation through time (Rasmussen, 2018), may all be applied to learn the dynamics from raw data. Likewise, Nengo supports the use of random feedback and online RLS learning with full weight matrices, with or without spikes, thus reducing the NEF network to a full-FORCE network. The NEF is compatible with all such methods, but makes a preference for those that are amenable to theoretical analysis while being scalable and efficient in time

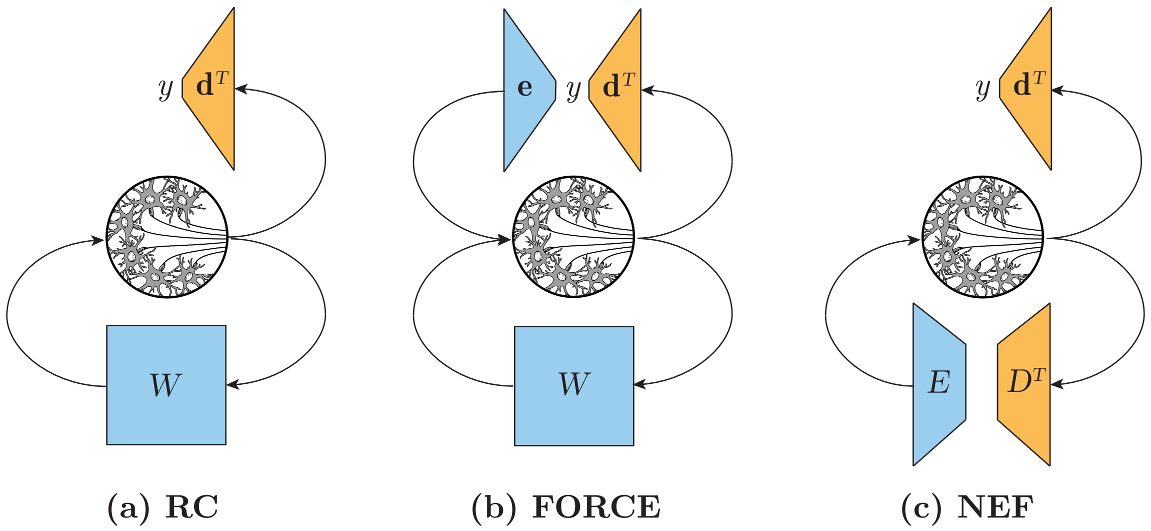


Figure 2.3: Comparing the recurrent architectures of (a) Reservoir Computing (RC), (b) First-Order Reduced and Controlled Error (FORCE), and (c) the Neural Engineering Framework (NEF). Blue weights are fixed and randomly chosen. Orange weights are learned either online or offline. Each ensemble is a pool of synapses and neurons that filter and nonlinearly encode the weighted activities. The input to each network is omitted for simplicity.

and space. We find the approach of introducing random high-dimensional feedback to be inefficient and unnecessary for our purposes (section 6.2.1), and furthermore re-encoding the target output into the network as a teaching signal can be counter-productive (i.e., reduce accuracy) even in simple cases (section 6.2.2).

We now consider the architectural relationships in considerable detail, focusing on the one-dimensional input-output case for notational convenience. In the case of RC networks (see Figure 2.3a), they make a conceptual separation between a fixed *reservoir* of randomly-connected neural units, and a learned *readout*. The reservoir is driven by encoding the input signal, $u(t) \in \mathbb{R}$, using a random vector, $\mathbf{e}^{\text{in}} \in \mathbb{R}^n$. The units within the reservoir are recurrently connected using a random matrix, $W \in \mathbb{R}^{n \times n}$, and a feed-forward readout vector, $\mathbf{d} \in \mathbb{R}^n$, is optimized to decode some target, $y(t) \in \mathbb{R}$. Intuitively, the reservoir expands the input into a set of nonlinear temporal basis functions—referred to as *echoes*—while the readout combines these dynamical traces to approximate the desired target. Since the readout is typically a linear combination of the reservoir's activity, the decoders may be learned via least-squares regression. This is equivalent to the optimization performed by the NEF on the output transformation, but using data that is collected by explicitly simulating the network over time. In contrast, the reservoir is left untrained with dynamical properties that remain fixed, independently of the desired task.

While RC solves the issue of training RNNs, the problem of efficient scaling remains. Indeed, as noted by RC theorists, "just simply creating a reservoir at random is unsatisfactory" and "setting up the reservoir such that a good state expansion emerges is an ill-understood challenge in many respects" (Lukoševičius, 2012b). Generally, the computational power of RC relies upon having a sufficiently large reservoir that represents a high-dimensional superset of the required dynamics, ultimately resulting in an inefficient use of resources for many systems (Wallace et al., 2013). This becomes problematic when considering the time and memory required to simulate RC networks, which scales as $\mathcal{O}\left(n^2\right)$, thus making large-scale simulations of the brain computationally prohibitive. In section 6.2.1 we show that rate-based reservoirs (ESNs) can be outperformed in accuracy, time, and memory, by *spiking* NEF networks of the same size.

The First-Order Reduced and Controlled Error (FORCE; Sussillo and Abbott, 2009) method extends ESNs by learning a low-rank component of the recurrent weight matrix. This can be decomposed into a separate feedback loop, that *autoencodes* the desired output back into the network (see Figure 2.3b). Specifically, the recurrent weights include an additional outer-product term, $\mathbf{ed}^T \in \mathbb{R}^{n \times n}$, where $\mathbf{e} \in \mathbb{R}^n$ is a fixed random vector, and $\mathbf{d} \in \mathbb{R}^n$ is the same decoding vector from before. By design, these weights decode an approximation of the desired output, $y(t)$, and subsequently encode it back into the reservoir, alongside the random mixing from W. This additional loop improves the performance the network, assuming the underlying dynamics are at least partially driven by a static function of the filtered and randomly encoded target signal. However, this is not always the case. In section 6.2.2, we show that this method of re-encoding the output signal does not always improve task performance. This discussion extends to the full-FORCE method (DePasquale et al., 2018), which learns a full-rank matrix that encodes the same state as in FORCE, but using additional degrees of freedom, similar to Tripp and Eliasmith (2006).

The Neural Engineering Framework (NEF; Eliasmith and Anderson, 1999, 2003) provides an alternative method for building dynamical neural networks (see Figure 2.3c). Rather than relying on random feedback, or always re-encoding the output (in the case of FORCE), the NEF optimizes the recurrent weights to represent the desired dynamical state. This can be understood as constructing an RC network with an optimized reservoir, although the approaches were developed independently. In the NEF, the desired dynamical state, $\mathbf{x}(t) \in \mathbb{R}^q$, is either expressed in closed form as a set of dynamical equations (in continuous-time or discrete-time), or provided via time-series data as is more typical within RC. Then, the recurrent weight matrix is factored into $ED^T \in \mathbb{R}^{n \times n}$, where $E \in \mathbb{R}^{n \times q}$ is a fixed encoding matrix, $D \in \mathbb{R}^{n \times q}$ is a learned decoding matrix, and a filtered combination of input and recurrent activations represent $\mathbf{x}(t)$. A central observation made by the NEF is that the optimal D can be determined from $\mathbf{x}(t)$, the

	Recurrence		Readout		Simulation	
Method	Online	Offline	Online	Offline	Explicit	Implicit
RC	×	×	×	✓	✓	×
FORCE	✓	∼	✓	∼	✓	×
Full-FORCE	✓	∼	∼	✓	✓	×
NEF	∼	✓	∼	✓	∼	✓

Table 2.1: Summary of the characteristics of each training method. Tildes (∼) denote less-commonly advertised, but sensible, use-cases (see text for details).

selection of neuron models, the models of postsynaptic current (PSC), and whether the simulation is analog or digital (Voelker and Eliasmith, 2018). In the same way as FORCE, the NEF may include $\mathbf{d}$ as a column of D, and $\mathbf{e}$ as a column of E, to re-encode $y(t)$ – equivalent to asserting that a filtered version of $y(t)$ is a dimension of $\mathbf{x}(t)$. This assumption is made if and only if it is helpful (e.g., to perform integration; Singh and Eliasmith, 2004). If all relevant state-variables are identified, and all target models are ideally leveraged, then the high-dimensional dynamics introduced by W serve absolutely no purpose. To this point, we find that even the expansion of the input signal into a rich set of nonlinear temporal basis functions can be accomplished by an optimally-designed, closed-form, dynamical system (see section 6.1.1).

Now we shift focus to discuss the method(s) of training in some detail (see Table 2.1). Within the NEF, the important distinction between ESNs and LSMs is captured simply by the choice of $G_i\left[\cdot\right]$ (see equation 2.2). In either case, we can identify the neural activity of the reservoir with $a_i(t)$, where $\mathbf{x}(t)$ is some *unknown* (latent) state that depends on both $u(t)$ and the dynamical properties of the reservoir. Note that characterizing $\mathbf{x}(t)$ is precisely the problem faced when trying to understand what information RC networks are representing. Regardless, the linear readout amounts to the following approximate nonlinear function of $\mathbf{x}(t)$:

$$y(t) \approx \sum_{i=1}^{n} (a_i * h)(t)\mathbf{d}_i^{\mathbf{f}}. \tag{2.14}$$

This has the same form as equation 4.8 from Principle 2 of the NEF. The important difference lies in how the decoders $D^{\mathbf{f}}$ are trained. Rather than solving for $D^{\mathbf{f}}$ using equation 2.9, the network is explicitly simulated to obtain the exact temporal basis functions. In general, the RC training method relies on *explicit* simulation, since $\mathbf{x}(t)$ is unknown. In the NEF the simulation is typically *implicit* within Principle 2, but can also be made explicit in exactly the same manner (Friedl et al., 2016; Duggins, 2017).

The NEF, by default, avoids explicit simulation because it is more costly (especially as the time-step becomes small), which is of practical importance when engineering large-scale networks (e.g., Eliasmith et al., 2012). However, the method does have the benefit of automatically refining any additional, non-Gaussian, error in the decoding introduced by the substitution of equation 2.7. Furthermore, the readout can learn to compensate for other sources of error and even mischaracterizations of the state-variable. On the one hand, this can simplify the engineering work involved in training the network, while on the other hand it can mask the effects of costly modelling mistakes.

For the non-adaptive, non-spiking, case the explicit training method adds absolutely no improvement, assuming the representation has been correctly characterized. Overall, the explicit training methodology is only suitable for black-box applications where there is little-to-no visibility into the latent dynamical state-variables that generated the data, or for when randomized high-dimensional feedback makes it generally intractable to analytically determine the trajectory of the network's dynamics.

From the perspective of RC, the NEF provides a way to solve for the recurrent connection weights by imposing a low-dimensional dynamical state $\mathbf{x}(t)$ on the reservoir. This can be thought of as building a *structured* reservoir that is optimized for a restricted subset of functions. Specifically, we may include a set of temporal basis functions (that incorporate prior knowledge of the desired function) using Principle 3, and then apply any training method to this specially constructed reservoir. The readout may still capture unanticipated nonlinear relationships between the state and the target, and this readout may still be trained from data. The NEF provides a way to understand the resulting computations, and to systematically relate the dynamics of the state-variables to the parameters of the computational models. This allows us to, for instance, understand the network-level effects of including different synapse models. Furthermore, this reduces the cost of simulation time and memory by a factor of $\mathcal{O}(n)$ for constant dimensionality, since the number of connection weights for RC scale as $\mathcal{O}\left(n^2\right)$.

The balanced SNN approach of Boerlin and Denève (2011); Boerlin et al. (2013) requires a separate discussion. These methods are closer to the NEF in that they begin with a model of the desired dynamics, and then map that system onto a latent representation within an SNN using factored weight matrices. These methods do not rely on randomized feedback, nor explicit simulation of the network over time to train its parameters. Rather, they derive a closed-form solution for the parameters. These parameters, in effect, deterministically control the spike-timing of each neuron, in such a way that the population collaboratively keeps any deviations from its ideal state bounded above by the peak magnitude of a single filtered and weighted spike, n^{-1}. In particular, each neuron's spike signals a constant deviation from the ideal, which causes all neurons to instantly update their local state to track the global error. The outcome is precision that scales as $\mathcal{O}(n)$, that is, linearly in the number of neurons, or the total number of spikes (Boahen, 2017). However, this scheme applies only to *linear* dynamical systems, relies on near-instantaneous communication between all neurons with a sufficiently small time-step, and requires linear neurons with relatively minimal leakage and no refractory period (Boerlin et al., 2013).[9]

The work has been extended to support more biologically-plausible mechanisms, but the scaling becomes $\mathcal{O}\left(\sqrt{n}\right)$ as in the NEF (Schwemmer et al., 2015, Figure 12d). In general, it is unclear how far these assumptions can be relaxed before scaling regresses to $\mathcal{O}\left(\sqrt{n}\right)$, which is of relevance when considering our targets of modelling biological systems and compiling onto neuromorphic hardware – both of these targets regularly violate the above assumptions. As we show, the NEF can account for and remain robust to non-idealities, such as discrete time-steps, higher-order synapses, and detailed neural dynamics. We find that some of the remaining differences between the statistics of spike-trains, and the number of spikes required to achieve some level of precision, can be made up for in the NEF by delaying spikes a variable length of time that is sensitive to the global error (unpublished) – but we cannot currently offer a biologically-plausible explanation to learn such a mechanism. Nevertheless, we believe further investigation into these areas of research are of utmost importance for determining the computational-theoretic limits of SNN precision with respect to some target set of dynamical primitives, and for maximizing the efficiency of information transmitted per spike (Eliasmith and Anderson, 2003, pp. 92–127).

Returning to the literature on RNNs and backpropagation, there exists a range of methods designed to convert a conventionally-trained deep network, that uses static non-spiking (i.e., rate-based) nonlinearities, into an equivalent SNN (Cao et al., 2015; Rueckauer et al., 2017; Yousefzadeh et al., 2019, and various methods mentioned in section 2.1.1). This comes with some minor loss in precision due to spike-variability. However, these methods on their own do not address the difficulties in training RNNs. Moreover, they do not immediately extend to recurrent SNNs, that is, unless the RNN has been trained with the same filter as the SNN. But of equal importance, is that common to all of the methods mentioned above are limitations regarding the nonlinearities employed by both networks. In particular, they commonly assume rectified linear activation units, corresponding to specifically-engineered integrate-and-fire units in the SNN, which limits the applicability of these methods in the context of biologically-grounded models and implementation on neurormorphic hardware.

[9]We thank Daniel Rasmussen for having implemented these methods in Nengo (unpublished), which served to validate these criticisms and to inform this discussion through additional experimentation.

2.3 Neuromorphic Computing

The term *neuromorphic computing*—also known as neuromorphic engineering—refers broadly to the use of specialized hardware to emulate computational principles of biological nervous systems (Mead, 1989; Liu et al., 2002). Here, we focus more specifically on the synthesis of Spiking Neural Networks (SNNs)—in both analog and digital hardware—for low-power, dynamical, brain-like, computation (Boahen, 2017).

This section provides a brief overview of the landscape of neuromorphic hardware, with an emphasis placed on state-of-the-art hardware made available to the Centre for Theoretical Neuroscience, at the University of Waterloo, for academic research, namely: SpiNNaker (Furber et al., 2014), Braindrop (Neckar et al., 2019), and Loihi (Davies et al., 2018). At the same time, this overlaps with the efforts of many companies, including: Applied Brain Research (ABR), Intel, IBM, Numenta, Qualcomm, BrainChip, General Vision, HRL Laboratories, Brain Corporation, Knowm, Samsung, and Vicarious FP (QY Research, 2018; Crunchbase, 2018) – although we focus primarily on ABR's marriage of Nengo and the NEF with neuromorphic hardware from Stanford's Brains in Silicon Lab (i.e., Braindrop) and Intel (i.e., Loihi). A non-exhaustive list of other architectures, many of which implement NEF networks, are provided in section 2.3.4.

Coming from a purely computational-theoretic perspective, there is nothing special about neuromorphic hardware; alternative hardware architectures cannot move beyond what is already Turing-computable. There is no logical nor physical reason to believe that neurons and synapses somehow enable new computations that were previously unable to be emulated using available technology. Instead, neuromorphic architectures aim to implement specific classes of algorithms, realized via massively-parallelized event-driven computations that are sparse in both space and time, using dramatically less power (Tang et al., 2017).

The human brain has been estimated to consume as little as 10–20 fJ[10] on average per "synaptic event" (Cassidy et al., 2014; Boahen, 2017), totalling approximately 20 W across its ~10^{11} neurons and ~10^{14} synapses (Koch, 2004). After more than 30 years of work (Cassidy et al., 2013), the state-of-the-art in neuromorphic computing is on the verge of reaching this level of performance, with Braindrop, Loihi, and TrueNorth (Merolla et al., 2014), each coming within a factor of 10^1–10^3 for roughly equivalent measures of energy per synaptic operation and typical network configurations (Neckar et al., 2019). Such potential energy savings are not only tremendous, but a prerequisite to reach the exascale (10^{18}) level of computation achieved by the human brain – unless one has access to a super-computing cluster powered by a nuclear power plant producing on the order of 10^9 watts (Furber, 2012; Benjamin et al., 2014). However, such power measurements are not straight-forward to interpret, as there are many other factors

[10] One femtojoule (fJ) is equivalent to 10^{-15} joules (J).

such as the cost of scaling, flexibility of the hardware, and open questions surrounding whether the energy per synaptic event is the best metric for evaluating system performance, and how much speed, accuracy, and biological detail is necessary (Eliasmith, 2013) – all of which make fair benchmarking an incredibly challenging problem (Stewart et al., 2015).

However, improved power-efficiency often comes with the costs of reduced programmability and narrowed application spaces; the more you know about the problem that you are trying to solve, the more that can be "baked in" to the hardware to save energy and space, while sacrificing flexibility. This fundamental trade-off between being able to solve fewer problems more efficiently, versus more problems less efficiently, is one that the community as a whole is still trying to navigate. It appears likely there will be no one-size-fits-all solution, and rather a variety of strategies will be needed to mirror the degrees of architectural specificity observed in the human brain. Yet, there are many common features between neuromorphic hardware architectures, including: dynamic spike-generation, massive parallelism, memory colocated with computational elements, heterogeneity, sparsity, structured connectivity, minimization of spike traffic and synaptic events, and real-time interfacing. These features are anticipated to provide robust, scalable, and energy-efficient solutions to the domain of tasks where our brains excel: processing sensory information, language, cognition, and motor behaviours – all dynamically coupled to the time-scales of our environment.

To fully unlock the potential of such hardware, we require significant advances at the intersection of computer science, computational neuroscience, cognitive modelling, electrical engineering, physics, and mathematics. This involves discovering new algorithms, developing new frameworks, and training the next generation of programmers to think differently about problems – but the benefits are far-reaching: real-time computation that can in principle scale up to the perceptual, cognitive, and motor capabilities of the human brain, without requiring the energy production of a nuclear power plant.

In chapter 5, we summarize some of the main challenges surrounding the use of neuromorphic hardware, and demonstrate how the NEF solves many of these problems. More generally, we provide a theoretical framework and extensions to formally understand the class of computations that neuromorphic hardware enables, given its computational elements and constraints, including resource budget and desired precision. A novel class of algorithms involving temporal computations are explored and analyzed in detail in chapter 6. Applications running on neuromorphic hardware are provided in chapter 7.

2.3.1 SpiNNaker

The SpiNNaker[11] project formally began in 2005 as a collaboration between several universities and industrial partners, housed at the University of Manchester, with the primary goals of simulating large-scale SNNs in real-time, and investigating new architectures which "lead to fundamentally new and advantageous principles for energy-efficient massively-parallel computing" (University of Manchester, 2012; Furber et al., 2014). Secondary goals include a balance between power, cost, and programmability, in order to make SpiNNaker accessible to a wide range of researchers spanning neuroscience and cognitive modelling communities.

The basic building block of SpiNNaker is a chip multiprocessor (CMP) containing 18 ARM968 cores and 128 MB SDRAM (Painkras et al., 2013; Furber et al., 2013). Each core was prescribed to simulate 10^3 spiking neurons and 10^6 synapses, although Nengo's integration doubled this target by achieving 2,000 neurons per core through the use of factorized weight matrices (Mundy et al., 2015). Each CMP is fully digital, locally synchronous, clocked at some frequency (e.g., 200 MHz), and allotted some amount of wall time (e.g., 1 ms) to complete each time-step. Thus, both neural and synaptic elements are virtualized (i.e., reusing the same shared silicon) by simulating as many of them as possible within each core given the allotted wall time. The limiting factor in scaling each core is predominantly the amount of available SDRAM split between cores, and the volume of spike traffic transmitted between CMPs. The architecture scales up to 2^{16} CMPs—over 10^9 neurons and 10^{12} synapses—asynchronously connected to one another through a 2D triangular toroidal mesh. An instantiation that realizes their initial target of 1% of the human brain, using 10^6 cores, was finally reached as of 2018.

Studies involving power-usage are surprisingly scarce, but one such study demonstrated that scaling to one quarter of a million neurons required less than 30 W of power, in the worst-case (Stromatias et al., 2013), or on the order of 10^5–10^6 times more power than a human brain of the same size.[12] This is a significant improvement upon high-performance computing (HPC) architectures, which devour at least 10^8 times more power than a human brain (Furber, 2012). However, a more recent study by van Albada et al. (2018) concluded much more conservatively that SpiNNaker is still a factor of 10^7–10^9 away from the efficiency of the mammalian brain.[13] In this scenario SpiNNaker was outperformed by NEST running on an HPC cluster, and likewise a smaller-scale implementation of an associative memory on SpiNNaker performed comparably to NEST on an Intel Core i7-4710MQ processor (Stöckel et al., 2017a). Similarly, Knight and Nowotny (2018) found that GPUs improved both the simulation

[11] SpiNNaker stands for "Spiking Neural Network Architecture."

[12] The human brain has 400,000 times as many neurons, and uses 20 W of power.

[13] The lower-bound on this estimate comes from a hypothetical extrapolation of their results. Furthermore, the range becomes 10^8–10^{10} if using the estimate of 10–20 fJ per synaptic event for the brain.

time and energy-to-solution for cortical circuit simulations, compared to SpiNNaker and CPU-based HPC hardware. Meanwhile, Sugiarto et al. (2016) estimated that both SpiNNaker and an FPGA provide a 10^2–10^3-fold improvement over both CPUs and GPUs, on an image processing task. Clearly it matters what one is actually scaling towards.

Nengo has been ported to SpiNNaker (Galluppi et al., 2012; Mundy et al., 2015) and used to run the major components of the Spaun brain model in real-time (Mundy, 2016). This goal represents a 10^4-fold speedup over a conventional X86 CPU implementation (Stewart and Eliasmith, 2014). This implementation required significant work by Mundy (2016) to leverage the flexibility of the SpiNNaker architecture, and so we do not go into detail here. General support has also been added for both supervised learning (Davies et al., 2013) and unsupervised learning (Knight et al., 2016) to learn human-scale vocabularies online using Nengo. Meanwhile, Nengo has also been deployed on a prototype autonomous robot equipped with a SpiNNaker board to sense and act in real-time (Galluppi et al., 2014). PyNN has also been ported over to SpiNNaker, further highlighting the flexibility of the architecture (Rhodes et al., 2018). Several other applications of SpiNNaker are reviewed by Rhodes et al. (2018), independently of Nengo.

The next generation of SpiNNaker, dubbed SpiNNaker 2, is currently in development with a prototype chip available for testing (Liu et al., 2018). The hardware promises to improve upon the speed- and power-efficiency of the first generation by use of improved fabrication processes, advanced per-core power management techniques (Hoeppner et al., 2019), and specialized hardware acceleration for the exponential function (Partzsch et al., 2017). Liu et al. (2018) deploys a deep neural network on the prototype SpiNNaker 2 test chip, realizing power improvements on the order of 10^2 versus a traditional X86 CPU implementation. Yan et al. (2019) demonstrates a structural plasticity learning rule on the same prototype to validate its power management and hardware acceleration. Nengo may be supported at the same time as the actual chip [personal communication].

2.3.2 Braindrop

Braindrop (Neckar et al., 2019) is a mixed-signal—both analog and digital—neuromorphic architecture that attempts to realize the energy-efficiency of the human brain, without sacrificing the flexibility required to simulate functional large-scale cognitive systems such as Spaun (Eliasmith et al., 2012). This project was a five year collaboration between Stanford, Yale, and the University of Waterloo, and was made possible in this time frame by building upon many of the tools and lessons learned from its predecessor, Neurogrid (Benjamin et al., 2014).

Both Neurogrid and Braindrop take a dynamical systems-based approach (Arthur and

Boahen, 2011; Gao et al., 2012) to realize network-level computation. In particular, the device-level physics of subthreshold analog circuits (Andreou et al., 1991)—emulating the behaviours of neurons and synapses—are leveraged by weighting the digital communication of spikes, to engineer dynamical systems (Dethier et al., 2011). It is argued that an appropriate mixture of analog computation for nonlinear encoding—sparsifying signals across both space and time—and digital communication for spike transmission, is responsible for the energy-efficiency of mammalian brains (Boahen, 2017). This hypothesis makes such architectures a natural target for NEF networks, in particular due to its modelling of the dynamical primitives encoding signals into spikes, and its efficient procedures to solve for the weights in the communication fabric.

The NEF has been deployed on Neurogrid primarily to implement control algorithms (Dethier et al., 2011; Choudhary et al., 2012; Menon et al., 2014). However, the use of Neurogrid requires advanced calibration (Kauderer-Abrams and Boahen, 2017) and temperature-compensation techniques (Kauderer-Abrams et al., 2017) in order to tame the analog variability – both of which contribute to it being difficult to use in practical applications.[14]

Braindrop is designed explicitly with Nengo and the NEF in mind, by way of a novel mapping (Voelker et al., 2017a; Neckar et al., 2018), accompanied by a number of inventive solutions including: temperature-invariance (Kauderer-Abrams et al., 2017; Reid et al., 2019; Benjamin and Boahen, 2019), H-tree routing (Fok and Boahen, 2018), sparse encoding by spatial convolution (Feinstein, 1988; Neckar et al., 2019), and decoding by accumulative spike-thinning (Fok et al., 2019). Of key notability is the fact that Braindrop, in essence, implements a factorization of the weight matrices akin to equation 2.11, which has cut down on area requirements for SRAM and permitted the sharing of synaptic circuitry across neurons, in turn leading to an estimate of 381 fJ per equivalent synaptic operation for network configurations typical in Spaun. A significant amount of detail addressing the design of the chip is available in Neckar (2018); Fok (2018); Neckar et al. (2019). The success of this project has prompted the formation of a start-up company, Femtosense, named after the mere femtojoules that synapses consume to sense and react (Crunchbase, 2018).

As stated above, the design of Braindrop is informed by many of the lessons learned from Neurogrid. Detailed models have been studied at the circuit level (Benjamin and Boahen, 2019) and NEF-related methods have been employed at the network level (Voelker et al., 2017a; Reid et al., 2019), in order to simultaneously tame and account for the non-idealities introduced by transistor mismatch, parasitic capacitances, and temperature variation. In chapter 5, we

[14]For example, a live demonstration that used Neurogrid to control a robotic arm, which had previously worked within an air-controlled climate, no longer functioned in the conference hall as it was bombarded by the warmth of onlookers [personal communication].

expose a few of these extensions at a theoretical level. In chapter 7, we provide applications that demonstrate robust dynamical computation on Braindrop.

2.3.3 Loihi

Loihi (Davies et al., 2018) is Intel's fully digital, barrier-synchronized, neuromorphic architecture, that promotes significant flexibility in its SNN feature set, while being optimized for low power consumption. It consists of 128 "neuromorphic cores" operating asynchronously from one another. Each core time-multiplexes 1,024 neural units—configurable as sets of trees of dendritic compartments to form neurons—in a pipelined manner. In contrast to SpiNNaker, barrier messages are by default transmitted between cores in order to keep the system synchronized to a global time-step. This has the advantage of provably guaranteeing globally deterministic behaviour, but the disadvantage of always waiting on the slowest core. Nevertheless, when compared to a globally-clocked system, this has the advantage of each time-step only taking as long as it needs.[15]

By virtue of being fully digital and barrier-synchronized, there exists far fewer challenges presented to its users when compared to solutions in the mixed-signal, purely analog, or fully asynchronous digital domains; in particular, the behaviour of the chip is guaranteed to be reliable and predictable. When compared to simulating SNNs on traditional hardware (e.g., an x86 CPU), the challenges that remain include quantization (i.e., truncation) of real values (e.g., weights, synaptic states, and neuronal states) due to the limited bit-precision of fixed-point numbers (1–9 bits), and constraints on connectivity and memory. The Nengo-Loihi emulator (GitHub, 2019b), developed by Applied Brain Research (ABR), accurately reproduces many of these limitations using detailed software models of the underlying hardware (Blouw et al., 2018). In addition, software tools exist to map SNNs onto Loihi while optimizing for power-efficiency with respect to the constraints imposed by the hardware (Lin et al., 2018a,b).

Loihi supports a broad variety of features that make it accessible to a wide range of researchers experimenting with the synthesis of large-scale SNNs. For example, stochastic noise can be added to internal states, discrete delays may be introduced to spikes and PSCs, synapses can have higher-order dynamics, dendritic trees may be constructed by repurposing the neural units within each core, neuron models may include adaptive terms, learning rules are described using a general template, and finally there is support for several weight compression formats, spike multicast, and closed-loop control via embedded x86 cores (Davies et al., 2018).

[15]One algorithmic time-step may take longer than the next, if for example learning rules are applied irregularly, or depending on relative workloads.

The flexibility and digital nature of Loihi comes at the cost of increased power-consumption relative to fully analog or mixed-signal architectures. Simulations from Davies et al. (2018, Table 2), compared in Neckar et al. (2019, Table 3), indicate that Loihi consumes on the order of 23.6 pJ[16] per synaptic operation – or two orders-of-magnitude more than Braindrop's equivalent measure. Regardless, the analog variability intrinsic to Braindrop changes the scope of target applications, and moreover the first demonstration of Loihi represents a three orders-of-magnitude improvement over a comparable solution running on a CPU. We speculate that the power-efficiency of Loihi, relative to Braindrop, can be improved by supporting factored weight matrices, and refining the architecture according to the network topology and particular feature set leveraged by the example application.

ABR has ported Nengo over to Loihi and used the NEF to simulate a nonlinear adaptive controller on a robotic arm (DeWolf et al., 2016, and personal communication). However, since Loihi does not support the use of factored weight matrices at the hardware level, a substantial benefit of the NEF—namely, a linear reduction in the number of multiplies and memory accesses per time-step—is not leveraged at the hardware level. As a way to resolve this at the software level, Nengo currently creates intermediate populations to represent latent variables explicitly in spikes, which has the adverse effects of amplifying representational error, increasing the number of neurons, and introducing additional rounds of filtering between layers. Alternatively, weight matrices can be left unfactored and fully-connected, but this misses out on potentially dramatic savings.

Nengo DL (Rasmussen, 2018) has been used to implement a fully-connected deep network performing keyword spotting, on Loihi, while outperforming state-of-the-art hardware in energy cost per inference (Blouw et al., 2018). Tang et al. (2019) implemented a simultaneous localization and mapping (SLAM) algorithm for robotic navigation on Loihi. After extensive review of the literature, we have found these two papers to be the only currently published examples, outside of Intel's demonstration of sparse pattern representation (Tang et al., 2017; Davies et al., 2018), to deploy an SNN on the physical chip.

It is currently unclear how important or useful each of Loihi's features are relative to one another, and especially in the context of target applications. We believe that an important avenue of research, and the role of Nengo and the NEF in this context, is to facilitate such exploration, by systematically harnessing relevant features for useful computation, which can then feed back into the co-design of future iterations (as was the case for Neurogrid and Braindrop). Generally speaking, specific classes of computations that leverage particular subsets of features will encounter distinct trade-offs when it comes to resource usage, precision, energy-to-solution, and time-to-solution; mathematical frameworks, such as the NEF, participate by

[16]One picojoule (pJ) is equivalent to 10^{-12} J, or 10^3 fJ.

providing a unified way of characterizing such trade-offs. A primary focus of chapters 5–7 is to take a concrete step in these directions by explicitly incorporating some of Loihi's features into the NEF and demonstrating dynamical systems functioning on the physical hardware.

2.3.4 Others

The literature on neuromorphic computing is extensive[17], dating back to the late 80's and recently inspiring a renewed interest from many historically-separate communities. This has resulted in a number of architectures mentioned below, which were not made available to us, to be considered outside the scope of this thesis. A number of excellent review articles by experts widely regarded to have advanced the field, are made available by Bartolozzi et al. (1999), Indiveri et al. (2011), Cassidy et al. (2013), Thakur et al. (2018), and Rajendran et al. (2019). A review that considers the role of biomimicry—the emulation of nature's solutions in order to solve real-world problems—in the context of power-efficient SNNs and neuromorphic computing, is provided by Krichmar et al. (2018).

Some of the earliest examples of neuromorphic engineering date back to the work of Sivilotti et al. (1985); Mead and Mahowald (1988); Boahen et al. (1989); Mahowald and Douglas (1991). Many of these techniques have since been refined, extended, and scaled up to showcase a large variety of analog, digital, and mixed-signal systems (excluding those mentioned thus far): Spikey (Pfeil et al., 2013), BrainScaleS (Schemmel et al., 2010; Aamir et al., 2018a,b), DYNAPs (Moradi et al., 2018), IBM's TrueNorth (Merolla et al., 2011, 2014; Esser et al., 2016), Sydney's DeepSouth (Thakur et al., 2018), ODIN (Frenkel et al., 2018), ROLLS (Qiao et al., 2015; Glatz et al., 2018), COLAMN (Wijekoon and Dudek, 2012), STPU (Smith et al., 2017), DANNA (Daffron et al., 2016; Mitchell et al., 2018), and many other related prototypes (e.g., Glackin et al., 2009; Moradi and Indiveri, 2011; Brink et al., 2013; Moradi and Indiveri, 2014; Park et al., 2014; Azghadi et al., 2015; Binas et al., 2016; Kim et al., 2018; Chen et al., 2018a; Zheng and Mazumder, 2018; Larras et al., 2018).

Among these systems, a number support the compilation of NEF networks, including a VLSI prototype (Corradi et al., 2014), several FPGA implementations (Naylor et al., 2013; Wang et al., 2014; Berzish et al., 2016; Wang et al., 2017) including ABR's own FPGA backend (GitHub, 2019a), several GPUs (Bekolay et al., 2014; Rasmussen, 2018; Blouw et al., 2018), and most recently, TrueNorth (Fischl et al., 2018).

An emerging framework called the Neural and Synaptic Array Transceiver (NSAT; Detorakis et al., 2018), which focuses on data-driven autonomy via online learning on neuromorphic

[17] See Schuman et al. (2017) for a survey that references nearly 3,000 other papers.

hardware, has been validated on a field-programmable gate array (FPGA). Likewise, an engine that repurposes generic computational elements in order to flexibly support a variety of network topologies at scale, has been used to implement polychronous SNNs on an FPGA (Wang, 2013; Wang and van Schaik, 2018).

There also exist multiple proposals to leverage memristors as an alternative hardware primitive for SNN computation in mixed-signal neuromorphic architectures, due to their compact and non-volatile (i.e., maintains memory without power) properties (Payvand et al., 2018). Several investigations are currently underway to explore the practical challenges associated with these proposals (e.g., Chang et al., 2013; Kudithipudi et al., 2016; Cady et al., 2018; Boybat et al., 2018).

Likewise, many other unconventional substrates for SNNs are actively being explored, such as nanoscale spintronic oscillators and multiferroic devices (Torrejon et al., 2017; Hoppensteadt, 2017; Manipatruni et al., 2018), analog non-volatile memory (Ambrogio et al., 2018), organic electronics (van De Burgt et al., 2018), phase-change oxides (Zhao and Ravichandran, 2019), photonics (Tait et al., 2018; Shainline, 2018), and superconducting circuits based on quantum phase-slip junctions (QPSJs; Cheng et al., 2019).

Chapter 3

Analysis of the Neural Engineering Framework

In this chapter, we provide a thorough examination of the Neural Engineering Framework (NEF; Eliasmith and Anderson, 2003) that is not currently published elsewhere. The goals are two-fold: (1) explore the consequences of its computational principles at a theoretical level, and in doing so challenge some common myths that have become associated with the NEF since its inception, and (2) assess its suitability as a framework for deploying SNNs on neuromorphic hardware by proving its correctness, scalability, completeness, robustness, and extensibility.[1]

3.1 Computational Sub-Principles

In this section, we focus primarily on the third principle of the NEF, and elucidate some high-level observations, or "sub-principles," that follow from its adoption. In doing so, we expose some important insights, including the effects of structuring deep RNNs, a framework to understand the roles of dynamical primitives, the observation of chaos in simple NEF networks, its strategy for neural encoding, and a relationship between the energy of synaptic events and signal bandwidth.

[1]To help delineate separate considerations, some sub-sections are significantly shorter than others.

3.1.1 Reducibility to a single layer

In section 2.2, we focused primarily on a single recurrently connected population of neurons. However, any multi-layer NEF network, where each layer may be recurrent, with feed-forward connections and feedback connections optionally included between any two layers – may be rewritten as a single recurrently connected population. Indeed, even for models as complex as Spaun (Eliasmith et al., 2012; Choo, 2018), the Nengo builder strips away the model specification, and leaves only channels that transmit spikes, and transfer functions that operate on these spikes (Bekolay et al., 2014; Gosmann and Eliasmith, 2017). The topology of the network reflects spatial and organizational structure (e.g., hierarchies), which may be used to constrain and inform the design of biologically-grounded models and to aid in understanding function. The mechanisms underneath are mere signals and operators, all coupled to one another to implement some dynamical system.

To examine some of the consequences of this observation, we assume without loss of generality that there are m layers, and that each layer uses n neurons to represent a q-dimensional state-vector. The procedure for rewriting the network is as follows:

1. Stack all of the populations together into a single layer that consists of nm neurons.
2. Stack all of the state-vectors together into one vector, $\mathbf{x}(t) \in \mathbb{R}^{qm}$, to represent the global state-space.
3. Construct the encoding matrix, $E \in \mathbb{R}^{nm \times qm}$, by inserting the encoding matrix from the i^{th} population into the $(i, i)^{\text{th}}$ block, such that E is in block-diagonal form (note: there are $m \times m$ blocks and each is $n \times q$).
4. Construct the decoding matrix, $D^{\mathbf{f}} \in \mathbb{R}^{nm \times qm}$, by inserting each decoding matrix $D^{\mathbf{f}_{i,j}}$ into the $(i, j)^{\text{th}}$ block, where $\mathbf{f}_{i,j}$ is the function computed between the i^{th} presynaptic population and the j^{th} postsynaptic population.

Then Principles 1–3 apply in the same way to this single population, while being functionally equivalent to the original network.

This reveals that the problem of constructing and training arbitrary network topologies may be reduced to the special case of a single layer. Conversely, we do not lose any computational power by limiting our focus to a single layer. However, the encoding matrix is extremely sparse with $(m-1)/m$ percent of its coefficients set to zero. Likewise, the block-structure of the decoding matrix is isomorphic to the graph structure of the original network.

This characterizes the functional role of network structure as: partitioning the global state-space into a union of state-vectors with a range of time-constants, sparsifying its encoders, and rearranging the decoders into a block-structure that mirrors the interactions between subspaces. In terms of computation, the network structure dramatically reduces the time and memory requirements that would otherwise be needed for a full-rank $nm \times nm$ matrix multiplication. In terms of training, the challenges involved in globally optimizing an RNN may be viewed under the lens of identifying relatively low-dimensional subspaces acting along various time-scales and their interactions with one another.

3.1.2 Heterogeneity of dynamical primitives

We make the observation that neurons (equation 2.3), synapses (equation 2.4), and even the functional behaviour of entire recurrent networks (equations 2.1 and 2.12), are all described in terms of differential equations. These dynamical systems, each of which can be thought of as a *primitive* for use in some larger computation, all have distinct filtering properties, linear and nonlinear transfer functions, and internal states that evolve over time in response to some input in order to produce some output.

A central insight from deep learning is that a variety[2] of static nonlinear functions, when structurally composed, enhance the *spatial complexity* of network function. An analogous insight in the above context is that a variety of transfer functions, when temporally composed, enhance the *temporal complexity* of network function. In particular, the time-constants of synaptic currents, membrane leakages, and recurrent network dynamics – all act along different time-scales. In effect, NEF networks compose rich temporal functions by leveraging dynamical primitives that are heterogeneous in time. A compelling example is provided in section 6.2.3, where we apply BPTT to a deep recurrent NEF network to predict a chaotic time-series.

3.1.3 Duality of macroscopic and microscopic dynamics

As a corollary to the previous observation, once all primitive operations are expressed in the same language (i.e., differential equations), they become both *composable* and *interchangeable*. To the point of interchangeability, one can implement a lowpass filter by using a single synapse, or by training a population to emulate the same dynamical system via Principle 3.[3] To the

[2]The variety of nonlinear responses comes from the distribution of weights and biases, analogous to the heterogeneity obtained in equation 2.2 via encoding vectors, gains, and biases.

[3]As inconsequential as this point might sound, a potent example is when one student, who had been working with Nengo for many years, used a network containing hundreds of neurons to implement the same computation as a single alpha synapse. This was due to not having them described in the same language.

point of composability, one can substitute the transfer function of a single synapse with that of an entire recurrent network.

We illustrate both points with a specific example by considering the following architecture:

$$A \rightarrow (B \rightarrow B) \rightarrow A,$$

where A and B are separate neural ensembles and "$\rightarrow$" denotes a synaptic weight matrix. That is, B is locally-recurrent, and A is globally-recurrent via the outer loop through B. We may encapsulate $H = B \rightarrow B$ as a dynamical primitive, and then re-interpret the whole system as:

$$A \rightarrow_H A,$$

where "$\rightarrow_H$" denotes some weight matrix where the *synapse* model has been replaced by H – akin to a "virtual" synapse, whose dynamics are implemented by B under the hood. This is simply a mathematical "trick," but the practical consequence is that we may use the same theory and the same tools to understand the composition and substitution of these systems. This is made possible since we employ a common language and framework for describing and leveraging dynamical systems.

In section 5.1.1, this idea forms the core idea for our proofs that extend Principle 3. In section 6.2.7, we demonstrate a method of systematically harnessing miniature delays in axonal spike-transmission to improve the accuracy of longer network-level delays.

3.1.4 Unpredictability, chaos, and strange attractors

An apparent misconception associated with NEF networks is that we already "understand" the computations being engineered into the network, and therefore have nothing to learn from such simulations. But just because a network's function can be written down using equations, does not mean that its effects are fully understood ahead of time. For example, the NEF has been used to construct chaotic systems, such as the Lorenz strange attractor (Eliasmith, 2005). This *deterministic* system is fully described by a three-dimensional nonlinear dynamical system. However, due to the nature of such chaotic systems, arbitrarily small perturbations in state-space diverge exponentially over time up to the diameter of the attractor (Strogatz, 2015, pp. 328–330). This leads to a fundamental inability to predict the trajectory of any physical instantiation of the system; the time-horizon that can be predicted, within some given tolerance, scales only logarithmically with the precision of the observer.

More to the point, this accusation is theoretically equivalent to suggesting the same of any algorithm or computer program for that matter. Even though *any* computer program can be written down using a finite set of symbolic expressions, and realized as a discrete sequence of steps that dynamically evolve a set of registers—such as the machine of Turing (1938)—does not mean that we know what that program will compute ahead of time on any given input. The assertion that the behaviour of certain programs are in essence "unpredictable" corresponds with the notion of *undecidability* in formal theories of computation; informally, most non-trivial programs need to be run in order to see what they will do with their inputs.

This discussion extends to dynamical systems, where chaos and undecidability are also interwoven (Moore, 1991). For example, suppose we take the linear time-invariant (LTI) system of equation 2.12 and then include saturation within each of the state-variables (such that their absolute magnitudes cannot exceed one). Surprisingly, this is a sufficient route to chaos. The inclusion of this single nonlinearity spawns attractors with properties that are undecidable (Blondel et al., 2001); basic questions about the system's response cannot be resolved in a finite number of steps (Bennett, 1990). This example is of special interest to us since the LIF neuron (equation 2.3) exhibits a similar saturation effect, namely $r(\mathbf{v}) \leq \tau_{\text{ref}}^{-1}$, where the upper-bound is approached as $\mathbf{x}(t) \in \mathbb{R}^q$ extends beyond the representational range established by the encoding parameters selected by Principle 1. It follows that even the most basic spiking instantiation of Principle 3 must, in general, be simulated in order to determine its behaviour whenever the input signal drives the state-space outside the range of neural representation. Notably, this saturation effect is exploited to overload working memory with sequential items in the Spaun model (Eliasmith et al., 2012), which suggests that the model behaviour is chaotic at multiple levels of analysis.

Perhaps even more surprisingly, one does not need to invoke any saturation to observe chaos, at the level of individual spiking neurons, in NEF networks implementing linear dynamics. For example, consider the one-dimensional, autonomous (i.e., no input), integrator:

$$\dot{x}(t) = 0,$$

implemented as an SNN using Principle 3 of the NEF ($A = B = D = 0$, $C = 1$). This is perhaps the simplest recurrent network that one can imagine; a line attractor without input, that is to hold its initial state indefinitely. Nevertheless, as shown in Figure 3.1, the neural states display the tell-tale signs of a strange attractor (cf. Strogatz, 2015, Figure 9.3.5). Specifically, let $\mathbf{v}_1(t), \mathbf{v}_2(t) \in \mathbb{R}^n$ be two voltage vectors (see equation 2.3) from two separate, but deterministic simulations, of the *exact* same network ($n = 10$, $\tau = 5$ ms, $\mathtt{dt} = 0.1$ ms, down-sampled every 20 ms). The first simulation is initialized to the default of $x_1(0) = 0$, while the second is initialized to $x_2(0) = 10^{-15}$. Now let:

$$\begin{aligned}\Delta(t) &= x_1(t) - x_2(t) \\ \delta(t) &= \mathbf{v}_1(t) - \mathbf{v}_2(t),\end{aligned}$$

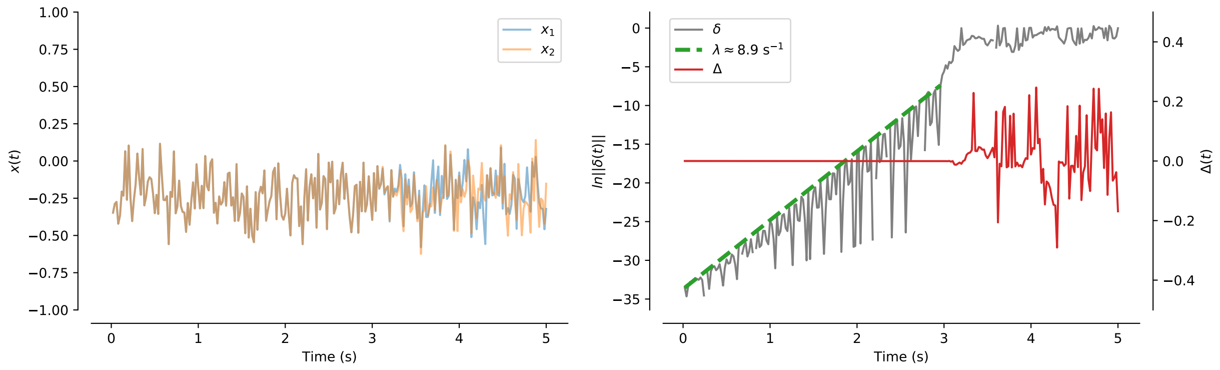


Figure 3.1: An autonomous scalar integrator—the simplest possible recurrent SNN, built using the NEF—exhibiting chaotic neural dynamics. (Left) Two deterministic simulations of the exact same network. One is initialized to $x_1(0) = 0$, and the other to $x_2(0) = 10^{-15}$. Both simulations fall into the same apparent strange attractor, close to zero. (Right) Plotting the difference in representational states (Δ) and neural states (δ) on a logarithmic scale. The neural states diverge exponentially quickly—leaping 30 orders of magnitude in 3 seconds—according to $\|\delta(t)\| \approx \mathcal{O}\left(e^{\lambda t}\right)$. The representational difference Δ is essentially zero until δ hits the diameter of the attractor. It is physically impossible to predict the future state of this system beyond a time-scale of $\mathcal{O}\left(\lambda^{-1}\right) \approx 0.1$ seconds (see text for details).

be the difference in representational state and neural state, respectively. We find that Δ remains bounded above by the accuracy of neural representation, as expected. However, δ diverges exponentially over time, consistent with the neural states being within a strange attractor (the Lyapunov exponent $\lambda \approx 8.9\,\mathrm{s}^{-1}$ gives the exponential rate of divergence); the voltage vectors are tracing out a fractal manifold embedded within an n-dimensional space that cannot be predicted beyond a time-horizon on the order of $\mathcal{O}\left(\lambda^{-1}\right) \approx 0.1$ s. Despite this, the NEF is capable of taming the chaos at the neural level and providing a robust estimate at the population level, due to the correspondence between each apparent strange attractor and a stable point in representational space (Eliasmith and Anderson, 2003, p. 237). In section 3.2.4, we remark that this leads to the NEF being robust to a distinctly different type of error.

At a high level, the NEF may be used to implement arbitrary algorithms encoded as dynamical systems (see section 3.2.3), which implies that it is not exempt from computability theory. Networks can perform interesting tasks via algorithmic manipulations of inputs and states (Choo, 2018). Then, as is the case for any Turing machine, the network must in general be simulated to carry out the computations of said algorithm. At the same time, the NEF provides a framework—akin to having a programming language for some model of computation—to understand the dynamical transformations being performed, examine relationships between structure and function, relate these functions to neurobiological systems, and compile them onto neuromorphic hardware.

3.1.5 Coding by postsynaptic currents

A long-standing debate in neuroscience has traditionally revolved around the question of whether biological neurons transmit information using a "rate code" in which the information is encoded by the firing rates of individual neurons (Adrian, 1928), or a "spike-timing code" in which information is encoded by the precise temporal patterns of spike trains (Rieke and Warland, 1997) or likewise their temporal order in relation to one another (Thorpe and Gautrais, 1998). However, there is historically little consensus between neuroscientists as to what exactly constitutes a rate code (Eliasmith and Anderson, 2003, pp. 89–91).

Gerstner (1999) reviews at least three different ways to define a rate code, and notes that in many important ways they are consistent with that of a timing-based code. Fairhall et al. (2001) models the adaptive dynamics of neurons in the fly visual system and concludes that principles of its code depend on the time-scale of interest. Brette (2015) argues that which code the brain uses is irrelevant, and that we ought to instead consider questions that address the causal role of neural activity. Eliasmith and Anderson (2003) likewise proposes that we should focus on the physical instantiation, and functional consequences therein, of any given approach to neural coding, rather than resorting to semantic labels that are ultimately irrelevant.

Nevertheless, many have mislabelled the NEF as employing a rate-coding scheme [personal communication], including Lagorce and Benosman (2015) and Frady and Sommer (2019) for two recent examples – we refer to these two below, to illustrate why this accusation constitutes a damaging use of semantics. Specifically, this mischaracterization has lead to the misapplication of many criticisms (Gautrais and Thorpe, 1998) that stem from the original proposal of Adrian (1928), namely the need to average spike rates over long windows of time. We find this important to clarify because it leads to imprudent conclusions or "myths" about the NEF such as those claimed by Lagorce and Benosman (2015) and Frady and Sommer (2019):

1. The NEF does not exhibit precise sequences of action-potentials.
2. The NEF does not support high-speed neural computation.
3. The NEF does not display rhythmic activity.
4. The NEF requires very large numbers of neurons to compute simple functions.

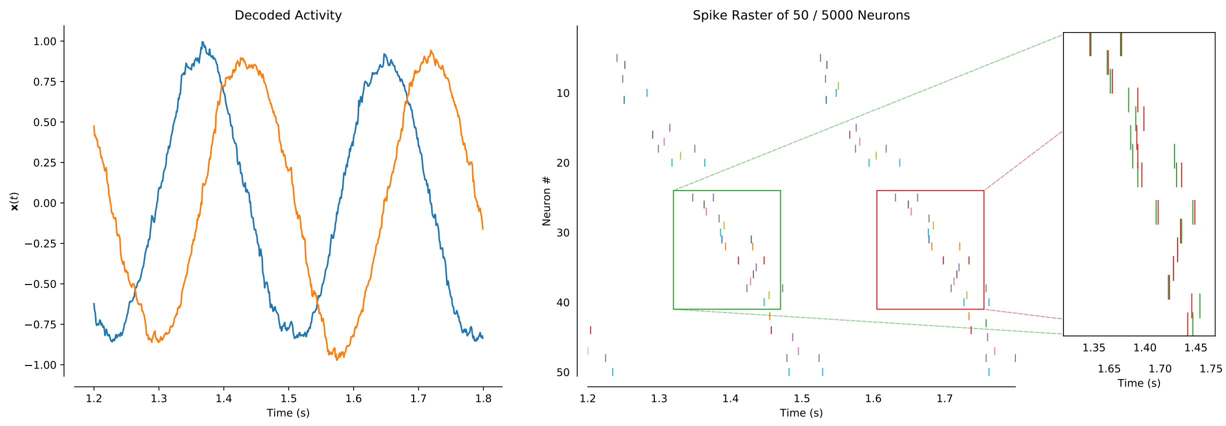


Figure 3.2: Demonstrating the irrelevance of identifying a “spike-time code” versus a “rate code” in the context of a standard NEF-optimized network. We define this as a “postsynaptic current code” instead. (Left) An ensemble of LIF neurons are trained to oscillate at approximately 3.5 Hz. (Right) Spike rasters of 50 randomly selected neurons, ordered by their encoding vector’s polar angle. Each neuron spikes only 0, 1, or 2 times per oscillation, and at a precise moment in time—on the order of milliseconds (see inset)—before remaining silent for another couple hundred milliseconds. Thus, the precise spike-timing of each individual neuron reliably conveys information about the state-space of the oscillation, despite never explicitly incorporating such a requirement about timing into the training procedure.

We now challenge each claim in turn. For the first three, we refer to the same simulation depicted in Figure 3.2. This simulation applies the NEF, as described in section 2.2, to the case of an autonomous two-dimensional oscillator ($n = 5{,}000$, $\tau = 0.1$ s, $\mathtt{dt} = 1$ ms). The encoding parameters are randomly tiled across the two-dimensional state-space such that each neuron only responds to 25% of the state’s projection onto its encoder (i.e., uniform $[0.5, 1)$ intercepts), and each neuron *would* fire at a rate of 20–40 Hz if encoding a constant state with maximal similarity to its encoder. As we explain, the firing statistics are not at all characterized by 20–40 Hz spike-trains. We omit the first 1.2 seconds of simulation to avoid initial transients.

1. The NEF can exhibit repeatedly precise sequences of action-potentials. See Figure 3.2 inset. When comparing the spike-trains between two separate oscillations at the same phase, not only is the order of spiking consistent, as in rank order coding (Thorpe and Gautrais, 1998), but also the precise spike-timing (± a couple milliseconds).

2. The NEF readily supports high-speed neural information processing. In our example, neurons respond quickly to encode the rapidly-fluctuating oscillatory state, and do so without any system-level "delay" or undesired filtering.[4] As will be explained in section 3.2.1, the precision of a feed-forward network scales as $\mathcal{O}\left(\tau\sqrt{n}\right)$. Thus, one is free to set the synaptic time-constant arbitrarily small, so long as the number of neurons are increased in proportion. We have verified this both theoretically and numerically in section 5.2.2. This enables arbitrarily fast transmission of information throughout the network assuming the criteria of Theorem 3.2.1 is met (also see Figure 5.4). Yet, even when constrained to longer time-constants, section 6.2.5 demonstrates a novel deep NEF network that is capable of *instantaneously* propagating low-frequency stimuli through 8 layers of synaptic filters ($\tau = 0.1$ s).

3. The NEF examples that invoke Principle 3 all display rhythmic activity. The NEF was designed as a toolkit for modelling dynamic rhythmic activity (Eliasmith and Anderson, 1999) such as the central pattern generator driving lamprey locomotion (Eliasmith and Anderson, 2000). Indeed, Figure 3.2 clearly displays rhythmic activity at both the population level (see left) and the activity level (see right). The properties of these rhythms can be understood as arising from the dynamics of the postsynaptic currents, in response to the neural encoding of the state-vector governed by some underlying set of differential equations.

4. The NEF can compute difficult functions with any number of neurons. See Figure 3.1 for an example of ten spiking LIF neurons implementing a line attractor ($\tau = 5$ ms), or section 6.2.3 for a six neuron cell that outperforms LSTM cells. No matter how complex the function, the mandate of the NEF is to leverage its neuron models as a basis for that function. Sometimes this can be done with sufficient accuracy using a single neuron, and in other cases one might need a few million or even more. In general, the feed-forward precision scales as $\mathcal{O}\left(\tau\sqrt{n}\right)$ while the dynamical precision scales as $\mathcal{O}\left(\theta\sqrt{n}\right)$, where θ is the time-constant of the dynamical system (under fairly weak assumptions; see section 3.2.1). But one cannot consider the questions of resource usage and functional precision within a vacuum. One must resolve such questions with respect to the device-level models of the physical hardware implementation as well as the intended target application. Some advances in this direction are underway (Schwemmer et al., 2015; Thalmeier et al., 2016). But many other considerations such as weight factorization, mechanistic constraints, and the energy consumed by different synaptic or neural operations, may all play a role. If one is interested in constructing functioning dynamical SNNs using

[4]The postsynaptic filters are leveraged to participate in the required computation (see Principle 3; section 2.2.3). There is no unwanted phase-shift.

neuromorphic hardware, we suggest that one considers all such factors in the context of the target application before labelling an approach as being unequivocally inferior.

The confusion surrounding rate coding in the NEF has essentially risen from the adoption of equation 2.10. However, as we have said, this merely reformulates the optimization procedure to be more efficient, without sacrificing correctness as proven in section 3.2.1. We remark that, in Figure 3.2, the firing statistics are completely unlike their rate model counterparts, despite the target postsynaptic currents and overall system dynamics remaining the same. That is, each neuron only fires at an average rate of 3 Hz across the simulation, much slower than the 20–40 Hz rate that they would fire at for constant inputs. Likewise, the postsynaptic impulse response that results from a single spike, decays to a factor of $e^{-\frac{10}{3}} \approx 3.5\%$, before *that same* neuron triggers another spike, on average. In general neither the average rates, inter-spike intervals, nor spike times tell the entire story.

But then how does equation 2.10 hold if each neuron is not spiking at its intended rate? Our proposal to resolve this seemingly paradoxical situation is to first establish a new label: "post-synaptic current code." This code does not care about the spike-rates of individual neurons; it is only sensitive to how well the weighted and synaptically-filtered spikes, *when pooled across the entire population*, approximate some desired set of postsynaptic currents (corresponding to an affine transformation of the required state-vector; Tripp and Eliasmith, 2006). This is summarized by taking the encoding equation 2.2 and decoding equation 2.6 which fold into the weight equation 2.11 – and combining them in a similar manner to Stöckel et al. (2018):

$$\alpha_i \langle \mathbf{e}_i, \mathbf{x}(t) \rangle + \beta_i \approx \sum_{j=1}^{n} \sum_{m} \omega_{ij} h\left(t - t_{j,m}\right). \tag{3.1}$$

In plain words, the represented state-vector is linearly projected onto the postsynaptic current of each neuron. Nothing needs to change about our exposition of the NEF in order to accommodate this viewpoint of how it codes information. How this works in light of equation 2.10 requires careful proof (Theorem 3.2.1), and the subtleties surrounding why this can matter are challenging to grasp. But as our example illustrates, the NEF cannot be adhering to any single definition of rate or timing code. Rather, it is representing desired transformations by mapping latent state-variables onto postsynaptic currents. In the following sections, we tease apart the NEF's coding principles, sources of error, and scaling properties.

If one is still unconvinced, then, as mentioned in section 2.2.4 when comparing the NEF to RC and FORCE, one may forgo equation 2.10 and perform the optimization directly in the spiking time-domain (Friedl et al., 2016; Duggins, 2017), or even apply backpropagation through time (Rasmussen, 2018). But the fact of the matter is that this becomes unnecessary (and inefficient) for a large class of interesting systems and models that we wish to explore.

3.1.6 Energy-minimization via low-frequency representation

We define a *low-frequency representation* as one in which the represented vector, $\mathbf{x}(t)$, carries its information within frequencies on the order of the cutoff frequency of the synaptic filter. That is, $(2\pi\tau)^{-1}$ Hz, where τ is the synaptic time-constant of equation 2.13 in seconds.[5] Such representations are known to be ubiquitous in biological systems (Pulvermüller et al., 1997; Singer, 1999; Szendro et al., 2001). The NEF provides a basic theoretical explanation for this observation from the perspective of energy-minimization.

As discussed in the previous section, and described by equation 3.1, the state-vector $\mathbf{x}(t)$ is projected onto postsynaptic currents. To analyze the frequency content of this code, we apply the Laplace transform to both sides of equation 3.1, and substitute the synapse model from equation 2.13:

$$\begin{aligned}\alpha_i \left\langle \mathbf{e}_i, \mathcal{L}\left\{\mathbf{x}(t)\right\}(s)\right\rangle + \beta_i &\approx \sum_{j=1}^{n} \sum_{m} \omega_{ij} \mathcal{L}\left\{h\left(t - t_{j,m}\right)\right\}(s) \\ &= \sum_{j=1}^{n} \sum_{m} \omega_{ij} \frac{e^{-t_{j,m}s}}{\tau s + 1} \\ &= (\tau s + 1)^{-1} \underbrace{\sum_{j=1}^{n} \omega_{ij} \mathcal{L}\left\{a_j(t)\right\}(s)}_{G_i(s)} .\end{aligned} \tag{3.2}$$

$G_i(s)$ is defined as the Laplace transform of the weighted combination of presynaptic (unfiltered) spike-trains. The magnitude of $|G(2\pi i f)|$, evaluated at real frequencies f, is the unitless (dimensionless) measure of energy required to drive all of the synapses of the i^{th} postsynaptic neuron, in order to generate such a PSC. Rearranging equation 3.2 yields:

$$G_i(s) \approx (\tau s + 1)\left(\alpha_i \left\langle \mathbf{e}_i, \mathcal{L}\left\{\mathbf{x}(t)\right\}(s)\right\rangle + \beta_i\right). \tag{3.3}$$

The transfer function $(\tau s + 1)$ is a highpass filter that linearly amplifies the frequencies of its input. Specifically, our dimensional analysis identifies the following asymptotic relationship:

$$|G_i(s)| \approx \mathcal{O}\left(|\tau s| \, \|\mathcal{L}\left\{\mathbf{x}(t)\right\}(s)\|_2\right), \tag{3.4}$$

where the Big $\mathcal{O}$ constants involve the length of the encoder (typically unit-length) and the gains and biases (unitless).[6]

[5]For example, a typical time-constant of $\tau = 5$ ms would imply that most information is present within a frequency band of < 32 Hz.

[6]If a physical quantity is normalized to some constant range then it becomes unitless.

From equation 3.4 it directly follows that, in order to represent $\mathbf{x}(t)$ with an absolute power of one (unitless) at a frequency of f, via the postsynaptic current code (i.e., linear projection onto each PSC), the relative amount of energy needed to drive all of the synapses of the postsynaptic neuron scales as $\mathcal{O}\left(2\pi\tau f\right)$. Therefore, for a *fixed* energy budget, the frequencies that can be successfully transmitted using this code scale by the inverse law (which happens to be the cutoff frequency of the filter):

$$\mathcal{O}\left((2\pi\tau)^{-1}\right). \tag{3.5}$$

In this context, the role of neural coding is to sparsify signals across time while allowing for their accurate reconstruction, consistent with notions from compressed-sensing, sparse coding, and sigma-delta modulation frameworks (Coulter et al., 2010; Chklovskii and Soudry, 2012; Yoon, 2017). The fidelity with which this can be achieved depends on factors such as the total energy available to drive all of the synapses and the useful frequencies within the signal being reconstructed. The NEF provides a way to formalize such relationships in the context of large-scale functional models, thus making predictions relevant to biological modellers, and establishing trade-offs relevant to neuromorphic engineers. We provide additional examples of such trade-offs in the following sections.

3.2 Suitability for Neuromorphic Hardware

In the prospectus of Boahen (2017), the NEF is recognized as a suitable framework for neuromorphic engineering. Many of these reasons have already been mentioned in earlier discussions, and have been validated by recent deployments of the NEF on Braindrop (Neckar et al., 2019), Loihi (Blouw et al., 2018), and TrueNorth (Fischl et al., 2018). This section aims to theoretically formalize the criteria that we currently view as being of utmost importance for neuromorphics – specifically, how the NEF is correct, scalable, complete, robust, extensible, and thus ultimately, *useful*.

3.2.1 Correctness

In the original formulation of the NEF (Eliasmith and Anderson, 2003), there exists three sources of error in the neural representation:

1. "static distortion" arising from the use of decoders; that is, the linear combination of nonlinear basis functions to approximate some desired transformation,

2. "spiking noise" arising from the use of discontinuous spikes; that is, the substitution of a continuous non-spiking model with a spiking model that produces variable PSCs, and

3. "state discrepancy" arising from the substitution of a stateless model with a stateful model.

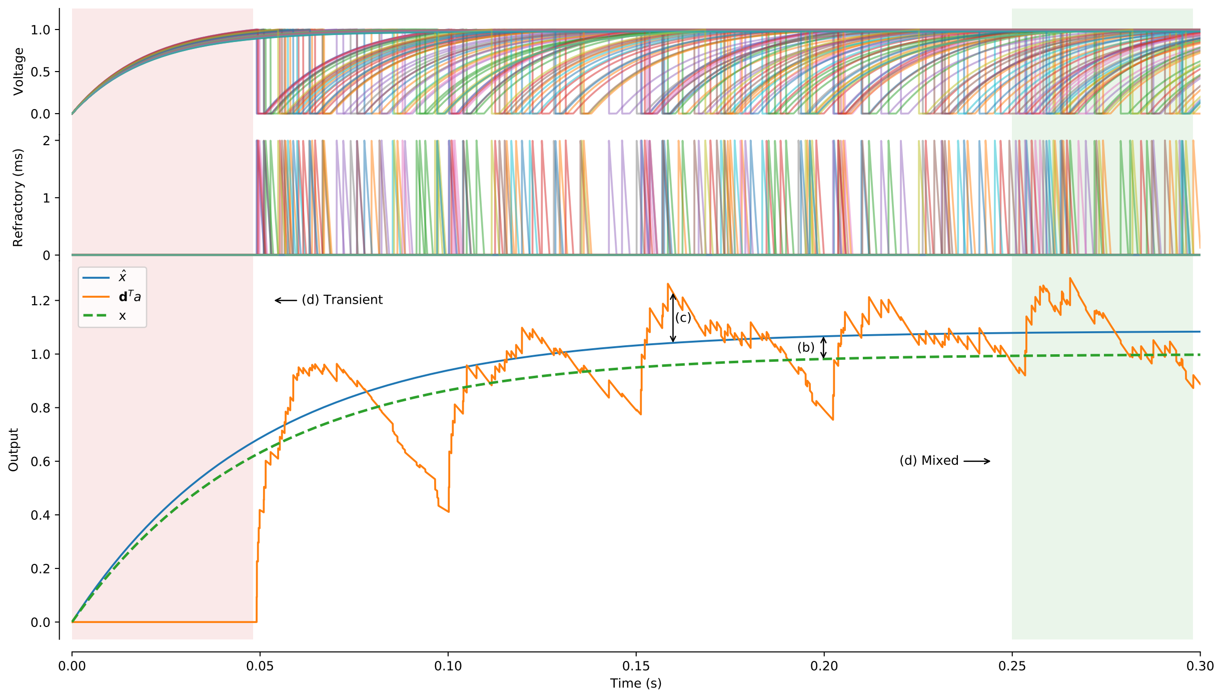
The first two sources of error have already been studied extensively throughout Eliasmith and Anderson (2003) and Bekolay et al. (2014). We review these findings below. However, as far as we know, the third has not received formal attention. A proof is found in Eliasmith and Anderson (2003, pp. 132–136, appendix C.1) that addresses the same issue from the perspective of including noise within the neuron model, and the contents herein may be viewed as an updated treatment of that study.

Figure 3.3 illustrates all three types of error within a single feed-forward simulation. The network is a one-dimensional communication channel ($f(x) = x$, $n = 100$, $\tau = 50\,\text{ms}$, $\texttt{dt} = 10^{-5}\,\text{s}$, $\tau_{\text{RC}} = 20\,\text{ms}$, $\tau_{\text{ref}} = 2\,\text{ms}$) with maximum firing rates of 10–20 Hz and intercepts uniformly distributed across $[-1, -0.5)$. Low firing rates are needed to expose the third source of error, which is normally masked by the other two at high firing rates for reasons that will become clear. The population, initially at rest, encodes a constant input signal of $u(t) = 1$, that is directly mapped onto the PSCs without additional filtering. We also duplicate the population, use the same decoders, but replace its neuron model with the corresponding continuous-valued rate model given by equation 2.8. For notational simplicity, $\mathbf{d}^\top a$ refers to the filtered and decoded spike-trains (i.e., the spiking solution), $\hat{x}$ denotes the filtered and decoded rate model (i.e., the non-spiking solution), and $x = u * h$ is the ideally filtered input signal (i.e., the desired output). We now describe each source of error in greater detail.

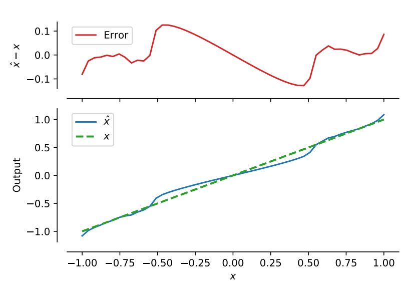
Static distortion

The optimization problem described by equation 2.9 can be understood as fitting n non-orthogonal static basis functions, r_i, to some desired static function, f (Broomhead and Lowe, 1988). Assuming relatively weak conditions on each r_i, namely monotonicity and continuity, any continuous f may be approximated arbitrarily well by a linear combination of these basis functions (Hornik et al., 1989), as in equation 2.6.

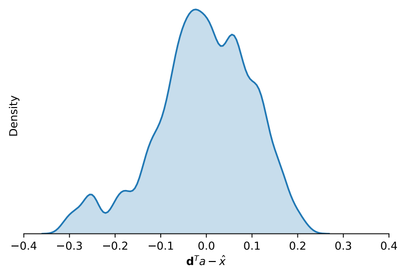
Specifically, for a given f, a linear increase in the number of basis functions will reduce the root-mean-squared error (RMSE) of the fit by the same factor. This particular RMSE is named *static distortion*, since it corresponds to the difference between the two static (i.e., non-



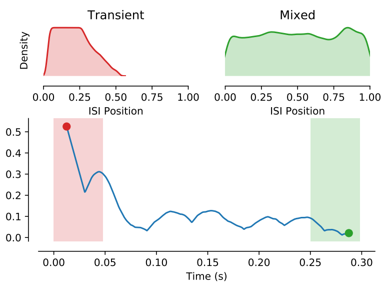
(a) Example Network Simulation



(b) Static Distortion



(c) Spiking Noise ($t > 0.15$ s)



(d) State Discrepancy

Figure 3.3: Visualizing the three sources of error in a standard NEF network. (a) A single simulation with arrows pointing to each type of error (b, c, d). (b) Static distortion describes the difference between the linear combination of rate models and the desired static function. (c) Spiking noise describes the difference between the spiking model and rate model. (d) State discrepancy provides an instantaneous statistic capturing the difference between the spiking noise and a true Gaussian distribution (p-values $< 10^{-6}$). See text for details.

temporal) functions. We commonly refer to the RMSE as simply the "error" throughout,[7] and define "precision" to be the reciprocal of error. Thus we say that the precision of a non-spiking network scales as $\mathcal{O}(n)$.

Figure 3.3 (b) visualizes the static distortion in two different ways. The bottom plot, x versus $\hat{x}$, compares the ideal against the approximation, and includes the dashed line $x = x$ for reference. The upper plot, x versus $\hat{x} - x$, provides a closer look to demonstrate that the error is continuous and with a mean close to zero. The root-mean-squared value of the latter curve is the static distortion.

Spiking noise

By separating out the static distortion and looking at the difference $\mathbf{d}^\top a - \hat{x}$, we can isolate the role of equation 2.10 in the approximation error. This is the difference between the spiking and non-spiking estimates, using the same set of basis functions. We refer to this as the *spiking noise*, since it corresponds to the variability in the estimate that is introduced by filtered spikes.

The variability in spike-trains across the population leads to an additional error term that scales as $\left(\tau\sqrt{n}\right)^{-1}$. The τ^{-1} term comes from the variance of the filter's impulse response – supposing equation 2.4 is applied to both the decoded spikes and the encoded input. The $\sqrt{n}^{-1}$ term follows from the convergence rate of the central limit theorem (CLT; Berry, 1941; Esseen, 1942), and assuming the spiking noise is Gaussian and independent. Under these assumptions, the spike noise asymptotically dominates the static distortion, and the SNN's feed-forward precision scales as $\mathcal{O}\left(\tau\sqrt{n}\right)$. However, in the context of Principle 3, the transformations of the signals are scaled by $\tau\theta^{-1}$, where θ is the time-constant of integration for the desired dynamical system, which results in a cancellation and the recurrent SNN having precision that scales as $\mathcal{O}\left(\theta\sqrt{n}\right)$.

Figure 3.3 (c) visualizes a kernel density estimate (KDE; Waskom et al., 2015) of this error ($t > 0.15\,\mathrm{s}$). A Kolmogorov-Smirnov (KS) test of $\mathbf{d}^\top a - \hat{x}$, versus a zero-mean Gaussian distribution with equal variance, produces a KS-statistic of ≈ 0.034 (p-value $< 10^{-6}$). The KS-statistic is defined as the maximal discrepancy between the empirical cumulative density function (CDF) and the model CDF (i.e., the supremum of their absolute difference; Massey Jr, 1951). In this case, the distribution of spike noise is at most 3.4% different from a true Gaussian distribution. This justifies the usage of the white noise term η in equation 2.9, since it captures $\approx 96.6\%$ of the statistical variance between the filtered and decoded spiking and non-spiking models. But

[7] Eliasmith and Anderson (2003) use "error" to refer to mean-squared error (MSE), rather than the RMSE. This may lead to confusion when comparing across sources.

why is there a difference? In theory it should approach 0 for sufficiently many samples, due to the CLT, but unfortunately it does not; there is a remaining systematic bias that never scales away (also see Figure 5.5). It turns out that our assumption of independent Gaussian noise was not entirely correct – an oversight that we will rectify at the end of this section. But first, this motivates the discovery of a third and final source of error, that we now describe and analyze in detail.

State discrepancy

Intuitively, unbiased variability in spike-trains is achieved by all of the neurons being independently and uniformly *ready to spike*. Looking at the "transient" portion ($t < 0.05\,\mathrm{s}$) of Figure 3.3 (a), we see that all of the neurons are initially at rest, and thus physically cannot spike in time to keep up with the target. This results in a period of time in which the error is quite large. The error is overall the smallest towards the end of the simulation, at which point we say the neural states have "mixed" together. We formalize these ideas by appealing to the inter-spike interval (ISI) in a manner that will turn out to be quite important.

We define a neuron's relative "ISI position" (ISIP), g_i, to be its normalized position within the inter-spike interval that corresponds to the instantaneous firing rate of the ideal rate model. For this, we fix our analysis at *any given moment in time* and determine the next spike time. To give an example, consider the LIF neuron's current state (although this same approach extends more generally to other neuron models) to have a voltage of v_i, remaining refractory period of $0 \le \rho_i \le \tau_{\mathrm{ref}}$, input current of s_i, and a corresponding instantaneous target rate of r_i. In the same way that we derived the rate model of the LIF neuron (see derivation leading to equation 2.8), we may derive the neuron's ISIP according to how far along it is within the ideal ISI normalized by its length, r_i^{-1}:[8]

$$\begin{aligned} g_i(v_i, \rho_i) &= 1 - \left(\rho_i + \tau_{\mathrm{RC}} \ln\left(1 + \frac{1 - v_i}{s_i - 1}\right)\right) r_i \\ &= 1 - \frac{\rho_i + \tau_{\mathrm{RC}} \ln\left(1 + \frac{1 - v_i}{s_i - 1}\right)}{\tau_{\mathrm{ref}} + \tau_{\mathrm{RC}} \ln\left(1 + \frac{1}{s_i - 1}\right)}. \end{aligned} \tag{3.6}$$

For example, $g_i(0, \tau_{\mathrm{ref}}) = 0$, since the neuron is at the very beginning of its ISI. Likewise, $g_i(1, 0) = 1$, as it has just spiked.

[8]If $r_i = 0$ then the neuron will not be active (and does not need to be), and so we can simply ignore it; the discussion, analysis, and results do not change.

Definition 3.2.1. The *state discrepancy*, at some chosen moment in time, is the KS-statistic between the empirical ISIP distribution across the population, $\{g_i : i = 1 \ldots n\}$, and a true uniform distribution, $\mathcal{U}[0,1]$.

In Figure 3.3 (d), we measure this value empirically across all of neurons, and at each time-step in the simulation, using a centered window of width 25 ms to smooth the estimate. We see that the magnitude of the KS-statistic corresponds very closely to the error between the ideal and the spiking solution. Although here we are looking at just a constant input for ease of visualization, we find that this statistic correlates highly (Pearson correlation coefficient $R = 0.90$, p-value $< 10^{-6}$) with the spiking noise across a wide range of input frequencies (see Figure 3.4). This is surprising since the definition of state discrepancy does not invoke any usage of decoders, spikes, or filters in its construction; it merely describes how different the distribution of relative ISI positions are from that of a uniform distribution.

We now prove that our definition of state discrepancy is indeed the correct one, and in doing so provide a sufficient criteria for scaling away this source of error. Specifically, this remaining source of error—state discrepancy—is tamed by minimizing the difference (KS-statistic) between a uniform distribution and the actual neuron's "readiness to spike," as characterized by its ISIP (e.g., equation 3.6 for the case of a LIF model).

Theorem 3.2.1 (Scaling of feed-forward precision in the Neural Engineering Framework)**.** *Let* $\mu(t) = \sum_{i=1}^{n} (a_i * h)(t)\mathbf{d}_i^{\mathbf{f}} - \sum_{i=1}^{n} (r_i(\mathbf{x}) * h)(t)\mathbf{d}_i^{\mathbf{f}}$ *be the spiking noise. If the ISIPs, g_i, are uniformly and independently distributed, then:*

1. $\mathbb{E}\left[\mu(t)\right] = 0$*; hence the expected spike noise is exactly zero at all times (no systematic bias),*
2. $\sqrt{\mathbb{V}\left[\mu(t)\right]} = \mathcal{O}\left(\left(\tau\sqrt{n}\right)^{-1}\right)$ *is the standard error, or root-mean-squared error; hence the precision scales as* $\mathcal{O}\left(\tau\sqrt{n}\right)$ *at all times.*

We prove each statement, in order, after a few remarks. We prove something slightly more general, and without invoking asymptotics until the very end. We also do so without assuming any particular neuron model – only that it has a well-defined ISIP. We shift our frame of reference, without loss of generality, such that we are currently observing $t = 0$. We then forecast the system out into the future, assuming the representation is held constant. But this does not make any assertions about the causal dynamics of the representation. This is because the statements in this theorem are only interested in the distribution of a statistic at arbitrarily small values of $t \to 0^+$ with respect to the shifted time frame. Hence, our work provides a more general result than what is needed. In order to give a concrete result, we suppose the synapse model is equation 2.4, but this is not a hard requirement. Likewise, we initialize the synapses to 0, without loss of generality by linearity. To remove boilerplate in the notation we substitute $r_i \leftarrow r_i(\mathbf{x}(t))$. We have verified each of the following equations with numerical experiments (not shown).

Preliminaries

Let $\tilde{y}_i(t;t_i) = (a_i * h)(t)$ be the fast-forward simulation of the i^{th} neuron up to time $t \geq 0$, supposing $t_i = \left(1 - g_i\right) r_i^{-1}$ is the next spike-time ($0 \leq t_i \leq r_i^{-1}$) by our definition of the ISIP. Let T_i be the random variable whose realizations are t_i. Hence, $T_i \sim \mathcal{U}[0, r_i^{-1}]$, with a probability density function (PDF) and cumulative density function (CDF) described by:

$$f_{T_i}(t_i) = r_i \quad \Longleftrightarrow \quad P(T_i \leq t_i) = t_i r_i. \tag{3.7}$$

By enumerating over the spike-times, we have:

$$\begin{aligned}
\tilde{y}_i(t;t_i) &= \sum_{0 \leq j \leq r_i \cdot (t - t_i)} (\delta_{j/r_i + t_i} * h)(t) \\
&= \sum_{0 \leq j \leq r_i \cdot (t - t_i)} \frac{1}{\tau} e^{-\frac{t - (j/r_i + t_i)}{\tau}} \\
&= \sum_{0 \leq j \leq r_i \cdot (t - t_i)} \frac{1}{\tau} \Bigg(\underbrace{e^{-(r_i \tau)^{-1}}}_{c_i} \Bigg)^{r_i \cdot (t - t_i) - j} ,
\end{aligned}$$

where $0 < c_i < 1$ is an important constant that we remark is dimensionless (i.e., unitless), and also time-invariant (i.e., depends only on r_i and τ). We then re-arrange as follows:

$$\begin{aligned}
\Longrightarrow \quad \tilde{y}_i(t;t_i) &= \frac{c_i^{r_i \cdot (t - t_i) + 1}}{\tau} \sum_{1 \leq j \leq r_i \cdot (t - t_i) + 1} \left(c_i^{-1}\right)^j \quad \text{(note: altered index)} \\
&= \frac{c_i^{r_i \cdot (t - t_i) + 1}}{\tau} \left(\frac{1}{c_i}\right) \frac{\left(\frac{1}{c_i}\right)^{\lfloor r_i \cdot (t - t_i) + 1 \rfloor} - 1}{\frac{1}{c_i} - 1} \quad \text{(geometric series)} \\
&= \frac{c_i^{e_i(t;t_i)} - c_i^{r_i \cdot (t - t_i) + 1}}{\tau(1 - c_i)}
\end{aligned} \tag{3.8}$$

where:

$$\begin{aligned}
e_i(t;t_i) &= r_i \cdot (t - t_i) - \lfloor r_i \cdot (t - t_i) \rfloor \\
0 &\leq e_i(t;t_i) < 1,
\end{aligned} \tag{3.9}$$

is a type of truncation error. Equation 3.8 provides a closed-form solution for $\tilde{y}_i$. In addition, the lower/upper bounds of $e_i(t;t_i)$ supply its upper- and lower-envelopes, respectively:

$$\frac{c_i - c_i^{r_i\cdot(t-t_i)+1}}{\tau(1-c_i)} < \tilde{y}_i(t;t_i) \leq \frac{1-c_i^{r_i\cdot(t-t_i)+1}}{\tau(1-c_i)}.$$

Now, the analysis proceeds in the following manner. We pick our t, and then sample our t_i. This determines $e_i(t;t_i)$ for each neuron, by a cyclic rotation $r_i t_i$ (modulo 1) from equation 3.9. We then define a new random variable $\tilde{Y}_i(t) = \tilde{y}_i(t;T_i)$, which one can understand as being distributed by "mapping" the random variable T_i through the function $\tilde{y}_i$ for some chosen t. Realizations of this random variable correspond directly to $(a_i * h)(t)$ by construction.

Proof of zero-mean error

We apply the law of the unconscious statistician (LOTUS) to $\tilde{Y}_i(t)$:

$$\begin{aligned}
\mathbb{E}\left[\tilde{Y}_i(t)\right] &= \int_D \tilde{y}_i(t;t_i) f_{T_i}(t_i)\,dt_i, \quad \text{where } D \text{ is the domain of } T_i \\
&= \int_D \frac{c_i^{e_i(t;t_i)} - c_i^{r_i\cdot(t-t_i)+1}}{\tau(1-c_i)} f_{T_i}(t_i)\,dt_i, \quad \text{by equation 3.8} \\
&= \frac{1}{\tau(1-c_i)}\left(\int_0^1 c_i^{e_i(t;t_i)}\,de_i(t;t_i) - \int_0^{r_i^{-1}} c_i^{r_i\cdot(t-t_i)+1}(r_i)\,dt_i\right) \\
&= \frac{1}{\tau(1-c_i)}\left((-r_i\tau)e^{-e_i(t;t_i)/(r_i\tau)}\Big|_{e_i(t;t_i)=0}^{1} - (r_i\tau)c_i^{r_i\cdot(t-t_i)+1}\Big|_{t_i=0}^{r_i^{-1}}\right) \\
&= \frac{r_i}{(1-c_i)}\left((1-c_i) - \left(c_i^{r_i t} - c_i^{r_i t+1}\right)\right) \\
&= r_i\cdot(1-e^{-t/\tau}).
\end{aligned} \tag{3.10}$$

Therefore, by linearity of expectation, the expected error at time t is:

$$\begin{aligned}
\mathbb{E}\left[\mu(t)\right] &= \mathbb{E}\left[\sum_{i=1}^{n}(a_i * h)(t)\mathbf{d}_i^{\mathbf{f}} - \sum_{i=1}^{n}(r_i * h)(t)\mathbf{d}_i^{\mathbf{f}}\right] \\
&= \sum_{i=1}^{n} r_i\cdot(1-e^{-t/\tau})\mathbf{d}_i^{\mathbf{f}} - \sum_{i=1}^{n}(r_i * h)(t)\mathbf{d}_i^{\mathbf{f}} \\
&= 0.
\end{aligned} \tag{3.11}$$

This completes our proof of the first statement. □

Note that we have not needed to use our assumption about the independence of samples. However, to obtain a closed-form solution for the variance, we will need to assume that g_i, and thus t_i, are independently sampled (i.e., the voltage vectors and refractory periods are not coupled with one another across time), as is the case for typical NEF networks. Boerlin et al. (2013) have shown that if the neural states do have a mechanism for collaborating in a global manner, then they can synchronize their activity; this corresponds to a means of *dependent* sampling that reduces the following variance.

Proof of spiking variance

To compute $\mathbb{V}\left[\tilde{Y}_i(t)\right] = \mathbb{E}\left[\tilde{Y}_i(t)^2\right] - \mathbb{E}\left[\tilde{Y}_i(t)\right]^2$ we rewrite the second moment:

$$\begin{aligned}\mathbb{E}\left[\tilde{Y}_i(t)^2\right] &= \mathbb{E}\left[\left(\frac{c_i^{e_i(t;T_i)} - c_i^{r_i\cdot(t-T_i)+1}}{\tau(1-c_i)}\right)^2\right] \\ &= \left(\frac{1}{\tau(1-c_i)}\right)^2 \left(\mathbb{E}\left[\left(c_i^{e_i(t;T_i)}\right)^2\right] - 2\mathbb{E}\left[c_i^{e_i(t;T_i)+r_i\cdot(t-T_i)+1}\right] + \mathbb{E}\left[\left(c_i^{r_i\cdot(t-T_i)+1}\right)^2\right]\right)\end{aligned}$$

and then separately determine each expectation. The PDFs of $c_i^{e_i(t;T_i)}$ and $c_i^{r_i\cdot(t-T_i)+1}$ are both $r_i\tau/x$ over their respective domains. This can be seen directly:

$$\begin{aligned}\frac{\partial}{\partial x} P\left(c_i^{e_i(t;T_i)} \le x\right) &= \frac{\partial}{\partial x} P\left(e_i(t;T_i) \ge -r_i\tau \ln x\right) \\ &= \frac{\partial}{\partial x}\left(1 - (-r_i\tau \ln x)\right) \\ &= r_i\tau/x.\end{aligned}$$

and the proof for $c_i^{r_i\cdot(t-T_i)+1}$ is analogous. As an aside, these two facts can be used to give a slightly different derivation for equation 3.10. Applying LOTUS to each of these random variables gives:

$$\begin{aligned}\mathbb{E}\left[\left(c_i^{e_i(t;T_i)}\right)^2\right] &= \int_{c_i}^{1} \frac{r_i\tau}{x} x^2\, dx = \frac{r_i\tau x^2}{2}\bigg|_{x=c_i}^{1} = \frac{r_i\tau\left(1-c_i^2\right)}{2} \\ \mathbb{E}\left[\left(c_i^{r_i\cdot(t-T_i)+1}\right)^2\right] &= \int_{c_i^{r_i t+1}}^{c_i^{r_i t}} \frac{r_i\tau}{x} x^2\, dx = \frac{r_i\tau x^2}{2}\bigg|_{x=c_i^{r_i t+1}}^{c_i^{r_i t}} = \frac{r_i\tau\left(1-c_i^2\right)c_i^{2r_i t}}{2}.\end{aligned}$$

To compute the remaining expectation, we require further insight into $e_i(t;T_i) = r_i\cdot(t-t_i) - \lfloor r_i\cdot(t-t_i)\rfloor$. Define $k_i(t) = \lfloor r_i t\rfloor$ as the spike index of t. Recall that t is fixed in this discussion.

The congruence relation $t_i = t - k_i(t)/r_i \equiv t \pmod{r_i^{-1}}$ is also a discontinuity in $e_i(t;t_i)$ with respect to t_i; that is when the spike becomes aligned with t in the current interval. Namely, if $0 \le t_i \le t - k_i(t)/r_i$, then $\lfloor r_i \cdot (t - t_i) + 1 \rfloor = \lfloor k_i(t) + 1 \rfloor = k_i(t) + 1$. Likewise, if $t - k_i(t)/r_i < t_i \le r_i^{-1}$, then $\lfloor r_i \cdot (t - t_i) + 1 \rfloor = k_i(t)$. We now apply LOTUS once more, along with the above facts to split up the integral.

$$
\begin{aligned}
\mathbb{E}\left[c_i^{e_i(t;T_i)+r_i\cdot(t-T_i)+1}\right] &= \int_0^{r_i^{-1}} c_i^{2(r_i\cdot(t-t_i)+1)-\lfloor r_i\cdot(t-t_i)+1\rfloor}(r_i)\,dt_i \\
&= r_i\left(\int_0^{t-k_i(t)/r_i} c_i^{(2r_it-k_i(t)+1)-(2r_it_i)}\,dt_i + \int_{t-k_i(t)/r_i}^{r_i^{-1}} c_i^{(2r_it-k_i(t)+2)-(2r_it_i)}\,dt_i\right) \\
&= r_ic_i^{2r_it-k_i(t)+1}(\tau/2)\left(e^{2t_i/\tau}\Big|_{t_i=0}^{t-k_i(t)/r_i} + c_ie^{2t_i/\tau}\Big|_{t_i=t-k_i(t)/r_i}^{r_i^{-1}}\right) \\
&= r_ic_i^{2r_it-k_i(t)+1}(\tau/2)\left(c_i^{-2r_i\cdot(t-k_i(t)/r_i)} - 1 + c_i^{-1} - c_i^{-2r_i\cdot(t-k_i(t)/r_i)+1}\right) \\
&= r_i\tau(1-c_i)\left(c_i^{k_i(t)+1} + c_i^{2r_it-k_i(t)}\right)/2
\end{aligned}
$$

Substituting all of these expectations into the initial formula yields:

$$
\mathbb{V}\left[\tilde{Y}_i(t)\right] = r_i\left(\frac{\left(1-c_i^2\right)\left(1+c_i^{2r_it}\right)}{2\tau\left(1-c_i\right)^2} - \frac{\left(c_i^{k_i(t)+1} + c_i^{2r_it-k_i(t)}\right)}{\tau(1-c_i)} - r_i\cdot\left(1-c_i^{r_it}\right)^2\right)
$$

Asymptotic analysis reveals that this scales as $\mathcal{O}\left(\tau^{-2}\right)$ as $\tau \to 0^+$. As well, we use the fact that regularized least-squares finds $\mathbf{d}_i^{\mathbf{f}}$ that are independently distributed according to $\mathcal{N}(0,\sigma^2)$ where $\sigma = \mathcal{O}\left(n^{-1}\right)$ – i.e., variance scales as $\mathcal{O}\left(n^{-2}\right)$. Finally we use equation 3.10 in the second step below, and then leverage the property that for any two uncorrelated *zero-mean* variables, A and B, we have $\mathbb{V}[AB] = \mathbb{V}[A]\,\mathbb{V}[B]$. Therefore,

$$
\begin{aligned}
\sqrt{\mathbb{V}\left[\mu(t)\right]} &= \sqrt{\mathbb{V}\left[\sum_{i=1}^{n}(a_i * h)(t)\mathbf{d}_i^{\mathbf{f}} - \sum_{i=1}^{n}(r_i * h)(t)\mathbf{d}_i^{\mathbf{f}}\right]} \\
&= \sqrt{\mathbb{V}\left[\sum_{i=1}^{n}\left(\tilde{Y}_i(t) - \mathbb{E}\left[\tilde{Y}_i(t)\right]\right)\mathbf{d}_i^{\mathbf{f}}\right]} \\
&= \sqrt{n\mathbb{V}\left[\tilde{Y}_\cdot(t)\right]\mathbb{V}\left[\mathbf{d}_\cdot^{\mathbf{f}}\right]} \\
&= \sqrt{n\mathcal{O}\left(\tau^{-2}\right)\mathcal{O}\left(n^{-2}\right)} \\
&= \mathcal{O}\left(\frac{1}{\tau\sqrt{n}}\right). \qquad \square
\end{aligned}
$$

Satisfying the uniform ISIP criteria

Theorem 3.2.1 has important consequences for the correctness of NEF networks and the precision of neural computation in general. Assuming neural states remain distributed in a manner that satisfies the criteria of uniform ISIPs, then equation 2.10, which forms the basis for the efficient training of NEF networks, is mathematically correct. Thus, precision continues to scale (i.e., error converges to zero by the CLT), even as: (1) the number of neurons goes towards infinity (see Figure 5.5), and (2) for arbitrarily high-frequency stimuli (see Figure 5.4). This ensures that the NEF, and its postsynaptic current coding scheme (see section 3.1.5), is asymptotically correct – even when encoding arbitrarily high-frequency signals into spikes with arbitrarily small synaptic time-constants and low firing rates. We summarize many of these scaling properties in section 3.2.2.

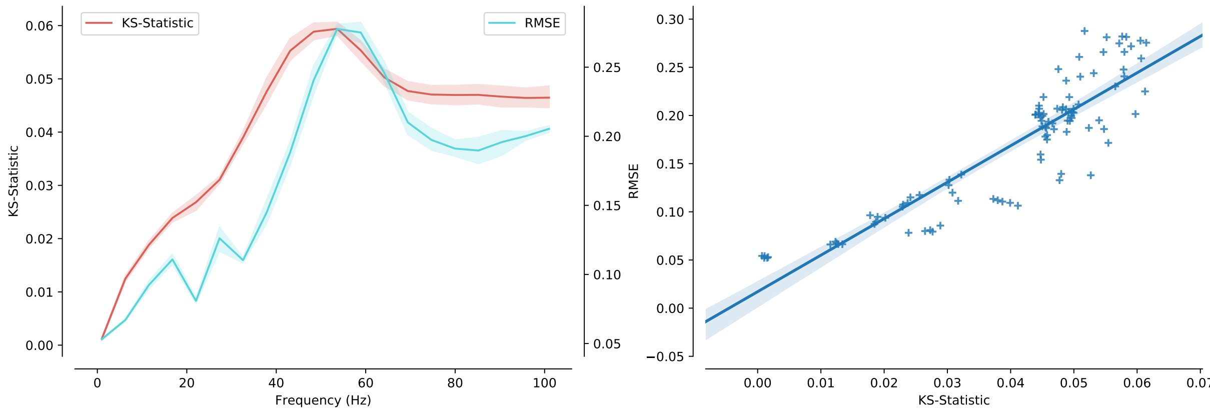


Figure 3.4: Relationship between state discrepancy (KS-statistic; definition 3.2.1) and representational error induced by filtered spikes (RMSE), for sinusoidal stimuli of varying frequencies. (Left) Empirical distribution of ISI positions (equation 3.6) is compared to the ideal uniform distribution using the KS-statistic (p-values $< 10^{-6}$). (Right) The RMSE is highly correlated with the KS-statistic, with a Pearson correlation coefficient of $R = 0.90$ (p-value $< 10^{-6}$).

However, in the NEF, when using LIF neurons, this criteria is violated as the frequency of the represented signal increases. To demonstrate, Figure 3.4 simulates a communication channel (n = 2,500, $\mathtt{dt}$ = 0.1 ms, τ = 1.6 ms, τ_{RC} = 20 ms, τ_{ref} = 2 ms), with tuning curves uniformly distributed to achieve maximum firing rates of 50–100 Hz, in order to encode sinusoid stimuli of increasing frequencies (visualizing 95% confidence interval, bootstrapped across 5 random configurations at each frequency). For 0 Hz (constant) inputs, the ISIP distribution is uniform (KS-statistic < 0.0002), as we should expect (i.e., the neural states remain independently mixed for a constant input). However, as the frequency increases, so does the state discrepancy (see left). Likewise, the state discrepancy is highly correlated ($R = 0.90$) with the RMSE of the spike noise (see right). Even though the state discrepancy is $< 6\%$ for input frequencies up to 100 Hz, it is important to remind the reader that this is a systematic error that *does not scale away* with higher neuron counts. On the one hand this makes theoretical predictions regarding the information-processing capabilities of LIF neurons for stimuli with various statistics, and in particular suggests a nonlinear transfer function that may have some functional relevance – but, on the other hand, here we are interested in understanding how this formalism relates to minimizing error and maximizing information throughput.

In order to scale away the state discrepancy, we note that the uniform ISIP assumption may be satisfied in a number of ways, many of which are validated in section 5.2.2:

1. The state of the system is initialized such that g_i is uniform. For the case of the LIF model, one can invert equation 3.6 and solve for (v_i, ρ_i) from a uniform sample, and then use this to initialize the state of each neuron. We do so in many of our experiments (including Figure 3.4) in order to remove the initial transient that would otherwise be a statistical anomaly (see Figure 3.3 (d)). Given a constant representation, the neural states naturally remain mixed due to heterogeneity in the encoding. However, this does not guarantee invariance over time for time-varying stimuli; leakage in the model as well as rapid inhibition can skew the ISIPs back towards zero.

2. The neuron model is replaced by a spiking rectified linear (ReLU) unit, also known as a (non-leaky) integrate-and-fire neuron. This model rectifies its input current, such that its state only evolves in one direction. This can also be achieved by setting τ_{RC} to be sufficiently large and lowering the neuron's voltage floor, as in Boerlin et al. (2013). Incidentally, Eliasmith and Anderson (2003, pp. 238–239) discovered that an integrator's performance was improved by increasing τ_{RC}, but attributed this improvement to static distortion being minimized by linear tuning curves, instead of having reduced the overall state discrepancy (also see Eliasmith and Anderson, 2003, appendix F.1).

3. Noise is introduced to the neuron model, to maintain a uniform subthreshold state (Eliasmith and Anderson, 2003, pp. 132–133). Indeed, this is the same assumption made by Eliasmith and Anderson (2003, p. 134) in their assessment of the quality of spiking neural representation.

4. The neuron model is replaced by its Poisson-spiking counterpart; that is, *any* static nonlinearity supplied to a Poisson spike-generator. Due to the memoryless nature of the Poisson process, at any given moment in time, the model is always uniformly ready to spike with the ideal probability (see section 5.2.2).

Future work is required for more biologically-detailed neuron models and synapses. We conjecture that the role of many adaptive mechanisms in biological systems is to invariantly maintain the uniform ISIP criteria with respect to the time-varying statistics of the represented signal. Such mechanisms would facilitate efficient coding by maximizing information throughput for a fixed energy-budget (see section 3.1.6 and Eliasmith and Anderson (2003, pp. 127, Table 4.1)).

3.2.2 Scalability

In this section we summarize the general trade-offs associated with scaling NEF networks. Some application-specific trade-offs are discussed in section 6.2.

Table 3.1 provides a comprehensive list of all such trade-offs that we are aware of to date. This includes guarantees on training time, resources for simulation, precision (in spiking, non-spiking, feed-forward, and recurrent architectures), energy usage, dimensionality, and finally relationships between signal bandwidth, synaptic time-constants, and neuron counts. For the scaling of spiking networks, we assume that the uniform ISIP criteria of Theorem 3.2.1 is satisfied. References to relevant sections and literature are included in the right-hand column.

As a brief but important aside, the trade-offs that involve time do so by fixing some time-scale of interest. That is, one chooses some unit associated with τ (e.g., seconds), or f (e.g., Hertz), and so forth. In theory, and in practice, the relative time-scales are what end up being important in determining the behaviour of the system. For example, if one were to slow down all of the time-constants in the network (τ, τ_{RC}, τ_{ref}, `dt`) by a factor of 10, while increasing all of the rates in the network by the same factor, this would automatically be equivalent to speeding up the stimulus' frequency (reciprocal of time) by a factor of 10 in the original network. Nothing else needs to be changed in order to compress or expand our perspective of time; weights, neuron counts, and decoded quantities are all unitless. This simply follows from dimensional analysis and consistent usage of time-constants and frequencies throughout, which highlights the importance of taking physical dimensions into consideration throughout each level of analysis. In theory, this means these trade-offs can be applied to any time-scale, although in practice biological and hardware constraints often determine the relevant time-scales.

Property	Scaling	Context	References
Multiply-adds per step	$\mathcal{O}\left(nq\right)$	Inference	Voelker and Eliasmith (2018)
Total memory	$\mathcal{O}\left(nq\right)$	Inference	Mundy et al. (2015); Voelker and Eliasmith (2018)
Time & memory	$\mathcal{O}\left(nq\right)$	Training	section 2.2.2 (assuming randomized SVD)
Static precision	$\mathcal{O}\left(n\right)$	Fitting	section 3.2.1
Spiking precision	$\mathcal{O}\left(\tau\sqrt{n}\right)$	Feed-forward	section 3.2.1
Spiking precision	$\mathcal{O}\left(\theta\sqrt{n}\right)$	Recurrent	section 3.2.1
Bandwidth (Hz)	$\mathcal{O}\left((2\pi\tau)^{-1}\right)$	Fixed energy	section 3.1.6
Relative energy	$\mathcal{O}\left(2\pi\tau f\right)$	Fixed precision	section 3.1.6
Neuron count	$\mathcal{O}\left(2\pi f\right)$	Fixed precision, $\tau = f^{-1}$	equation 3.4 and Figure 5.4
Dimensionality	$\mathcal{O}\left(n^{\frac{2}{3}}\right)$	Fixed precision	Gosmann (2018, p. 60)

Table 3.1: A summary of trade-offs present in standard NEF networks. Each property's scaling is context-dependent. Variables include neuron count (n), dimensionality (q), synaptic time-constant (τ), system-level time-constant (θ), and representational frequency (f).

3.2.3 Completeness

Continuous-time dynamical systems, such as those implemented by the NEF, serve as powerful, *universal models* of computation (Bennett, 1995). For example, a dynamical system describing the motion of a particle in three-dimensional space, consisting of a finite number of equations, is capable of simulating a Turing machine (Moore, 1990). Such systems are physically implementable, while being undecidable in the sense that virtually every question about their long-term behaviour is impossible to predict, even with perfect knowledge (Moore, 1991).

As established in the previous sections, the NEF may be used to train a network to approximate any continuous-time dynamical system arbitrarily well, with precision scaling in the number of neurons. Therefore, it follows that the class of NEF networks are equivalent in power to a Turing machine. The same can be said for continuous-time RNNs and other SNN models (Funahashi and Nakamura, 1993; Thalmeier et al., 2016). In theory, this means that these methods have the potential to implement arbitrary algorithms in distributed networks of computing elements.

In some ways, the universality of continuous-time dynamical computations should not come as a surprise, as a Turing machine is nothing more than a discrete-time dynamical system with an unbounded (but finite for decidable tasks) number of state-variables (Turing, 1938). Early work by Voelker and Eliasmith (2014) provided a constructive proof of Turing-completeness in the NEF, by using the Semantic Pointer Architecture (SPA; Eliasmith, 2013) to process the instructions of algorithms encoded into graphs that were sufficiently powerful to emulate multiple pushdown automata.

There is extensive literature on computability in both physical and non-physical substrates, leveraging dynamics that are either discrete or continuous in space and time. However, these theoretical results do not have much to say in practice about the constraints that a certain task will impose on the system, the resources needed by an optimal solution, or likewise the algorithms that demand the least amount of power according to the physical devices leveraged by the model of computation.

We see these concerns as being in the realm of what should, and can, be addressed by frameworks, methods, and tools (e.g., NEF and Nengo) that aim to expose and leverage some underlying model of computation. Thus, we do not take Turing-completeness to be the ultimate arbiter of whether some model of computation is suitable for describing the brain. Rather, we take this discussion as theoretical justification for the expressive power of dynamical systems, and therefore the utility in having well-understood tools for realizing them in neuromorphic hardware. The following sections are concerned with the robustness and extensibility of these tools.

3.2.4 Robustness

Many different notions of robustness in the NEF have already been studied by Eliasmith and Anderson (2003), Eliasmith et al. (2012), and Eliasmith (2013). Robustness to sources of variability and modelling errors in particular are essentially built-in to the framework. The NEF uses variable spikes to encode information, and heterogeneous basis functions for transformation, with zero-mean error (see Theorem 3.2.1). This enables the network's function to remain robust to randomly dropped or delayed spikes, weights that are quantized, perturbed, or sparsified, the removal of neurons, intrinsic noise, and similar sources of variability.

But in addition, there is a distinctly different kind of robustness that, to our knowledge, has not been discussed in the literature. As demonstrated in section 3.1.4, every non-trivial recurrent NEF network is naturally chaotic at the level of individual neurons. Yet, in spite of this chaos, the network functions correctly at the level of its state-space representation. Robustness to chaos has an important practical consequence: small measurement errors in the state of individual neurons—arising either from mechanisms in the system that rely on noisy observations, or through modelling errors of the mechanisms themselves—may be tamed at the population level. For example, an NEF integrator can be supplied a white noise process, of sufficiently small amplitude (relative to the radius of its stable point), and it will remain within the strange attractor corresponding to its fixed point, indefinitely. This enables a working memory that can persist states for long periods of time even in the presence of noise (Singh and Eliasmith, 2004).

In contrast, the methods of Boerlin et al. (2013) are not robust to such noise models, since they treat any noise as being a part of the signal. To be precise, if one constructs an integrator using their methods, and provides the same white noise process described above, then it will compute the exact integral of white noise: a random walk (not shown). After a sufficient amount of time, the state of the system will have completely wandered off from where it had started, with no way of remembering the value that it was supposed to store. This occurs even when the input is filtered using a lowpass synapse with an arbitrarily long time-constant; its integral is still a transient stochastic process.

We take this to be of importance in the context of biological modelling and neuromorphic hardware, since there is usually some undesirable signal, error, noise, or likewise, that must eventually be accounted for in some manner. Integrating zero-mean error is not a general solution (although it may be sufficient in some contexts in conjunction with saturation effects). The NEF accounts for this by explicitly incorporating a noise model into the training procedure (see equation 2.9) and implicitly establishing an array of fixed points in state-space (Eliasmith and Anderson, 2003, p. 237). In addition, the NEF is robust to the use of a number of different neuron models and synapse models, as is the focus of extensions discussed below.

3.2.5 Extensibility

As stated in section 2.3, NEF networks have been deployed on a number of different backends, including: SpiNNaker (Mundy et al., 2015), Neurogrid (Dethier et al., 2011), Braindrop (Neckar et al., 2019), Loihi (GitHub, 2019b), TrueNorth (Fischl et al., 2018), a VLSI prototype (Corradi et al., 2014), several FPGA implementations (Naylor et al., 2013; Wang et al., 2014; Berzish et al., 2016; Wang et al., 2017; GitHub, 2019a), and several GPUs (Bekolay et al., 2014; Rasmussen, 2018; Blouw et al., 2018). This would not have been possible if not for the extensibility of the NEF – that is, its flexibility in accounting for various non-idealities (e.g., Voelker et al., 2017a).

Our work in this direction is the topic of chapter 5. Of particular focus are the higher-order synapses with variable time-constants on Braindrop, and the configurably-delayed axons on Loihi. Our extensions can successfully leverage both to improve network-level computation. The focus of chapter 7 is then to showcase several recurrent SNNs deployed on both architectures, while using the same high-level model description for each network. The machinery of the NEF and Nengo, together with the extensions in this thesis, take care of compiling the model description onto the hardware while respecting its constraints and leveraging its dynamical primitives (Neckar et al., 2019).

Chapter 4

Methodology

This chapter is devoted to a brief overview of our methods, namely the software tools that we use, and our approaches to modelling useful computations as dynamical systems. Additional details are the focus of later chapters.

4.1 Software

The simulations in this thesis make extensive use of the following Python software packages (versions listed when important):

- Nengo 2.8.0 (Bekolay et al., 2014); to construct and simulate NEF models on the CPU, which depends on NumPy for its algebraic routines,
- Nengo-Loihi 0.5.0 (Blouw et al., 2018; GitHub, 2019b); to compile Nengo models onto the Loihi emulator and hardware (courtesy of ABR),
- Nengo-Brainstorm (pre-release; Neckar et al., 2018, 2019, courtesy of Terry Stewart); to compile Nengo models onto Braindrop, which depends on PyStorm (courtesy of the Boahen Lab),
- Hyperopt 0.0.2 (Bergstra et al., 2015); to optimize the hyperparameters associated with RC networks,
- Keras 2.2.4 (Gulli and Pal, 2017); to train LSTMs and some custom RNNs,

- TensorFlow 1.12.0 (Abadi et al., 2016); as a backend for Keras,
- Seaborn (Waskom et al., 2015); to generate figures, which depends on Matplotlib and Pandas for plotting and formatting data, respectively.

We do not take credit for any of the above software. However, we have created the Python package, NengoLib 0.5.1 (Voelker and Eliasmith, 2016; Voelker, 2019; Voelker and Eliasmith, 2019, patent pending), open-sourced on GitHub, to make the contributions of this thesis publicly available. Code, LaTeX, and instructions for reproducing figures, are located at: `https://github.com/arvoelke/phd/`. Unless stated otherwise, all simulations use a time-step of `dt` = 1 ms, and assume the input signal is held constant across each time-step, also known as the "zero-order hold" (ZOH) assumption. Experiments that implement the methods of Boerlin et al. (2013) in Nengo are currently unpublished (provided by Dan Rasmussen).

NengoLib includes the core code for most of the extensions discussed in chapter 5 and the networks discussed in chapter 6. It has been used by publications including Knight et al. (2016), Friedl et al. (2016), Voelker et al. (2017a), Voelker and Eliasmith (2017b), Voelker and Eliasmith (2018), and Neckar (2018). In addition, we use it to construct RC, FORCE, full-FORCE, and Nengo networks—each in spiking and non-spiking mode, and in an analogous manner to one another—in order to facilitate meaningful comparisons between these architectures. A core feature of this software is a toolkit for constructing and analyzing linear time-invariant systems, in a number of representations (state-space, transfer function, zero-pole-gain), while being compatible with the synapses of Nengo. This unifies the language used to build, describe, and analyze LTI networks, synapses, and subthreshold neural dynamics. We now summarize some additional features of this software that are used throughout.

State-space realizations

This section has been adapted from Voelker and Eliasmith (2018, appendix A.3). The state-space model (equation 2.12) is *not* a unique description of the input-output behaviour of an LTI system. In general, one may consider any invertible matrix T with the same shape as A, and observe that the state-space model $(TAT^{-1}, TB, CT^{-1}, D)$ has the same transfer function as (A, B, C, D). Thus, the state-space model is only unique up to a change of basis. However, in the NEF, the basis T may be "absorbed" into the representation of $\mathbf{x}(t)$ by using the encoders ET in Principle 1, which, in turn, results in the decoders $D^{\mathbf{f}}\left(T^{-1}\right)^{\top}$ from Principle 2. In other words, considering an alternative state-space model is equivalent to considering a change of basis for the representation. Nevertheless, in practice, when aiming to accurately represent $\mathbf{x}(t)$ using few neurons, it is important to balance the relative range of values within each dimension, such that a typical trajectory for $\mathbf{x}(t)$ stays within the space represented by the distribution of encoders, consistent with the samples of S (see equation 2.9), and the dynamic range of each neuron.

For this, there is a feature to *balance* the range of values by numerically computing the T that results in a "balanced realization" of (A, B, C, D) (Laub et al., 1987; Perev, 2011). We then set the encoders to be unit-length and axis-aligned, and optimize each dimension independently by using the methods from section 2.2.2. There is also the support to normalize each dimension by a diagonal transformation T with the i^{th} diagonal equaling $\max_t |x_i(t)|^{-1}$ where $x_i(t)$ is obtained by simulating the ideal system directly on a randomly sampled input. Finally, NengoLib supports a diagonal transformation T with the i^{th} diagonal corresponding to the reciprocal of two times the sum of the Hankel singular values (Glover and Partington, 1987) of the subsystem corresponding to $x_i(t)$. This has the effect of bounding the absolute value of each dimension above by 1 in the worst case (Khaisongkram and Banjerdpongchai, 2007).

In general, we find that each of the above methods to normalize state-space models typically improves the robustness of our networks across a wide range of parameters, but one is never strictly better than all others in every situation.

Random sampling

Many common routines in Nengo employ the use of high-dimensional random sampling (Voelker et al., 2017b; Gosmann, 2018), in particular the uniform sampling of encoders $\mathbf{e}_i$ from the surface of the hypersphere, and the uniform sampling of evaluation points from S (typically the interior of the hyperball).

Considering the NEF's optimization problem from equation 2.9, Monte Carlo (MC) sampling S introduces $\mathcal{O}\left(m^{-\frac{1}{2}}\right)$ error into the integral, where m is the number of samples, but this can be improved to $\widetilde{\mathcal{O}}\left(m^{-1}\right)$—effectively squaring m—by the use of quasi-Monte Carlo methods (Fang and Wang, 1994; Knight et al., 2016). For this, NengoLib overloads Nengo's default versions of these distributions, with its own method of uniformly scattering the samples using quasi-MC sampling. Specifically, we take a scattered sample from the hypercube, and then apply the inverse transform method with a spherical coordinate transform (Fang and Wang, 1994) to map samples onto either the hyperball or hypersphere. This supports the use of any number of quasi-MC methods for sampling from the cube, although we have implemented the most promising ones: Sobol sequence and R_2 (Sobol, 1967; Roberts, 2018).

We find that quasi-MC sampling the encoders also improves the representation at low numbers of neurons on average, by ensuring a better tiling or coverage of the encoded space.

For efficient coding in higher dimensions, methods of randomly sampling points from the Leech lattice—a 24-dimensional Lattice built from 196,560 unit-length vectors, with beautiful mathematical properties regarding their similarity in relation to efficient sphere-packing and error-correction codes (Conway and Sloane, 1999)—used by Knight et al. (2016) are also available in NengoLib.

4.2 Dynamics as a language

As discussed in section 3.2.3, dynamical systems represent a complete (i.e., universal) model of computation. But how are useful high-level computations embedded into such systems? To this end, we are interested in developing a language for describing useful algorithms within the dynamical modelling framework of the NEF, and libraries for combining these models within Nengo.

Early work by Eliasmith and Anderson (2003) has shown that many important neurobiological systems, such as the nuclei involved in horizontal eye control, may be modelled as an integrator. That is, some velocity signal, indicating a desired direction, is represented by one population and then integrated by another. Such systems are describable as a one-dimensional linear time-invariant system (see equation 2.12), such as the one demonstrated in section 3.1.4 using ten spiking neurons. When extended to higher dimensions, this same system is capable of storing multiple items in series, and has consequently been applied to build sophisticated models of working memory (Singh and Eliasmith, 2004; Choo, 2010).

Oscillatory dynamics (e.g., relaxation oscillators), such as those modelled in Eliasmith and Anderson (2000) have utility in generating rhythmic activity to support various computations. For example, theta oscillations in the range of 4–12 Hz, through a phenomenon known as phase precession, have been shown to support the encoding of spatial information using Nengo (O'Keefe and Recce, 1993; Orchard et al., 2013). Likewise, a bank of oscillators can be used to facilitate motion processing in the early visual system (Hurzook, 2012).

Attractor networks describe a much broader class of nonlinear dynamical systems that capture many important phenomena observed in neurobiological systems (Amit, 1989; Eliasmith, 2005). When oscillations, integrators, and point attractors, are all combined, networks may generate complex trajectories through space (Ijspeert et al., 2013; DeWolf and Eliasmith, 2017). By arranging all of these building blocks in an engineered fashion, together with feed-forward synaptic dynamics coupling various modules, complex systems such as Spaun may carry out end-to-end cognitive tasks and even learn to follow general sets of instructions (Eliasmith et al., 2012; Choo, 2018). The role of chapter 6 is to unveil another class of dynamical systems for continuous nonlinear memory, that connects with "time cell" data from the neuroscience literature, and naturally fits into this framework. But first, we describe a few concrete examples to help illustrate the universality of dynamical descriptions of computation, in a number of distinct contexts.

4.2.1 Winner-take-all

Winner-take-all (WTA) circuits essentially engineer a "max" function that pools across *spatial* dimensions. These systems may be described in state-space using nonlinear differential equations, that compete with one another in order to resolve the correct solution over time, as in an attractor network (Usher and McClelland, 2001; Gosmann et al., 2017). Here, we are interested in a very different kind of WTA circuit that must compute its max across temporal dimensions. We call this circuit a "peak detector".[1] Specifically, we suppose the input vector, $\mathbf{u} \in \mathbb{R}^m$, is supplied online, one dimension at a time, such that:

$$\mathbf{u}[k] = (u[k], u[k-1], \ldots, u[k-m])^{\mathsf{T}}$$

for some (possibly infinite) number of samples, $m \geq 1$. In this context, determining the maximum of $\mathbf{u}[k]$ is equivalent to the task of tracking the peak of the past history of the discrete-time signal, $u[k]$ (see Figure 4.1), at a sampling interval of $\mathtt{dt}$. In other words, the problem of tracking the maximum across time, is equivalent to a WTA mechanism where the input dimensions are provided sequentially, as opposed to being provided all at the same moment in time (as is usually the case).

In the limit of $\mathtt{dt} \to 0$, the desired continuous-time dynamical system is:

$$\dot{x}(t) = f(x(t), u(t)), \quad f(x,u) = \theta^{-1}\,[u - x]_+ \tag{4.1}$$

where $[\cdot]_+$ denotes positive half-wave rectification. $\theta > 0$ is a time-constant that determines how quickly the state should adjust to the positive difference. Note that in a purely continuous-time system, assuming no noise, θ can be made arbitrarily small, in order to track the peak arbitrary quickly. However, since we have a discrete sampling time, our desired dynamical system is:

$$x[k+1] = \bar{f}(x[k], u[k]), \quad \bar{f}(x,u) = \bar{\theta}^{-1}\,[u - x]_+ + x \tag{4.2}$$

where $\bar{\theta} > 0$ is a (dimensionless) discrete time-constant. Equation 4.2 is related to equation 4.1 by $\bar{\theta} = \theta\,(\mathtt{dt})^{-1}$, assuming Euler integration. Note that $\bar{\theta} = 1$ will give a perfect peak detector, assuming no noise.

[1] This problem was proposed by Jan Gosmann (unpublished).

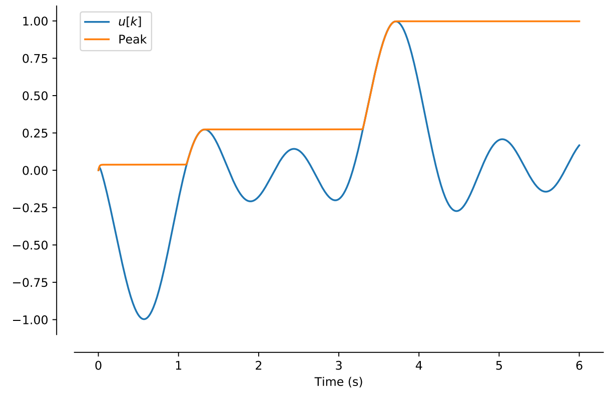


Figure 4.1: An ideal "peak detector" implemented as a discrete-time dynamical system. This is equivalent to performing a winner-take-all across data provided sequentially.

In Figure 4.1, we implement a peak detector by using Nengo directly as a dynamical systems simulator. This does not use any neurons, but rather digitally implements equation 4.2 with $\bar{\theta} = 1$ by coupling each variable in the correct way:

```
with nengo.Network() as model:
    u = nengo.Node(stim)  # stim defines the test input signal
    peak = nengo.Ensemble(1, dimensions=2, neuron_type=nengo.Direct())
    nengo.Connection(u, peak[1], synapse=None)
    function = lambda x: (x[1] - x[0]).clip(min=0) + x[0]
    nengo.Connection(peak, peak[0], synapse=~z, function=function)
```

We remark that $\sim z$ is a dynamical primitive in NengoLib that implements a single time-step delay, in correspondence with the $\mathcal{Z}$-transform in signal processing. This illustrates the utility of discrete-time dynamical systems, the flexibility of this framework for describing useful computations, and the versatility of Nengo as a programming tool for simulating such systems. Both continuous-time and discrete-time versions of this system may be implemented using spiking neurons as well (not shown). In chapter 5 we will address the implementation of such systems—with both discrete- and continuous-time dynamics—using populations of spiking neurons in analog and digital hardware.

4.2.2 Unsupervised learning

This section has been adapted from Voelker et al. (2014), and discusses the dynamics of an unsupervised learning rule used by Voelker and Eliasmith (2014), Trujillo (2014), and Aubin et al. (2017). This has been implemented on SpiNNaker (Knight et al., 2016), and is discussed in some detail by Aubin (2018). Here we expose the dynamics of this learning rule, while highlighting its function for sparsifying across space in large-scale models.

Associative memories have been an active area of research over the last forty years (Willshaw et al., 1969; Kohonen, 1972; Hopfield, 1982) because they form a central component of many cognitive architectures (Pollack, 1988; Anderson and Lebiere, 1998). We focus specifically on associative memories that store associations between arbitrary pairs of neural states. When a noisy version of an input state vector is presented to the network, it must output a "clean" version of the associated state vector. Stewart et al. (2011) has built such a memory network, possessing a number of desirable properties including: high accuracy, a fast, feed-forward recall process, and efficient scaling, requiring a number of neurons linear in the number of stored associations. These memories have played a central role in several cognitive models including Spaun, as well a proposal for human-scale, biologically-plausible knowledge representation (Crawford et al., 2015). However, these memories are constructed using an offline optimization method that is not biologically-plausible.

We describe a method for building large-scale networks for online learning of associations using spiking neurons. Connection weights can be arrived at through a biologically-plausible, online learning process featuring a novel synaptic learning rule inspired in part by the well-known Oja learning rule (Oja, 1989). We call this rule the *Voja* learning rule (for "vector Oja"). Although the network itself is feed-forward, the dynamics that carry out the important computations are embedded within the learning rules that modify its synaptic weights online. We now describe the learning rule.

Given a learning rate η, an input vector $\mathbf{x}$ encoded by the activity of the input layer, the filtered activity $a(t)$ of neurons in the middle layer, and the matrix E whose rows are the "preferred direction" vectors of the middle layer neurons, we modify the preferred direction vectors of the middle layer neurons according to the dynamical system:

$$\dot{E} = \eta \left(a(t)\mathbf{x}^{\mathsf{T}} - a(t)E \right) = \eta a(t) \left(\begin{pmatrix} \mathbf{x}^{\mathsf{T}} \\ \vdots \\ \mathbf{x}^{\mathsf{T}} \end{pmatrix} - E \right). \tag{4.3}$$

To understand the effect of this rule over time, we set $\dot{E}_i = 0$ and solve for the fixed point. This gives $a_i(t)\mathbf{x} = a_i(t)\mathbf{e}_i$, and thus, for a particular $\mathbf{x}$, convergence is characterized by:

$$[\dot{\mathbf{e}}_i = 0] \iff [a_i(t) > 0 \implies \mathbf{e}_i = \mathbf{x}]. \tag{4.4}$$

This defines an attractor landscape whose set of fixed points correspond to the set of inputs presented over time. In plain words, the encoding vectors of the neurons that become active will, over time, become more active and store the input within its encoder. The effect of this rule is to self-organize a sparse representation, such that a subset of neurons fire only when $\mathbf{x}$, or a similar enough (i.e., noisy) version of it, is presented (see Figures 4.2 and 4.3). The preferred direction vectors of each neuron are embedded within the synaptic weights projecting from the previous layer. Thus, this can be realized as a local learning rule by multiplying the decoders through and expressing the update as its weight with the j^{th} presynaptic neuron (Knight et al., 2016):

$$\dot{\omega_{ij}} = \dot{\mathbf{e}}_i \cdot \mathbf{d}_j = \eta\, a_i (\mathbf{d}_j \cdot \mathbf{x} - \omega_{ij}). \tag{4.5}$$

In effect, this rule achieves some useful computation—the unsupervised learning of inputs—by implementing a dynamical system within the state of each synapse. The networks of Knight et al. (2016); Aubin et al. (2017) learn to associate each input vector with a distinct output vector, by combining this rule with a supervised learning rule discussed below.

4.2.3 Supervised learning

This section has been adapted from preliminary work by Voelker (2015b) and Voelker and Eliasmith (2017a). This approach highlights the utility in shifting perspectives in learning to that of a dynamical system embedded within some larger function. In particular, online learning is *nothing more* than a dynamical system mapped onto localized computations manipulating state-variables within a network. We illustrate this by unifying Nengo's supervised learning and the NEF's dynamical systems framework at a theoretical level, and then leverage the resulting relationship in a practical spiking example with online learning.

The NEF typically learns its connection weights offline, but Nengo also supports a number of biologically-plausible supervised and unsupervised learning rules to learn these weights online (Bekolay, 2011). Prescribed Error Sensitivity (PES; Bekolay et al., 2013) is a biologically-plausible Hebbian supervised learning rule that is frequently used within Nengo models. PES modifies the connection weights between populations of neurons to minimize an external (i.e., supervised) error signal. It is equivalent in function to a single-layer application of gradient descent, applied online (i.e., stochastically) to its current input and supervised output. This learning rule has been used to model episodic memory (Trujillo and Eliasmith, 2014),

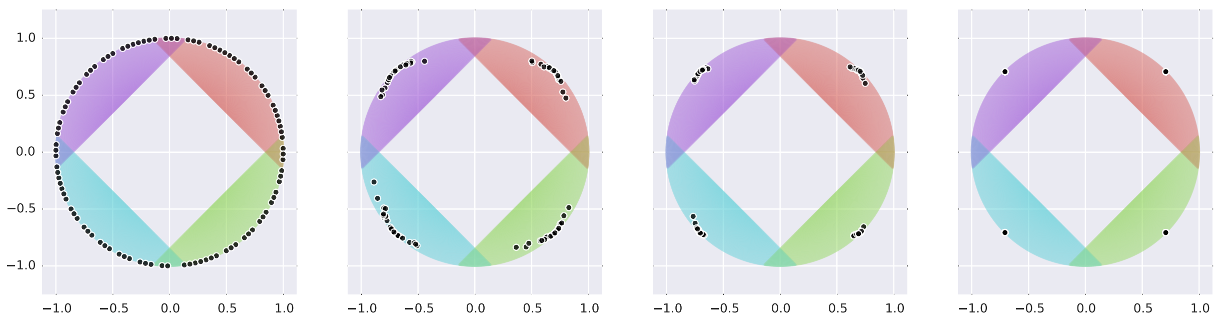


Figure 4.2: The effect of Voja on the encoders of a 2-dimensional population, over time. Each point corresponds to the encoder of one of 100 neurons. Four input vectors are chosen of the form $(\pm 1/\sqrt{2}, \pm 1/\sqrt{2})$, and their areas of attraction within the unit circle are indicated by shaded regions ($c = 0.6$). As the simulation progresses, from left to right, each encoder converges to one of the possible inputs. Reproduced from Knight et al. (2016).

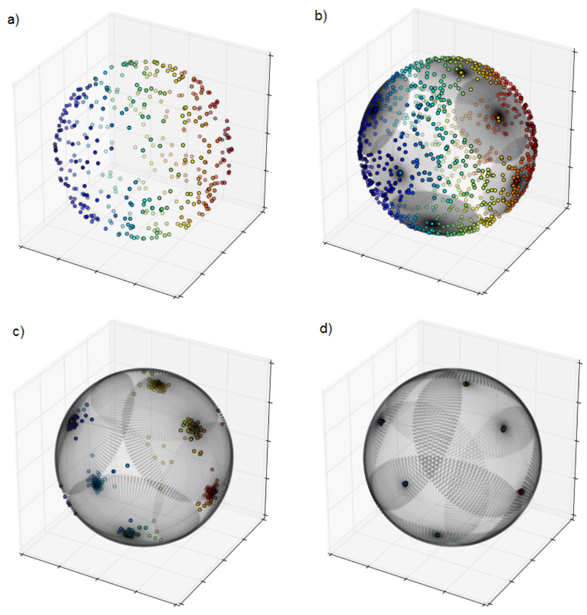


Figure 4.3: Input vector clustering of a 3-dimensional ensemble, presented with 6 evenly spaced **x**. (a) The initial clustering, before any input has been given. (b) Each input vector has been shown for two seconds of simulation time, with intercepts set to $\cos(\pi/6)$. Gray plates show the area of attraction for each input vector. (c) Same as b, with intercepts chosen to give $p_i = 1/6$. (d) Same as b, with intercepts set to $\cos(\pi/3)$. Reproduced from Voelker et al. (2014).

hierarchical reinforcement learning (Rasmussen et al., 2017), adaptive motor control (Komer, 2015; DeWolf et al., 2016), and many other tasks (Aubin, 2018; Choo, 2018). The PES learning rule is also partially supported by Loihi [personal communication].

We solve the discrete dynamical system for the case of constant inputs and no noise, to show that the decoding vectors given by the NEF have a simple closed-form expression in terms of the number of simulation time-steps. Moreover, with $\gamma = (1 - \kappa \|\mathbf{a}\|^2) < 1$, where κ is the learning rate and $\mathbf{a}$ is the vector of firing rates, the error at time-step k is the initial error times γ^k. Thus, $\gamma > -1$ implies exponential convergence to a unique stable solution, $\gamma < 0$ results in oscillatory weight changes, and $\gamma \le -1$ implies instability.

Lemma 4.2.1 (Dynamics of discrete unfiltered supervised learning)**.** (Voelker, 2015b) *Let* $\gamma = (1 - \kappa \|\mathbf{a}\|^2) < 1$*, and* $e_0 = y^* - \mathbf{d}[0]^T \mathbf{a}$*, then:*

$$\mathbf{d}[k] = \mathbf{d}[0] + e_0 \frac{\mathbf{a}}{\|\mathbf{a}\|^2}(1 - \gamma^k), \quad k \in \mathbb{N}, \tag{4.6}$$

and so the error at time-step k is $y^* - \mathbf{d}[k]^T \mathbf{a} = e_0 \gamma^k$. In particular, if $\gamma > -1$, then $\mathbf{d}[\infty]$ exists and is given by the unique solution,

$$\mathbf{d}[\infty] = \mathbf{d}[0] + e_0 \frac{\mathbf{a}}{\|\mathbf{a}\|^2}. \tag{4.7}$$

On the other hand, if $\gamma \le -1$ then the system is unstable. The proof is given by Voelker (2015b).

Continuing this work, we solve for the dynamics of PES, while filtering the error with an arbitrary linear synapse model. For the most common case of a lowpass filter, the continuous-time weight changes happen to be characterized by a second-order bandpass filter with frequency $\omega = \sqrt{\tau^{-1} \kappa \|\mathbf{a}\|^2}$ and bandwidth $Q = \sqrt{\tau \kappa \|\mathbf{a}\|^2}$, where τ is the exponential time constant, κ is the learning rate, and $\mathbf{a}$ is the activity vector. Therefore, the error converges to zero, yet oscillates if and only if $\tau \kappa \|\mathbf{a}\|^2 > 1/4$. This provides a heuristic for setting κ based on the synaptic τ, and a method for engineering remarkably accurate decaying oscillators using only a single spiking leaky integrate-and-fire neuron.

Lemma 4.2.1 fully characterized the discrete-time dynamics of PES under the restricted setting of a constant input signal, constant reference signal, and no noise. Due to the absence of noise, no filter was required for the error signal. However, for spiking networks considered in practice, a lowpass is applied to the error to filter out spike-noise (e.g., DeWolf et al., 2016; Rasmussen et al., 2017), We now relax the assumption of a constant reference signal, and apply an arbitrary linear filter to the error signal. For simplicity, we do so for the case of a continuous-time simulation, but our analysis can also be applied to the discrete-time setting via the $\mathcal{Z}$-transform. To keep our analysis tractable, we still assume a constant input signal, and briefly discuss implications for the general dynamic setting.

We begin by formulating a mathematical description of the network, present our theoretical results, prove them mathematically, validate them with numerical simulations, and demonstrate the utility of this analysis by engineering oscillators with predetermined frequencies and decay rates. Finally, we conclude by discussing some implications for learning spiking dynamical networks online.

Prescribed Error Sensitivity

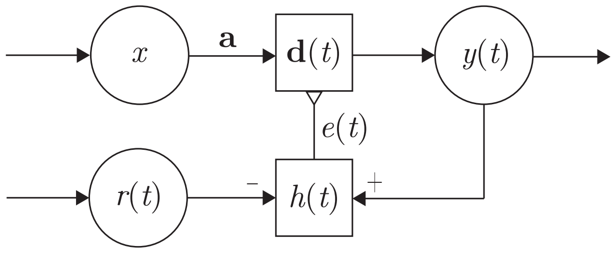


Figure 4.4: Network diagram used to analyze the PES rule. A constant input x is represented by a population of n spiking neurons with the activity vector $\mathbf{a} \in \mathbb{R}^n$. A dynamic reference signal $r(t)$ determines the error $e(t) = \big((y - r) * h\big)(t)$ (equation 4.10), which in turn drives $y(t)$ towards $r(t)$ by modulating the connection weights via PES (equation 4.9). These learned connection weights decode $y(t)$ via the decoders $\mathbf{d}(t) \in \mathbb{R}^n$ (equation 4.8). A linear filter $h(t)$ models the post-synaptic current induced by each spike. Reproduced from Voelker and Eliasmith (2017a, Figure 1).

Consider a network containing a population of n spiking neurons, encoding the constant scalar input x. Let $\mathbf{a} \in \mathbb{R}^n$ be the average (i.e., rate) activity of each neuron in response to this encoding.[2] This vector is determined by the first principle of the NEF, and remains fixed for constant x. The *decoders* $\mathbf{d}(t) \in \mathbb{R}^n$ determine the scalar output $y(t)$ via the dot product:

$$y(t) = \mathbf{a}^\mathsf{T}\mathbf{d}(t). \tag{4.8}$$

The PES rule learns these decoders, online, according to the following dynamics:

$$\dot{\mathbf{d}}(t) = -\kappa e(t)\mathbf{a}, \tag{4.9}$$

[2]Here on, we assume that $\mathbf{a} \neq \mathbf{0}$, otherwise PES will have no effect.

where $\kappa > 0$ is the learning rate,[3] and $e(t)$ is the chosen error signal:[4]

$$e(t) = \big((y - r) * h\big)(t), \tag{4.10}$$

where $r(t)$ is the reference (i.e., ideal) output, and $h(t)$ is some arbitrary linear filter modeling the post-synaptic current (PSC) induced by a spike arriving at the synaptic cleft. Typically, $h(t)$ is a first-order lowpass filter with time constant $\tau > 0$ (i.e., equation 2.13):

$$h(t) = \frac{1}{\tau} e^{-\frac{t}{\tau}} \iff H(s) = \frac{1}{\tau s + 1}. \tag{4.11}$$

The final network is summarized in Figure 4.4. This also naturally extends to the case where x and y are vectors (using a population code), but we consider the scalar case for simplicity.

Now, we aim to characterize the dynamics of $e(t)$ in response to the control signal $r(t)$. Alternatively, we could characterize the dynamics of $y(t)$ or $\mathbf{d}(t)$, but the former is easier to work with, while describing the latter via equations 4.8 and 4.9 (i.e., by integrating $e(t)$).

Theorem 4.2.2 (Dynamics of supervised learning with continuous filtered feedback)**.** (Voelker and Eliasmith, 2017a) *Let $\phi = \tau\kappa\|\mathbf{a}\|^2$. For the network described in Figure 4.4, we have:*

$$e(t) = (r * f)(t), \tag{4.12}$$

where:

$$F(s) = \frac{-s}{sH(s)^{-1} + \kappa\|\mathbf{a}\|^2}, \tag{4.13}$$

hence $F(s)$ is the transfer function from $R(s)$ to $E(s)$. For the case of a first-order lowpass filter (equation 4.11),

$$\implies \quad F(s) = \frac{-s}{\tau s^2 + s + \kappa\|\mathbf{a}\|^2} = \frac{-\left(\kappa\|\mathbf{a}\|^2\right)^{-1} s}{\left(\frac{1}{\omega^2}\right)s^2 + \left(\frac{1}{\omega Q}\right)s + 1} \tag{4.14}$$

$$\omega = \sqrt{\tau^{-1}\kappa\|\mathbf{a}\|^2} = \tau^{-1}\sqrt{\phi} \tag{4.15}$$

$$Q = \sqrt{\tau\kappa\|\mathbf{a}\|^2} = \sqrt{\phi}. \tag{4.16}$$

Thus, $F(s)$ is a second-order Q-bandpass filter with frequency ω in radians per second ($\frac{\omega}{2\pi}$ is the frequency in hertz) (Zumbahlen et al., 2011, pp. 8.9–8.10). The poles of $F(s)$ are:

$$s = \frac{-1 \pm \sqrt{1 - 4\phi}}{2\tau}. \tag{4.17}$$

[3] κ is automatically scaled by n^{-1} in Nengo, to balance the linear scaling of $\|\mathbf{a}\|^2$.

[4] Signs are flipped in Voelker (2015b); equations 4.9 and 4.10 are consistent with Nengo.

Since $\phi > 0$, this system is exponentially stable, and, moreover, the impulse response, $f(t)$, is a decaying oscillator if and only if $\phi > 1/4$.

Proof. We begin by transforming equations 4.8–4.10 into the Laplace domain:

$$\begin{aligned} Y(s) &= \mathbf{a}^\top \mathbf{D}(s) \\ s\mathbf{D}(s) &= -\kappa E(s)\mathbf{a} \\ E(s) &= (Y(s) - R(s))\, H(s).^{5} \end{aligned}$$

Substituting the first two equations into the last, yields:

$$\begin{aligned} E(s) &= \left(\mathbf{a}^\top \mathbf{D}(s) - R(s)\right) H(s) \\ &= \left(-\mathbf{a}^\top s^{-1} \kappa E(s)\mathbf{a} - R(s)\right) H(s) \\ &= -\kappa \|\mathbf{a}\|^2 s^{-1} H(s) E(s) - R(s) H(s) \\ \iff \qquad \left(1 + \kappa \|\mathbf{a}\|^2 s^{-1} H(s)\right) E(s) &= -R(s) H(s) \\ \iff \qquad E(s) &= R(s) \left(\frac{-H(s)}{1 + \kappa \|\mathbf{a}\|^2 s^{-1} H(s)} \right) \\ &= R(s) \left(\frac{-s}{sH(s)^{-1} + \kappa \|\mathbf{a}\|^2} \right) \\ &= R(s) F(s). \end{aligned}$$

Equations 4.12 and 4.13 follow from the convolution theorem. Equations 4.14–4.16 are verified by substituting $H(s)^{-1} = \tau s + 1$ into equation 4.13.

The poles of the system (equation 4.17) are obtained by applying the quadratic formula to the denominator polynomial from equation 4.14 ($\tau s^2 + s + \kappa \|\mathbf{a}\|^2$). Exponential stability is implied by both poles being strictly in the left half-plane. Lastly, $f(t)$ oscillates if and only if the poles are complex, if and only if the discriminant $(1 - 4\phi)$ is negative, if and only if $\phi > 1/4$. □

Validation

We construct the network from Figure 4.4 using Nengo, with only $n = 1$ spiking leaky integrate-and-fire neuron (mean firing rate of 262 Hz),[6] $\tau = 0.1$ s (equation 4.11), $x = 0$, and κ such that $\phi > 1/4$. We construct the transfer function from equation 4.14 using NengoLib:

[5] This has nice form since **a** is a constant – otherwise multiplication of two time-varying signals becomes a complex integral in the Laplace domain.

[6] Spikes are used in place of **a** in equations 4.8 and 4.9.

```
import nengolib
from nengolib.signal import s
H = nengolib.Lowpass(tau)
F = −s / (s/H + kappa*a.dot(a))
```

where `tau` $\leftarrow \tau$ is the time constant of the synapse, `kappa` $\leftarrow \kappa$ is the learning-rate supplied to Nengo (divided by n), and `a` $\leftarrow \mathbf{a}$ is the NumPy array for the population's activity. We evaluate $(r * f)(t)$ using `F.filt(r, dt=dt)`, and compare this to the $e(t)$ obtained numerically in simulation. The `filt` method automatically discretizes $F(s)$ according to the simulation time-step (`dt` $= 1\,\text{ms}$) using zero-order hold (ZOH).[7]

In Figure 4.5, we confirm that $(r * f)(t)$ approximates the numerical $e(t)$ given white noise $r(t)$. In Figure 4.6(Top), we exploit our knowledge of the impulse response, $f(t)$, to engineer a number of decaying oscillators by controlling $r(t)$. In Figure 4.6(Bottom), we evaluate `np.abs(F.evaluate(freqs))` at a variety of frequencies (`freqs`) to visualize the bandpass behaviour of each filter.

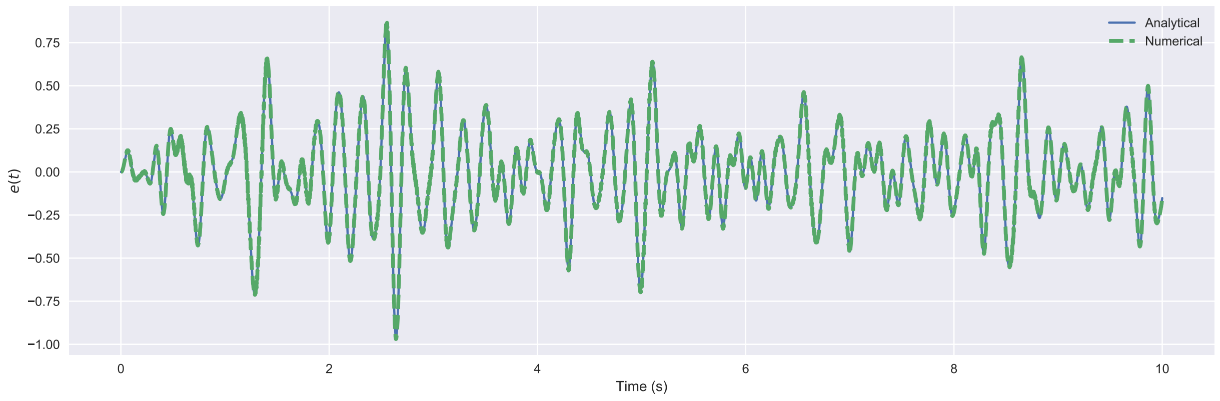


Figure 4.5: Comparison of the analytical error (equation 4.14) to the numerical $e(t)$ obtained by simulating the network from Figure 4.4 ($\kappa = 10^{-3}$). The control signal $r(t)$ is randomly sampled white noise with a cutoff frequency of 10 Hz. The normalized root-mean-square error is approximately 3.7%. Reproduced from Voelker and Eliasmith (2017a, Figure 2).

[7]Technically, for a discrete-time simulation, the problem and results should have been formulated in the discrete-time domain using the Z-transform to begin with, as opposed to discretizing at the end, but the difference is quite subtle.

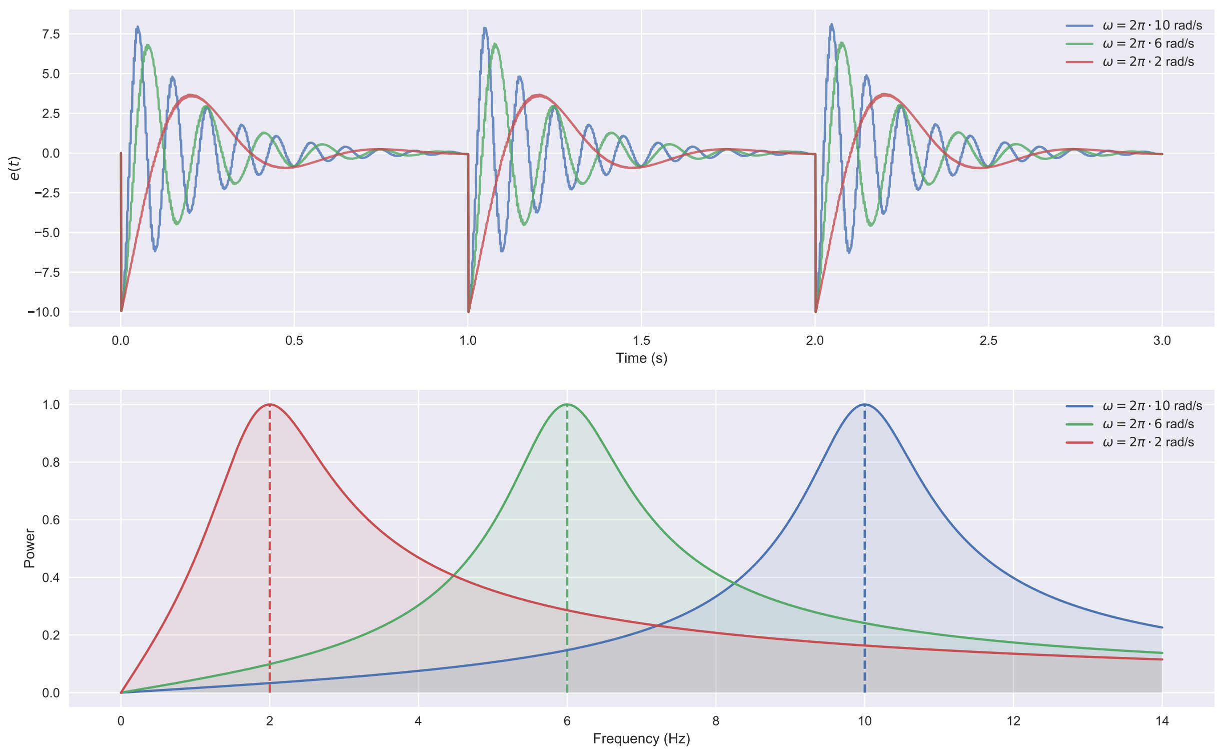


Figure 4.6: Harnessing the dynamics of PES with various κ to engineer decaying oscillators with predetermined frequencies (ω) and bandwidths (Q), using only a single leaky integrate-and-fire neuron. The time constant of the first-order lowpass filter is fixed at $\tau = 0.1$ s, while κ is set to achieve the desired ω via equation 4.15. (Top) Once every second, $r(t)$ is set to a unit-area impulse. Consequently, $e(t)$ oscillates according to the impulse response, $f(t)$. (Bottom) Visualizing the ideal frequency response of $F(s)$ (equation 4.14). Dashed lines at ω (equation 4.15) align with the peak of each bandpass filter, or equivalently the frequency of each oscillation. The width of each filter is proportional to the decay rate Q^{-1} (equation 4.16). Reproduced from Voelker and Eliasmith (2017a, Figure 3).

Discussion

Since $\phi = \tau\kappa\|\mathbf{a}\|^2 > 1/4$ if and only if the weights oscillate, this motivates a simple heuristic for setting the learning rate to prevent oscillatory weight changes: set $\kappa \leq \dfrac{1}{4\tau\|\mathbf{a}\|^2}$, where $\|\mathbf{a}\|^2$ is maximal over all possible activity vectors. In this case, equation 4.14 factors into a (differentiated) double-exponential:

$$F(s) = \frac{-\tau_1\tau_2 s}{\tau(\tau_1 s+1)(\tau_2 s+1)} = \left(-\tau_1\tau_2\tau^{-1}s\right)\left(\frac{1}{\tau_1 s+1}\right)\left(\frac{1}{\tau_2 s+1}\right),$$

that is, two first-order lowpass filters chained together, where:

$$(\tau_1, \tau_2) = \frac{2\tau}{1 \mp \sqrt{1-4\phi}},$$

by equation 4.17. In other words, the non-oscillatory regime ($0 < \phi \le 1/4$) of PES is characterized by the dynamics of a double-exponential. We remark that $\tau_1 = \tau_2 = 2\tau$ (i.e., an alpha filter) directly on the point of bifurcation from double-exponential to oscillatory behaviour ($\phi = 1/4$; see Figure 4.7).

In all applications involving online learning (that we are aware of) oscillatory weight changes are viewed as problematic, and so the relevant constants (τ, κ, and $\|\mathbf{a}\|^2$) are tweaked until the issue disappears. In contrast, we have shown that not only can the relationship between these constants and the oscillations be fully understood, but they can be harnessed to engineer bandpass filters (with respect to the transformation $r(t) \mapsto e(t)$) with specific frequencies (ω) and bandwidths (Q). More generally, the PES learning rule can be used to construct dynamical systems whose transfer function (equation 4.13) depends on $H(s)$, κ, and $\|\mathbf{a}\|^2$. As we used only a single spiking neuron, the accuracy of these systems rely solely on the accuracy of the PES implementation, the model of $H(s)$, and the constancy of $(\mathbf{a} * h)(t)$ in practice (i.e., given spiking activity).

Although we have analyzed the continuous-time setting, the same proof technique can be applied to the discrete-time domain by use of the Z-transform. Likewise, although we have assumed x is a constant, we can apply a "separation of time-scales" argument (i.e., assuming $x(t)$ changes on a slower time-scale than $f(t)$) to carry this same analysis over to dynamic $x(t)$. By equation 4.17, this analysis holds approximately for $x(t)$ with frequencies $\ll (4\pi\tau)^{-1}$ Hz, by applying a time-varying filter to $\mathbf{d}(t)$ that depends on the current activity vector. Incidentally, this bound is only half that of our definition of low-frequency representation from section 3.1.6.

In conclusion, we have extended our previous analysis of PES to include linearly filtered feedback and a dynamic reference signal. This fully characterizes the rule in the context of NEF networks representing a constant value, as a transfer function from the reference signal to the error signal. This transfer function may then be readily analyzed and exploited using linear systems theory, to instantiate oscillators using far fewer resources (i.e., a single neuron coupled with a local learning rule) than what would be needed to achieve the same level of precision with a recurrent SNN implementing the same dynamical system. This demonstrates a more general principle of recurrently coupling available dynamical primitives in biological models (here, a PSC that is integrated by an online learning rule) to improve network-level computations.

Many other aspects of NEF networks may also be analyzed from a dynamical systems perspective. For instance, the dynamics of the representational error, the chaotic dynamics of voltage traces, or the undecidable attractor dynamics induced by neural saturation (see section 3.1.4). We do not go into detail here. We mention this because, similar to our lessons learned in the above discussion, it may become important to have the ability to systematically characterize and subsequently leverage the dynamics that are already naturally present in the network's function.

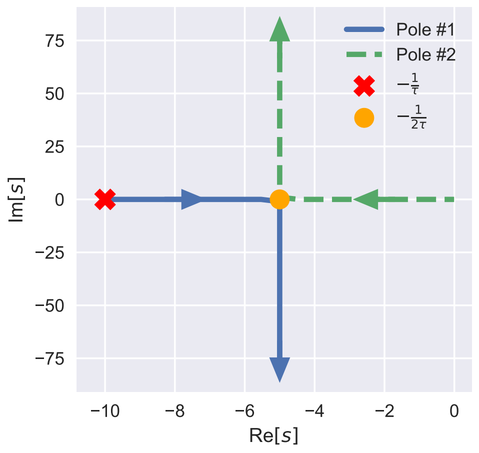


Figure 4.7: Visualizing the poles of $F(s)$ (equation 4.17) by sweeping $\kappa > 0$ (while $\|\mathbf{a}\|^2$ and $\tau = 0.1\,\mathrm{s}$ remain fixed). Arrows follow the direction of *increasing* κ. When $\phi \le \frac{1}{4}$, the dynamics of PES are a double-exponential. The learning rule becomes an alpha filter when the two poles collide: $\phi = \frac{1}{4} \iff s = -\frac{1}{2\tau}$ (marked by a solid circle). When $\phi > \frac{1}{4}$, the weight changes become oscillatory (due to complex poles). As κ increases, the oscillatory frequency, ω, scales as $\mathcal{O}\left(\sqrt{\kappa}\right)$. As κ decreases, the first pole converges to $s = -\frac{1}{\tau}$ (marked by a solid x) while the second pole cancels the zero at $s = 0$. Reproduced from Voelker and Eliasmith (2017a, Figure 4).

4.2.4 Optimization

There is extensive literature on the general topic of implementing optimization algorithms in the form of dynamical systems, dating back at least to the work of Brockett (1991). Voelker (2014) noticed that the "hill climbing" algorithm may be cast as a dynamical system, and leveraged this to construct an SNN that locally searches across an error landscape. This observation naturally extends to any gradient-based algorithm, since these algorithms are expressed as state-vectors that update continuously according to some (possibly nonlinear) function of the state of the system. Instances of this include gradient descent, feedback alignment (**?**), Bayesian inference, and expectation-maximization (Sharma et al., 2017).

Apart from what has been done in the NEF, there exist many other dynamical systems that solve optimization problems, that could just as easily be realized using Principle 3, including: compressed sensing (Kim et al., 2007), sparse coding (LASSO; Shapero et al., 2014; Tang et al., 2017), dictionary learning (Lin and Tang, 2018), Hopfield networks (Hopfield, 1982; Frady and Sommer, 2019), non-negative similarity matching (Pehlevan, 2019), and various other algorithms related to eigenvalues, independent component analysis, and non-negative matrix factorization. But rather than focusing on any one of these algorithms in particular, we turn our efforts to developing a comprehensive understanding of how the NEF may be extended to implement all of them.

Chapter 5

Extensions to the Neural Engineering Framework

Rapid growth in the field of neuromorphic engineering—over the past 30 years (Mead and Mahowald, 1988)—has accelerated the production of a large variety of neuromorphic architectures, each with distinct constraints and trade-offs that impact the degree of programmability and performance as a function of energy-consumption (section 2.3). Among these considerations, are issues involving: discretization (in time), quantization (truncation of both neural states and weight matrices), connectivity constraints, memory constraints, volume of spike traffic, external input-output bandwidth and delays, thermal variability, transistor mismatch, conversion of analog signals to digital pulses, and the introduction of higher-order dynamics throughout. The NEF already solves many of these problems in theory with varying degrees of success in practice. The focus of this chapter is to expose our novel contributions to solving these problems in the context of extending the scope of Principle 3. The goal is to bolster up the NEF with a variety of additional methods and techniques for analyzing and leveraging spiking dynamical computations.

5.1 Synapses

This section has been adapted from the works of Voelker and Eliasmith (2016, patent pending), Voelker et al. (2017a), Voelker and Eliasmith (2017b), Voelker and Eliasmith (2018), and Voelker and Eliasmith (2019, patent pending). Here, we focus on a dynamical primitive of SNNs known as the synapse model, $h(t)$, and address many of the differences that emerge between the idealization made by the NEF and the realities imposed by neuromorphic hardware (and often too, the brain). This leads to several high-level results that exploit their higher-order dynamics, account for time-constant mismatch, leverage their heterogeneity, and solve for optimal filters.

5.1.1 Linear transfer functions

We build a comprehensive theory, adapted from Voelker and Eliasmith (2018), for including arbitrary linear synapse models in the neural architecture. This is applied to the case of implementing linear systems, but many of these same techniques also carry over to later sections in the case of nonlinear systems.

Consider the linear time-invariant (LTI) system (repeating equation 2.12, for clarity):

$$\begin{aligned} \dot{\mathbf{x}}(t) &= A\mathbf{x}(t) + B\mathbf{u}(t) \\ \mathbf{y}(t) &= C\mathbf{x}(t) + D\mathbf{u}(t). \end{aligned} \tag{5.1}$$

This state-space model is related to its *transfer function*, $F(s)$, which uniquely characterizes its input-output response in the Laplace domain according to

$$F(s) = \mathcal{L}\left\{f(t)\right\}(s) = \frac{\mathcal{L}\left\{\mathbf{y}(t)\right\}(s)}{\mathcal{L}\left\{\mathbf{u}(t)\right\}(s)} = \frac{\mathbf{Y}(s)}{\mathbf{U}(s)},$$

by the following equation (Brogan, 1991):

$$F(s) = C(sI - A)^{-1}B + D. \tag{5.2}$$

Now let $F(s)$ be the transfer function for the linear dynamics that we wish to implement (derived by equations 5.1 and 5.2), and let $H(s)$ be the transfer function for an arbitrary linear synapse model ($H(s) = \mathcal{L}\{h(t)\}$). As stated in section 2.2.3, introducing the synapse model means replacing the integrator (s^{-1}) with $H(s)$. This is equivalent to replacing s with $H(s)^{-1}$. Notably, substituting $H(s)$ for the integrator results in the transfer function $F\left(H(s)^{-1}\right)$, which no longer implements the original, desired dynamics $F(s)$. However, we would like to ensure that the new dynamics match the originally specified $F(s)$. The key insight is that we can determine a new function, $F^H(s)$, such that $F^H\left(H(s)^{-1}\right) = F(s)$. That is, we can solve for a function that provides the original dynamics when implemented using the transfer function $H(s)$ as the dynamical primitive. This is formalized by the following definition:

Definition 5.1.1. A function $F^H(s)$ *maps F onto H* if and only if it satisfies:

$$F^H\left(\frac{1}{H(s)}\right) = F(s). \tag{5.3}$$

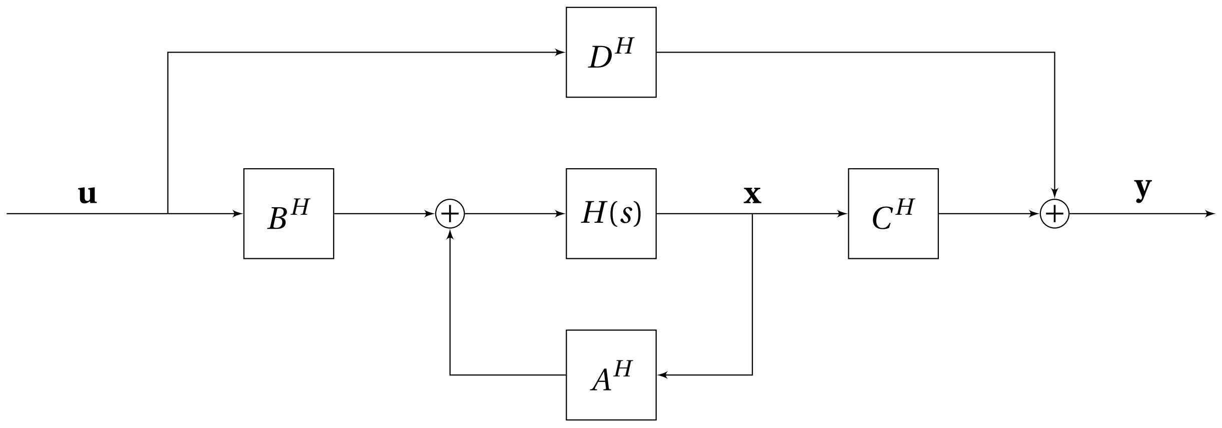


Figure 5.1: Block diagram for an LTI system, equivalent to Figure 2.1, with the integrator replaced by a more general linear filter $H(s)$. The state-space model $\left(A^H, B^H, C^H, D^H\right)$ is obtained from some transfer function $F^H(s)$ that maps F onto H, as defined in the text. This generalizes Figure 2.2 to arbitrary linear synapse models. Reproduced from Voelker and Eliasmith (2018, Figure 7).

This definition compactly expresses the notion of a "change of dynamical primitive", in that $F(s)$ is mapped from the canonical primitive, s^{-1}, onto some new primitive, $H(s)$. Trivially, $F(s)$ maps itself onto s^{-1}. Non-trivial examples are given below.

Once we identify a $F^H(s)$ that maps F onto H, any state-space model $\left(A^H, B^H, C^H, D^H\right)$ that satisfies

$$F^H(s) = C^H\left(sI - A^H\right)^{-1} B^H + D^H \tag{5.4}$$

will implement the desired dynamics when using $H(s)$ as the dynamical primitive, by equations 5.2 and 5.3 (see Figure 5.1).

Therefore, supposing $F^H(s)$ satisfies definition 5.1.1, and that it is convertible to a state-space model (equation 2.12), then Figure 5.1 is just another form of Figure 2.1, but with the integrator replaced by the synapse. Note that this construction, on its own, does not specify how to find a satisfying $F^H(s)$, nor whether such a function exists, nor whether it can be converted to a state-space model. We provide several examples leading to such a specification at the end of this section.

Before proceeding, we remark that the above theory directly carries over from the continuous-time domain to the discrete-time domain. The discrete-time formulation of an LTI system is similar to equation 2.12, but increments time in steps of length dt:

$$\begin{aligned} \mathbf{x}[t+\mathtt{dt}] &= \bar{A}\mathbf{x}[t] + \bar{B}\mathbf{u}[t] \\ \mathbf{y}[t] &= \bar{C}\mathbf{x}[t] + \bar{D}\mathbf{u}[t], \end{aligned} \tag{5.5}$$

where the discrete state-space model $\left(\bar{A}, \bar{B}, \bar{C}, \bar{D}\right)$ fully defines the system. The discrete-time equivalent to the Laplace transform is the $\mathcal{Z}$*-transform*, named for its use of the variable z to denote the complex frequency domain. In this domain, z^{-1} plays the role of s^{-1}, by performing a discrete shift forwards, one step in time (i.e., a delay of one time-step), instead of integration. A well-known result is that the transfer function of this discrete LTI system—defined as the ratio of the $\mathcal{Z}$-transform of the output to the $\mathcal{Z}$-transform of the input—is equal to $F(z) = \bar{C}(zI - \bar{A})^{-1}\bar{B} + \bar{D}$, analogously to equation 5.2. Consequently, all of the previous discussion carries over to discrete LTI systems. In particular, for a discrete synapse expressed using the $\mathcal{Z}$-transform, $H(z)$, we have the analogous definition:

Definition 5.1.2. A function $F^H(z)$ *maps* F *onto* H if and only if it satisfies:

$$F^H\left(\frac{1}{H(z)}\right) = F(z). \tag{5.6}$$

Given some $F^H(z)$ that maps F onto H, any state-space model $\left(\bar{A}^H, \bar{B}^H, \bar{C}^H, \bar{D}^H\right)$ that satisfies

$$F^H(z) = \bar{C}^H\left(zI - \bar{A}^H\right)^{-1}\bar{B}^H + \bar{D}^H \tag{5.7}$$

will implement the desired dynamics $F(z)$ when using $H(z)$ as the dynamical primitive. Hence, the task of determining $F^H(\cdot)$ is identical for both continuous- and discrete-time domains—it is only $F(\cdot)$ and $H(\cdot)$ that differ.

Continuous lowpass synapse

The first example we consider demonstrates that our new theory recovers the standard form of Principle 3 from the NEF (see section 2.2.3). For the case of a continuous-time first-order lowpass filter (equation 4.11), $H(s) = (\tau s + 1)^{-1}$, let:

$$\begin{aligned} F^H(s) &:= C\left(sI - (\tau A + I)\right)^{-1}(\tau B) + D \\ &= C\left(\left(\frac{s-1}{\tau}\right)I - A\right)^{-1} B + D. \end{aligned}$$

Then,

$$\begin{aligned} F^H(\tau s + 1) &= C\left(\left(\frac{(\tau s+1)-1}{\tau}\right)I - A\right)^{-1} B + D \\ &= F(s), \end{aligned}$$

which satisfies definition 5.1.1. Therefore, by equation 5.4,

$$\begin{aligned} A^H &= \tau A + I, \qquad & C^H &= C, \\ B^H &= \tau B, \qquad & D^H &= D. \end{aligned} \tag{5.8}$$

This completes our novel proof of Principle 3 from section 2.2.3.

Discrete lowpass synapse

When simulating any NEF network on a digital computer, we necessarily use time-steps of finite length $\mathtt{dt} > 0$ to advance the state of the network, updating at discrete moments in time (Bekolay et al., 2014). For instance, Nengo currently uses a default of $\mathtt{dt} = 1\,\mathrm{ms}$, and implements a zero-order hold (ZOH) discretization of the synaptic filter. Implementing ZOH means that all continuous-time signals are held constant within each time-step. This discretization of equation 4.11 gives:

$$H(z) = \frac{1-a}{z-a}, \quad a := e^{-\frac{\mathtt{dt}}{\tau}}.$$

If our desired transfer function is expressed in continuous-time, $F(s)$, then we should also discretize it to $F(z) = \bar{C}(zI - \bar{A})^{-1}\bar{B} + \bar{D}$, with the same time-step, and again using ZOH discretization for consistency. Let,

$$\begin{aligned} F^H(z) &:= \bar{C}\left(zI - \frac{1}{1-a}\left(\bar{A} - aI\right)\right)^{-1}\left(\frac{1}{1-a}\bar{B}\right) + \bar{D} \\ &= \bar{C}\left((z(1-a)+a)\,I - \bar{A}\right)^{-1}\bar{B} + \bar{D}. \end{aligned}$$

Then,

$$\begin{aligned} F^H\left(\frac{z-a}{1-a}\right) &= \bar{C}\left(\left(\frac{z-a}{1-a}(1-a)+a\right)I - \bar{A}\right)^{-1}\bar{B} + \bar{D} \\ &= F(z), \end{aligned}$$

which satisfies definition 5.1.2. Therefore, by equation 5.7,

$$\begin{aligned} \bar{A}^H &= \frac{1}{1-a}\left(\bar{A} - aI\right), & \bar{C}^H &= \bar{C}, \\ \bar{B}^H &= \frac{1}{1-a}\bar{B}, & \bar{D}^H &= \bar{D}, \end{aligned} \tag{5.9}$$

provides an exact implementation of the desired system for digital architectures, regardless of the simulation time-step (assuming ZOH).

Delayed continuous lowpass synapse

Next, we consider a continuous-time first-order lowpass filter with a time-delay of λ:

$$H(s) = \frac{e^{-\lambda s}}{\tau s + 1}. \tag{5.10}$$

This same model has been proposed by Roth and van Rossum (2009, equation 6.2) as a more realistic alternative to equation 4.11, that includes an axonal transmission delay of length λ (on the order of τ) to account for the finite-velocity propagation of action potentials. By commutativity of convolution, modeling the delay in the synapse (as in equation 5.10) is equivalent to modeling the delay in spike propagation. Equation 5.10 may also be used to account for feedback delays within some broader setting (e.g., when the feedback term is computed via some delayed system).

In the mammalian cortex, λ is known to range between 0.1–44 ms while being precise and reproducible (Lagorce and Benosman, 2015). In analog circuitry, λ grows in proportion to the square of the wire length, and is necessarily introduced by the pulse-extension of digital-to-analog converters (Voelker et al., 2017a). This delay term may also be configured on digital architectures such as Loihi (Davies et al., 2018).

Letting $d := \frac{\lambda}{\tau} e^{\frac{\lambda}{\tau}}$, and $W_0(\cdot)$ denote the principal branch of the Lambert-W function (Corless et al., 1996), we invert $y := H(s)^{-1}$ as follows:

$$\begin{aligned} & & y &= (\tau s + 1)\, e^{\lambda s} \\ &\iff & \frac{\lambda}{\tau} e^{\frac{\lambda}{\tau}} y &= \left(\lambda s + \frac{\lambda}{\tau}\right) e^{\lambda s + \frac{\lambda}{\tau}} \\ &\iff & W_0(dy) &= \lambda s + \frac{\lambda}{\tau} \\ &\iff & \frac{1}{\lambda} W_0(dy) - \frac{1}{\tau} &= s, \end{aligned}$$

where the second-last line assumes that $|\eta| < \pi$ and $\lambda \mathrm{Re}\,[s] + \frac{\lambda}{\tau} > -\eta \cot \eta$, where $\eta := \lambda \mathrm{Im}\,[s]$, in order for $\lambda s + \frac{\lambda}{\tau}$ to be within the principal branch (Corless et al., 1996, equation 4.4).[1] Therefore,

$$\begin{aligned} & F^H(s) := F\left(\frac{1}{\lambda} W_0(ds) - \frac{1}{\tau}\right) \quad (5.11) \\ \implies \quad & F^H(H(s)^{-1}) = F^H(y) = F(s). \end{aligned}$$

As a demonstration of how we might use this mapping, suppose the desired transfer function for our system is a time-delay of θ seconds, $F(s) = e^{-\theta s}$ (equation 6.2). In this setting, we are attempting to "amplify" a delay of λ seconds in the synapse into a system delay of θ seconds at the network level. Letting $c := e^{\frac{\theta}{\tau}}$ and $r := \frac{\theta}{\lambda}$, and then substituting $F(s)$ into equation 5.11, provides the required function:

$$F^H(s) = \exp\left\{-\theta\left(\frac{1}{\lambda} W_0(ds) - \frac{1}{\tau}\right)\right\} = c \exp\left\{-r W_0(ds)\right\} = c\left(\frac{W_0(ds)}{ds}\right)^r.$$

This may be numerically converted into a state-space model, of arbitrarily chosen dimensionality q, via the $[q-1/q]$ Padé approximants of the following Maclaurin series:

$$F^H(s) = cr \sum_{i=0}^{\infty} \frac{(i+r)^{i-1}}{i!}(-ds)^i. \quad (5.12)$$

To our knowledge, there is no closed-form expression for the Padé approximants of equation 5.12, but there are methods to compute them accurately and in $\mathcal{O}\left(q^2\right)$ time (Sidi, 2003). Given these approximants, we may follow the same procedure that we use in section 6.1.1 (see equation 6.3) to obtain an LTI system in the form of equation 2.12. In section 6.2.7, we use this approach to improve the accuracy of the delay network demonstrated in section 6.1.1. We remark that each of d, c, and r are dimensionless (i.e., unitless) constants that can be used to relate measurable properties of a biological system that may be governed by this description to the necessary network-level computations.

Coordinate transformation

Before proceeding with the general solution, we introduce some useful intermediate results:

[1]A simpler (but only sufficient) condition is $\mathrm{Re}\,[s] \geq -\frac{1}{\tau}$ and $|\mathrm{Im}\,[s]| < \frac{\pi}{2\lambda}$. Thus, it suffices to consider input frequencies $< \frac{1}{4\lambda}$ Hz.

Lemma 5.1.1 (Transforming coordinates between dynamical systems)**.** *Let* $\mathbf{f}(t)$ *and* $\mathbf{g}(t)$ *be infinitely time-differentiable signals, that are related by:*

$$\mathbf{f} = \sum_{i=0}^{\infty} c_i \mathbf{g}^{(i)}, \tag{5.13}$$

for some coordinates $\{c\}_i$, $c_0 \neq 0$. *Then this dynamical system is equivalent to:*

$$\mathbf{g} = \sum_{i=0}^{\infty} b_i \mathbf{f}^{(i)}, \tag{5.14}$$

where the coordinates $\{b\}_i$ *are defined by the recursive transformation:*

$$b_i = c_0^{-1} \begin{cases} 1 & i = 0 \\ -\sum_{j=0}^{i-1} b_j c_{i-j} & i \geq 1. \end{cases} \tag{5.15}$$

Corollary 5.1.1.1. *Let:*

$$H_c(s) = \frac{1}{\sum_{i=0}^{\infty} c_i s^i}, \quad H_b(s) = \frac{1}{\sum_{i=0}^{\infty} b_i s^i},$$

where b is defined by equation 5.15. Then equation 5.13 is equivalent to $F(s)H_c(s) = G(s)$ *and similarly equation 5.14 is equivalent to* $G(s)H_b(s) = F(s)$*, and moreover:*

$$H_c(s)H_b(s) = 1, \tag{5.16}$$

hence $H_c(s)$ *and* $H_b(s)$ *are each other's reciprocals. Furthermore, the coordinate transformation equation 5.15 is its own inverse.*

Proof. Noting that $\mathbf{g}^{(0)} = \mathbf{g}$, rearrange equation 5.13 as:

$$\mathbf{g} = c_0^{-1}\mathbf{f} + \sum_{i=1}^{\infty} (-c_0^{-1} c_i)\mathbf{g}^{(i)}. \tag{5.17}$$

Differentiate each side an infinite number of times, to obtain the following set of equations that hold for all $j \in \mathbb{N}$:

$$\mathbf{g}^{(j)} = c_0^{-1}\mathbf{f}^{(j)} + \sum_{i=1}^{\infty} (-c_0^{-1} c_i)\mathbf{g}^{(i+j)}. \tag{5.18}$$

Now recursively substitute equation 5.18 into equation 5.17 for all occurrences of $\mathbf{g}^{(i)}$, $i \geq 1$ until we are only left with $\mathbf{f}^{(j)}$ terms for $j \leq i$, and take the limit as $i \to \infty$. This is equivalent to treating equation 5.18 as a recursive function in j and then evaluating $\mathbf{g}^{(0)}$. Although this is infinitely generative from a bottom-up perspective, we are 'allowed' to do this because each substitution of equation 5.18 increases the order of $\mathbf{g}$ by 1 (hence it terminates top-down). Another way to view this is we pick some finite k in order to gather all occurrences of $\mathbf{f}^{(k)}$ in the infinite expansion. After repeating this for all $k \to \infty$, we get something of the form:

$$\mathbf{g} = \sum_{k=0}^{\infty} \tilde{b}_k \mathbf{f}^{(k)}.$$

Now, all that remains is to show that $\tilde{b}_k = b_k$ from equation 5.15 in order to match equation 5.14. We do this inductively in a way that parallels the recursive structure of the substitution procedure.

For the base case ($k = 0$) it should be clear that $\tilde{b}_0 = c_0^{-1} = b_0$ since this is the only way to construct $\mathbf{f}^{(0)}$. For the inductive case ($k \geq 1$), the only way to construct $\mathbf{f}^{(k)}$ is through the substitution of $\mathbf{g}^{(k)}$ in equation 5.18. Furthermore, this occurs for all $0 \leq j \leq k-1$. In particular, for every such j, we have the coefficient $(-c_0^{-1}c_i)$ multiplied by $\mathbf{g}^{(i+j)}$ to yield $c_0^{-1}\mathbf{f}^{(k)}$ where $k = i + j$. Finally, this occurs $\tilde{b}_j c_0$ times since it is mirrored by each occurrence of $\mathbf{f}^{(j)}$. Putting this all together, we get $\tilde{b}_k = \sum_{j=0}^{k-1} -c_0^{-1} c_{k-j} \tilde{b}_j = c_0^{-1} \sum_{j=0}^{k-1} -b_j c_{k-j} = b_k$ (by the inductive hypothesis). □

The coordinate transformation equation 5.15 is equivalent to taking the Padé approximants (Padé, 1892) for the special case of when the numerator is $p = 0$. To be precise,

$$\sum_{i=0}^{q} c_i s^i \approx \frac{1}{\sum_{i=0}^{q} b_i s^i}$$

has the same optimal approximation from equation 5.15 as taking the $[0/q]$ Padé approximants:

$$\{b\}_{i=0\ldots q} = [0/q] \sum_{i=0}^{q} c_i s^i.$$

This can be seen through equation 5.16 and verified by comparing the algorithm for the extended Euclidean algorithm for the polynomial GCD, in this special case, to the algorithm for equation 5.15. That is, they both implement long division. We remark that the coordinate transformation equation 5.15—that is, the algorithm for finding the Padé approximants when $p = 0$—is its own inverse. That is, we may convert back and forth using the same transformation without any loss of information.

We may also interpret equation 5.15 as a discrete dynamical system by realizing that b is the discrete convolution of itself with c. With some work, we can show that this is the same as stating that $\mathcal{Z}\{b\}\mathcal{Z}\{c\} = 1$, or equivalently in the time-domain, $\{b\}_{i=0\ldots q}$ is the impulse-response of the following discrete transfer function:

$$H^q_{c\to b}(z) = \frac{z^q}{\sum_{i=0}^q c_i z^{q-i}},$$

and likewise, $H^q_{b\to c}(z)$ has the impulse-response $\{c\}_{i=0\ldots q}$. Furthermore,

$$\lim_{q\to\infty} H^q_{c\to b}(z) H^q_{b\to c}(z) = 1,$$

analogously to equation 5.16. Lastly, we note that the above Lemma and Corollary also apply to the case where derivatives are replaced by discrete time-shifts and the Laplace transform is replaced by the $\mathcal{Z}$-transform.

General linear synapse

Finally, we consider the general class of all linear synapse models of the form:

$$H(s) = \frac{1 + \sum_{i=1}^p b_i s^i}{\sum_{i=0}^q \tilde{c}_i s^i}, \tag{5.19}$$

That is, a rational function with real polynomial coefficients of potentially infinite order. The only constraint is that $b_0 \neq 0$ (coefficients may be normalized to put it into this form where $b_0 = 1$). This encompasses the class of nearly all linear filters, and, as far as we know, all linear synapse models used in the literature. For instance, this includes the first-order lowpass synapse that is standard in the NEF. It also includes the second-order alpha synapse, $(\tau s + 1)^{-2}$ (Rall, 1967)—the convolution of two exponentials with identical time-constants—which is commonly used in biological models (Destexhe et al., 1994b; Mainen and Sejnowski, 1995; Koch and Segev, 1998; Destexhe et al., 1998; Roth and van Rossum, 2009). The alpha synapse essentially filters the spike-trains twice, to produce PSCs with finite (non-instantaneous) rise-times. In addition, equation 5.19 includes a generalization of the alpha synapse, the double-exponential synapse, $(\tau_1 s + 1)^{-1}(\tau_2 s + 1)^{-1}$ (Wilson and Bower, 1989)—the convolution of two exponentials with time-constants τ_1 and τ_2—which has different rise- and fall-times to account for the separate time scales of rapid transmitter binding followed by slow unbinding (Destexhe et al., 1994b; Häusser and Roth, 1997; Roth and van Rossum, 2009). The double-exponential is also a suitable model to account for parasitic capacitances in neuromorphic hardware (Voelker et al., 2017a). Furthermore, equation 5.19 includes a higher-order generalization of the alpha synapse, the gamma kernel, $(\mu^{-1}s+1)^{-k}$ (De Vries and Principe, 1992, equation 19). Finally, this class contains more exotic models, such as the Q-bandpass synapse model, $(\omega^2 s^2 + (\omega Q)^{-1} s + 1)^{-1}$, where ω is the peak frequency in radians per second, and Q is inversely proportional to the bandwidth. Such synapses have been used to model bandpass filtering from mechanoreceptors in the fingertip (Friedl et al., 2016) or from rods in the retina (Armstrong-Gold and Rieke, 2003).

As well, in the following section we demonstrate that equation 5.19 may be used to model synapses with pure delays, which proves a complete connection to ZOH discretization. This same technique permits us to model, for instance, synapses with pulse-extended responses (Voelker et al., 2017a). Similarly, the delayed lowpass (equation 5.10) may be expressed in the form of equation 5.19. Finally, any linear combinations of the aforementioned synapse models will also be a member of this class. Nonlinear synapse models, such as conductance-based synapses (e.g., Destexhe et al., 1994a, equation 6), are the active subject of study in the NEF (Stöckel et al., 2017b; Stöckel et al., 2018).

The first step is to apply Corollary 5.1.1.1 to the numerator $\{b\}_{i=0}^{p}$ in order to rewrite the transfer function in the following form:

$$H(s) = \frac{1}{\sum_{i=0}^{k} c_i s^i}, \tag{5.20}$$

where k is potentially infinite. Now we state and prove the main theorem of this section.

Theorem 5.1.2 (General state-space solution to leverage postsynaptic currents)**.** *To map F onto equation 5.20, we let:*

$$\begin{aligned} A^H &= \sum_{i=0}^{k} c_i A^i, & C^H &= C, \\ B^H &= \left(\sum_{j=0}^{k-1} s^j \sum_{i=j+1}^{k} c_i A^{i-j-1} \right) B, & D^H &= D. \end{aligned} \tag{5.21}$$

This satisfies the required identity, $F^H(H(s)^{-1}) = F(s)$ (equation 5.3).

Proof.

$$\begin{aligned}
\left(\sum_{j=0}^{k-1} s^j \sum_{i=j+1}^{k} c_i A^{i-j-1}\right)(sI-A) &= \sum_{j=0}^{k-1}\sum_{i=j+1}^{k} s^j c_i A^{i-j-1}(sI-A) \\
&= \sum_{i=0}^{k}\sum_{j=0}^{i-1} s^j c_i A^{i-j-1}(sI-A) \\
&= \sum_{i=0}^{k} c_i \left(\sum_{j=0}^{i-1} s^{j+1}A^{i-(j+1)} - s^j A^{i-j}\right) \\
&= \sum_{i=0}^{k} c_i \left(s^i A^{i-i} - s^0 A^{i-0}\right) \\
&= \left(\sum_{i=0}^{k} c_i s^i\right) I - \sum_{i=0}^{k} c_i A^i \\
&= H(s)^{-1} I - A^H.
\end{aligned}$$

We then complete the proof by substituting this result and the state-space model into our expression for the mapped transfer function from equation 5.4:

$$\begin{aligned}
F^H(H(s)^{-1}) &= C^H(H(s)^{-1}I - A^H)^{-1}B^H + D^H \\
&= C(H(s)^{-1}I - A^H)^{-1}\left(\sum_{j=0}^{k-1} s^j \sum_{i=j+1}^{k} c_i A^{i-j-1}\right)B + D \\
&= C(sI-A)^{-1}B + D \\
&= F(s).
\end{aligned}$$

□

An alternative derivation may also be found in section 5.1.3. An interesting insight gained with this solution is that it is not, in general, the case that B^H is time-invariant, since it depends on the time-differential, s, for values of $k \geq 2$. As a result, solutions will often not be a typical state-space model in the form of equation 2.12.

Nevertheless, such solutions can still be implemented, as is, in practice. Since s^j is the j^{th}-order differential operator, this form of B^H states that we must supply the j^{th}-order input derivatives $\mathbf{u}^{(j)}$, for all $j = 1 \ldots k-1$. When $k = 1$, this is trivial (no derivatives are needed). For $k \geq 2$, let us first define $B_j^H := \left(\sum_{i=j+1}^{k} c_i A^{i-j-1}\right)B$. Then equation 5.21 shows that the ideal state-space model must implement the input transformation as a linear combination of input derivatives, $\sum_{j=0}^{k-1} B_j^H \mathbf{u}^{(j)}$. If the derivatives of the input are available to the model, then the $F^H(s)$ we described may be used to precisely implement the desired $F(s)$.

However, if the required derivatives are not included in the neural representation, then it is natural to use a ZOH method by assuming $\mathbf{u}^{(j)} = 0$, for all $j = 1 \ldots k-1$:

$$B^H = B_0^H = \left(\sum_{i=1}^{k} c_i A^{i-1} \right) B, \tag{5.22}$$

with A^H, C^H, and D^H as before (see equation 5.21). This is now an equivalent model to equation 5.21 assuming ZOH, and in the form of the standard state-space model (equation 2.12). In section 5.1.2, we characterize the dynamics of this new system in terms of F and H (independently of the chosen state-space model) for general inputs. We find that equation 5.22 adds $k-1$ new dimensions, for every dimension in F, to the state underlying the resulting dynamical system. In the following section, we show that the our results yield a novel derivation of ZOH discretization for LTI systems.

To our knowledge, this specific state-space architecture has not been explored in theory or in practice. Given that equation 5.21 requires the input derivatives to accurately compute low-dimensional network-level dynamics, this strongly suggests that it is important to represent and compute derivatives in neural systems. Methods of computing derivatives have been explored by Tripp and Eliasmith (2010) within the NEF. As well, it has long been suggested that adaptive neural mechanisms (e.g., synaptic depression or spike-rate adaptation) may also play a fundamental role in computing such derivatives (Abbott and Regehr, 2004; Lundstrom et al., 2008). However, there has typically been less emphasis placed on understanding temporal differentiation in comparison to other temporal operators, such as integration (Tripp and Eliasmith, 2010).

Finally, we again note that these results translate directly to the discrete-time domain. The required state-space matrices are the same as in equation 5.21, but with $(\bar{c}_i)$ corresponding to the coefficients of $H(z)$, bars affixed to each state-space matrix, and z substituted for s in $\bar{B}^H$. However, the z^j operator is a shift backwards by j time-steps (i.e., an acausal lookahead), and so the ZOH assumption is instead $\mathbf{u}[t + j\texttt{dt}] = \mathbf{u}[t]$ for all $j = 1 \ldots k-1$. Thus, the discrete analog to equation 5.22 is:

$$\bar{B}^H = \left(\sum_{j=0}^{k-1} \sum_{i=j+1}^{k} \bar{c}_i \bar{A}^{i-j-1} \right) \bar{B}. \tag{5.23}$$

Relationship to discretization

There is an interesting connection between definition 5.1.1 and the well-known problem of discretizing a linear dynamical system. A discrete LTI system (equation 5.5) is identical to a continuous LTI system (equation 2.12), with three adjustments: (1) the integrator s^{-1} is replaced by a time-delay of $\texttt{dt}$ seconds, (2) the input signal is sampled every $\texttt{dt}$ seconds, and (3) the output signal is sampled every $\texttt{dt}$ seconds. Focusing on point 1, this is precisely the notion captured by definition 5.1.1 with respect to:

$$H(s) = e^{-\texttt{dt}s} = \frac{1}{e^{\texttt{dt}s}} = \frac{1}{\sum_{i=0}^{\infty} \frac{\texttt{dt}^i}{i!} s^i},$$

by the Maclaurin series of e^x. That is, a synapse model that represents a pure delay of $\lambda = \texttt{dt}$ without any additional filtering. This is in the form of equation 5.20 with $c_i = \dfrac{\texttt{dt}^i}{i!}$ as $k \to \infty$. These coefficients are also the $[0/\infty]$ Padé approximants of $\displaystyle\sum_{i=0}^{\infty} \frac{(-\texttt{dt})^i}{i!} s^i$, and likewise the result of the coordinate transformation equation 5.15.

If we make the ZOH assumption—that the input signal is held piecewise-constant over each continuous-time interval of length $\texttt{dt}$—then $\mathbf{u}^{(j)} = 0$ for all $j \geq 1$. Therefore, by equation 5.22, an equivalent state-space model is:

$$\begin{aligned} A^H &= \sum_{i=0}^{k} c_i A^i = \sum_{i=0}^{k} \frac{(A\texttt{dt})^i}{i!} = e^{A\texttt{dt}}, & C^H &= C, \\ B^H &= \left(\sum_{i=1}^{k} c_i A^{i-1} \right) B = A^{-1} \left(A^H - I \right) B, & D^H &= D, \end{aligned}$$

which is precisely the discrete state-space model $(\bar{A}, \bar{B}, \bar{C}, \bar{D})$ obtained by ZOH discretization (Brogan, 1991). The discrete signals in equation 5.5 correspond with points 2 and 3.

This connection helps to illustrate the scope and consistency of our theory. In particular, the important procedure of discretizing linear state-space models may be viewed as a special case of accounting for changes in dynamical primitives. Furthermore, as one should hope, the ZOH assumption recovers the correct result, which is normally proven by integrating the linear differential equations over the interval $[0, \texttt{dt}]$.

5.1.2 Approximating derivatives

In the previous section, we discovered that higher-order synapses must be driven with higher-order derivatives of the state-vector, in order to cancel out the additional rounds of integration and successfully encode $\mathbf{x}$. These computations could be facilitated by adaptive mechanisms in more biologically-realistic neuron models (Lundstrom et al., 2008), approximated at the population level (Tripp and Eliasmith, 2010), computed using models of the system's Jacobian, modelled from the input's dynamics, or obtained directly from sensors that represent such derivatives. If none of these cases are possible, we resort to the following analysis of what can be done.

Zero-order assumption

This section has been adapted from Voelker and Eliasmith (2018, appendix A.4). The approach that we take here is to apply the ZOH assumption and pretend that the higher-order derivatives are all zero. We then examine the consequences of making this assumption.

Consider the F^H determined by equation 5.22, when the derivatives of the input signal are completely inaccessible. Let $\hat{F}(s) := F^H(H(s)^{-1})$ be the dynamics of the implemented system for this particular F^H. Due to the ZOH assumption, $\hat{F} \neq F$ in general, and so F^H does not technically map F onto H (see definition 5.1.1). To be clear, $\hat{F}(s) = F(s)$ for $s = 0$, but not necessarily for $s \neq 0$. Thus, it is important to characterize the difference between $\hat{F}$ and F for general inputs.

To do so, we can examine the *poles* of the transfer function $F(s) = \dfrac{\mathcal{C}(s)}{\mathcal{D}(s)}$, which are defined as the complex roots of $\mathcal{D}(s)$. The poles of a system fully define the dynamics of its state (up to a change of basis). For instance, each pole λ corresponds to a time-constant of $\left(-\lambda^{-1}\right)$ in the system's dynamics. A system is exponentially stable if and only if $\mathrm{Re}\,[\lambda] < 0$ for all poles $\lambda \in \mathbb{C}$, since all time-constants will be positive. Furthermore, $\lambda \in \mathrm{sp}(A)$ if and only if λ is a pole, where $\mathrm{sp}(A)$ denotes the eigenvalues of the state-space matrix A.[2] Therefore, we may characterize the poles of $\hat{F}$ in terms of the poles of F, in order to understand the behaviour of the implemented system.

We begin by deriving the poles of F^H, recalling that $A^H = \sum_{i=0}^{k} c_i A^i$. Let $\mathbf{v} \neq \mathbf{0}$ be an eigenvec-

[2]Note that $\mathrm{sp}(A) = \mathrm{sp}(TAT^{-1})$ for any invertible matrix T with the same shape as A.

tor of A with eigenvalue λ, so that:

$$A\mathbf{v} = \lambda\mathbf{v} \quad \implies \quad A^H\mathbf{v} = \sum_{i=0}^{k} c_i A^i \mathbf{v} = \left(\sum_{i=0}^{k} c_i \lambda^i\right)\mathbf{v}.$$

Hence, $\text{sp}\left(A^H\right) = \left\{\sum_{i=0}^{k} c_i \lambda^i : \lambda \in \text{sp}(A)\right\}$ is the full set of eigenvalues for A^H. This is also true for equation 5.21, since A^H is identical, but we do not need this fact.

The denominator of $\hat{F}$ may now be written as $\prod_{\lambda \in \text{sp}(A)} \left(H(s)^{-1} - \sum_{i=0}^{k} c_i \lambda^i\right)$. Therefore, the poles of $\hat{F}$ are the roots of the q polynomials:

$$\mathcal{P}(\phi) := \sum_{i=0}^{k} c_i \left(\phi^i - \lambda^i\right), \quad \lambda \in \text{sp}(A).$$

A trivial set of roots are $\phi = \lambda$, and thus each pole of F is also a pole of $\hat{F}$, as desired. However, for a synapse of order k, there will also be $k-1$ additional poles for every pole of F. For this system to behave as F given low-frequency inputs, we must have the old poles dominate the new poles. That is, we require $\text{Re}\left[\lambda\right] \gg \text{Re}\left[\phi\right]$ for all $\phi \neq \lambda$.

To provide a specific example, let us consider the double-exponential synapse, $H(s)^{-1} = (\tau_1 s + 1)(\tau_2 s + 1)$,

$$\begin{aligned}
\implies \qquad & \mathcal{P}(\phi) = \tau_1\tau_2\phi^2 + (\tau_1+\tau_2)\phi - (\tau_1\tau_2\lambda^2 + (\tau_1+\tau_2)\lambda) = 0 \\
\iff \qquad & \phi = \frac{-(\tau_1+\tau_2) \pm \sqrt{(\tau_1+\tau_2)^2 + 4\tau_1\tau_2\left(\tau_1\tau_2\lambda^2 + (\tau_1+\tau_2)\lambda\right)}}{2\tau_1\tau_2} \\
& \quad = \frac{-(\tau_1+\tau_2) \pm (2\tau_1\tau_2\lambda + (\tau_1+\tau_2))}{2\tau_1\tau_2}.
\end{aligned}$$

In this instance, the '+' case gives back the known poles, $\phi = \lambda$, and the '−' case provides the new poles, $\phi = \left(-\frac{\tau_1+\tau_2}{\tau_1\tau_2} - \lambda\right)$. Consequently, the poles of $\hat{F}$ are the poles of F, duplicated and reflected horizontally about the real line $-b$, where $b := \frac{\tau_1+\tau_2}{2\tau_1\tau_2}$ (and the imaginary components are unchanged).

Interestingly, b may be rewritten as $\left(\frac{\tau_1+\tau_2}{2}\right) / \sqrt{\tau_1^2\tau_2^2}$, which is the ratio of the arithmetic mean to the geometric mean of $\{\tau_1, \tau_2\}$, which may in turn be interpreted as a cross-entropy expression (Woodhouse, 2001). Regardless, $\hat{F}$ behaves like F when $\left(-\mathrm{Re}\left[\lambda\right]^{-1}\right) \gg b^{-1}$. We note that b^{-1} is largest for the case of the alpha synapse ($b^{-1} = \tau$), and smallest for the case of the lowpass synapse ($b^{-1} \to 0$ as $\tau_2 \to 0$).

Taylor-series approximation

Another way to avoid needing the derivatives is to work out a different approximation when attempting to satisfy definition 5.1.1. Here we use an example discussed in Voelker and Eliasmith (2016) and explore some consequences. This method is very similar to that of the delayed lowpass example in section 5.1.1.

Considering the alpha synapse:

$$H(s) = \frac{1}{(\tau s + 1)^2} = \frac{1}{\tau^2 s^2 + 2\tau s + 1}$$

we can observe that:

$$F^H\left(H(s)^{-1}\right) = F(s) \quad \iff \quad F^H(s) = F\left(\frac{\sqrt{s} - 1}{\tau}\right).$$

Hence this satisfies our requirement for F^H mapping F onto H. However, challenges arise when obtaining its state-space matrices via equation 5.2, as, in general, the expression $F\left(\frac{\sqrt{s}-1}{\tau}\right)$ is not necessarily proper and finite-dimensional (rational in s). For instance, with the integrator,

$$F(s) = 1/s \quad \iff \quad F^H(s) = \frac{\tau}{\sqrt{s} - 1}$$

is infinite-dimensional (irrational in s). We cope with this by taking the Taylor series approximation $(\sqrt{s} - 1)/\tau \approx (s-1)/(2\tau)$, which yields the first-order approximation:

$$F^H(s) \approx F\left(\frac{s-1}{2\tau}\right) = \frac{2\tau}{s-1}. \tag{5.24}$$

This system *does* have finite-dimensional state-space matrices that may be extracted via equation 5.2 to implement an approximation of the dynamical system with respect to the alpha synapse. But what is the approximation error? The main insight is that we may convert back to the world of $F(s)$ to reveal the dynamics that we've actually implemented:

$$F(s) = F^H\left(H(s)^{-1}\right) \approx \frac{2\tau}{(\tau s + 1)^2 - 1} = \frac{2\tau}{\tau s(\tau s + 2)} = \frac{1}{s\left(\frac{\tau}{2}s + 1\right)}.$$

This reveals that, instead of the integrator, we have implemented an integrator convolved with a lowpass with a time-constant of $\tau/2$. More generally, one can apply a Taylor series or the Padé approximants to any irrational solution to equation 5.3, and then work backwards through the mathematical machinery to see how the dynamics have been altered from their intended state.

Lagrange–Bürmann approximation

Our last approach will build upon the previous approach more generally, and without needing to explicitly work out a solution to equation 5.3. In general, we are trying to find an $F^H(s)$ such that $F^H(H(s)^{-1}) = F(s)$. However, it may not be clear how to do this for general $H(s)^{-1} = \sum_{i=0}^{k} c_i s^i$, since the inverse of $H(s)^{-1}$ does not necessarily exist. Fortunately, we can use a powerful tool named the *Lagrange–Bürmann inversion theorem* to approximate the inverse of $H(s)^{-1}$. Essentially it is a way to get the Taylor series of the inverse of some function around a point without explicitly constructing the inverse. Applying it here around the zero frequency:

$$\begin{aligned} f(s) &:= H(s)^{-1} = \sum_{i=0}^{k} c_i s^i \\ g_n &:= \lim_{s \to 0} \left[\frac{\partial^{n-1}}{\partial s^{n-1}} \left(\frac{s}{f(s) - c_0} \right)^n \right] \\ F^H(s) &\approx F\left(\sum_{n=1}^{\infty} g_n \frac{(s - c_0)^n}{n!} \right) \end{aligned} \tag{5.25}$$

in some neighbourhood of $s = 0$. Any finite truncation of the series g_n yields a finite-dimensional and proper approximation of the required $F^H(s)$ assuming the same of $F(s)$.

For example, returning to the alpha example, we have the first-order approximation:

$$\begin{aligned} f(s) &= (\tau s + 1)^2 \\ g_1 &= \lim_{s \to 0} \left(\frac{s}{s(\tau^2 s + 2\tau)} \right) = \frac{1}{2\tau} \\ \implies F^H(s) &\approx F\left(\frac{s-1}{2\tau} \right) \end{aligned}$$

which is identical to equation 5.24 in the case of the integrator. For more general $H(s)^{-1} = \sum_{i=0}^{k} c_i s^i$, we have $g_1 = \frac{1}{c_1}$ as our first approximant,

$$\implies F^H(s) \approx F\left(\frac{s - c_0}{c_1} \right) = C\left(sI - (c_1 A + c_0 I)\right)^{-1} (c_1 B) + D. \tag{5.26}$$

Although here we have shown just the first-order solution to make this concrete, the advantage of this method is that, through the use of symbolic differentiation, one may extend equation 5.25 out to any finite order for any rational and finite $F(s)$ and $H(s)$. In other words, this math does not need to be figured out by hand; this procedure may be automated up to any chosen dimension by Lemma 5.1.1 and equation 5.25. The main difficulty with this approach compared to others is that it will effectively expand the dimensionality of the state-vector that is to be represented (similar to equation 5.12).

5.1.3 Nonlinear state-spaces

This section has been adapted from the work of Voelker et al. (2017a), which in turn extends the work of Voelker and Eliasmith (2017b) in applying the NEF to the mixed-signal synapses of Braindrop (Neckar et al., 2019). Here we derive two theorems for nonlinear systems, by taking a subtly different perspective than that of the previous section. This generalizes the approach taken in Voelker and Eliasmith (2017b), which considered the special case of a pulse-extended (i.e., time-delayed) double-exponential synapse.

We wish to implement some desired nonlinear dynamical system,

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}, \mathbf{u}), \tag{5.27}$$

using equation 5.20 (rewritten from the form of equation 5.19) as the synaptic filter, $h(t)$. Letting $\mathbf{w}(t) = f^h(\mathbf{x}, \mathbf{u})$ for some recurrent function f^h and observing that $\mathbf{x}(t) = (\mathbf{w} * h)(t)$, we may express these dynamics in the Laplace domain:

$$\begin{aligned}
& & \frac{\mathbf{X}(s)}{\mathbf{W}(s)} &= \frac{1}{\sum_{i=0}^{k} c_i s^i} \\
&\iff & \mathbf{W}(s) &= \mathbf{X}(s) \sum_{i=0}^{k} c_i s^i = \sum_{i=0}^{k} c_i \left[s^i \mathbf{X}(s) \right] \\
&\iff & \mathbf{w}(t) &= \sum_{i=0}^{k} c_i \mathbf{x}^{(i)}
\end{aligned}$$

since s is the differential operator. This proves the following theorem:

Theorem 5.1.3 (Nonlinear solution to leverage continuous postsynaptic currents)**.** *Let the function computed along the recurrent connection be:*

$$f^h(\mathbf{x}, \mathbf{u}) = \sum_{i=0}^{k} c_i \mathbf{x}^{(i)} \tag{5.28}$$

where $\mathbf{x}^{(i)}$ *denotes the* i^{th} *time-derivative of* $\mathbf{x}(t)$*, and* c_i *are given by equation 5.20. Then the resulting dynamical system is precisely equation 5.27.*

For the discrete case, we begin with some desired nonlinear dynamics expressed over discrete time-steps:

$$\mathbf{x}[t+dt] = \bar{f}(\mathbf{x},\mathbf{u}), \tag{5.29}$$

using equation 5.20 (with s replaced by z; the coordinate transformation still applies) as the synaptic filter $h[t]$, followed by an analogous theorem:

Theorem 5.1.4 (Nonlinear solution to leverage discrete postsynaptic currents)**.** *Let the function computed along the recurrent connection be:*

$$\bar{f}^h(\mathbf{x},\mathbf{u}) = \sum_{i=0}^{k} \bar{c}_i \mathbf{x}^{[i]} \tag{5.30}$$

where $\mathbf{x}^{[i]}$ *denotes the* i^{th} *discrete forwards time-shift of* $\mathbf{x}$*, and* $\bar{c}_i$ *are likewise read from the denominator of* $H(z)$*. Then the resulting dynamical system is precisely equation 5.29.*

The proof for the discrete case is nearly identical. For sake of completeness, let $\mathbf{w}[t] = \bar{f}^h(\mathbf{x},\mathbf{u})$ for some recurrent function $\bar{f}^h$ and observe that $\mathbf{x}[t] = (\mathbf{w} * h)[t]$:

$$\begin{aligned} && \frac{\mathbf{X}(z)}{\mathbf{W}(z)} &= \frac{1}{\sum_{i=0}^{k} \bar{c}_i z^i} \\ \iff && \mathbf{W}(z) &= \mathbf{X}(z)\sum_{i=0}^{k} \bar{c}_i z^i = \sum_{i=0}^{k} \bar{c}_i \left[z^i \mathbf{X}(z) \right] \\ \iff && \mathbf{w}[t] &= \sum_{i=0}^{k} \bar{c}_i \mathbf{x}^{[i]} \end{aligned}$$

since z is the forwards time-shift operator. □

Continuous lowpass synapse

For standard Principle 3, we have $H(s) = \dfrac{1}{\tau s + 1} \implies k = 1,\ c_0 = 1$ and $c_1 = \tau$,

$$\implies f^h(\mathbf{x},\mathbf{u}) = c_0\mathbf{x}^{(0)} + c_1\mathbf{x}^{(1)} = \mathbf{x} + \tau\dot{\mathbf{x}} = \tau f(\mathbf{x},\mathbf{u}) + \mathbf{x}. \tag{5.31}$$

Note that equation 5.31 is consistent with equation 5.8 and with Principle 3 from the NEF.

Discrete lowpass synapse

For the discrete case of Principle 3, we have $H(z) = \frac{1-a}{z-a}$, where $a = e^{-\mathrm{dt}/\tau} \implies k = 1$, $\bar{c}_0 = -a(1-a)^{-1}$, $\bar{c}_1 = (1-a)^{-1}$,

$$\implies \bar{f}^h(\mathbf{x}, \mathbf{u}) = \bar{c}_0 \mathbf{x}^{[0]} + \bar{c}_1 \mathbf{x}^{[1]} = (1-a)^{-1}(\bar{f}(\mathbf{x}, \mathbf{u}) - a\mathbf{x}). \tag{5.32}$$

Note that equation 5.32 is consistent with equation 5.9.

Continuous double-exponential synapse

For the double exponential synapse:

$$H(s) = \frac{1}{(\tau_1 s + 1)(\tau_2 s + 1)} = \frac{1}{\tau_1 \tau_2 s^2 + (\tau_1 + \tau_2)s + 1} \tag{5.33}$$

$$\begin{aligned} \implies f^h(\mathbf{x}, \mathbf{u}) &= \mathbf{x} + (\tau_1 + \tau_2)\dot{\mathbf{x}} + \tau_1 \tau_2 \ddot{\mathbf{x}} \\ &= \mathbf{x} + (\tau_1 + \tau_2) f(\mathbf{x}, \mathbf{u}) + \tau_1 \tau_2 \left(\frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}} \cdot f(\mathbf{x}, \mathbf{u}) + \frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}} \cdot \dot{\mathbf{u}} \right). \end{aligned} \tag{5.34}$$

In the linear case, this simplifies to:

$$f^h(\mathbf{x}, \mathbf{u}) = \left(\tau_1 \tau_2 A^2 + (\tau_1 + \tau_2) A + I \right) \mathbf{x} + (\tau_1 + \tau_2 + \tau_1 \tau_2 A)\, B\mathbf{u} + \tau_1 \tau_2 B \dot{\mathbf{u}}.$$

General linear synapses

Consistent with what was found in section 5.1.1, Theorems 5.1.3 and 5.1.4 require that we differentiate the desired dynamical system. For the case of nonlinear systems, this means (if $k \geq 2$) determining the (possibly higher-order) Jacobians of f, as shown in equation 5.34. For the special case of LTI systems, we can determine this analytically to obtain a closed-form expression. By induction it can be shown that:

$$\mathbf{x}^{(i)} = A^i \mathbf{x} + \sum_{j=0}^{i-1} A^{i-j-1} B \mathbf{u}^{(j)}.$$

Then by expanding and rewriting the summations:

$$\begin{aligned} f^h(\mathbf{x},\mathbf{u}) &= \sum_{i=0}^{k} c_i \mathbf{x}^{(i)} \\ &= \sum_{i=0}^{k} c_i \left[A^i \mathbf{x} + \sum_{j=0}^{i-1} A^{i-j-1} B \mathbf{u}^{(j)} \right] \\ &= \underbrace{\left(\sum_{i=0}^{k} c_i A^i \right)}_{\text{Recurrent Matrix}} \mathbf{x} + \sum_{j=0}^{k-1} \underbrace{\left(\sum_{i=j+1}^{k} c_i A^{i-j-1} \right) B}_{\text{Input Matrices}} \mathbf{u}^{(j)}. \end{aligned} \tag{5.35}$$

The discrete case is identical:

$$\bar{f}^h(\mathbf{x},\mathbf{u}) = \underbrace{\left(\sum_{i=0}^{k} \bar{c}_i \bar{A}^i \right)}_{\text{Recurrent Matrix}} \mathbf{x} + \sum_{j=0}^{k-1} \underbrace{\left(\sum_{i=j+1}^{k} \bar{c}_i \bar{A}^{i-j-1} \right) \bar{B}}_{\text{Input Matrices}} \mathbf{u}^{[j]}. \tag{5.36}$$

This gives a matrix form for any LTI system with a k^{th} order synapse, provided we can determine $\mathbf{u}^{(j)}$ or $\mathbf{u}^{[j]}$ for $0 \le j \le k-1$. Again, equation 5.35 is consistent with equation 5.21 and equation 5.22, as is equation 5.36 with equation 5.23.

Pulse-extended double-exponential

Finally, we show how this all connects to the higher-order dynamics of the pulse-extended double-exponential synapses in Braindrop (Voelker et al., 2017a). Specifically, due to parasitic capacitances in the analog circuitry, a double-exponential (τ_1, τ_2) emerges, and, due to the pulse-extension from digital-to-analog (i.e., spike-to-synapse) conversion, a pulse is generated that is modelled as a box-filter with finite width, ϵ, and height, γ. This model is summarized by Voelker et al. (2017a, equation 8):

$$H(s) = \frac{\gamma \left(1 - e^{-\epsilon s}\right) s^{-1}}{(\tau_1 s + 1)(\tau_2 s + 1)} \tag{5.37}$$

Now we apply Lemma 5.1.1 in the following context:

$$\mathbf{f} = (\epsilon\gamma)^{-1} \left(\mathbf{x} + (\tau_1 + \tau_2)\dot{\mathbf{x}} + \tau_1\tau_2\ddot{\mathbf{x}}\right),$$

$$c_i = \frac{(-\epsilon)^i}{(i+1)!},$$

where $\mathbf{g}$ from the lemma is the signal that will be be used to drive the pulse-extender to satisfy $\mathbf{f} = \sum_{i=0}^{\infty} c_i \mathbf{g}^{(i)}$. We now apply a fourth-order truncation to the coefficients:

$$c_0 = 1, \quad c_1 = -\frac{\epsilon}{2}, \quad c_2 = \frac{\epsilon^2}{6}, \quad c_3 = -\frac{\epsilon^3}{24}, \quad c_i \approx 0 \quad \forall i \geq 4.$$

And then the coordinate transformation of equation 5.15 yields:

$$b_0 = 1, \quad b_1 = \frac{\epsilon}{2}, \quad b_2 = \frac{\epsilon^2}{4} - \frac{\epsilon^2}{6} = \frac{\epsilon^2}{12}, \quad b_3 = \frac{\epsilon^3}{24} - \frac{\epsilon^3}{12} + \frac{\epsilon^3}{24} = 0,$$

and so:

$$\begin{aligned} \mathbf{g} &= \sum_{i=0}^{\infty} b_i \mathbf{f}^{(i)} \\ &= \mathbf{f}^{(0)} + \frac{\epsilon}{2}\mathbf{f}^{(1)} + \frac{\epsilon^2}{12}\mathbf{f}^{(2)} + \ldots \\ &= (\epsilon\gamma)^{-1}\left(\mathbf{x} + (\tau_1 + \tau_2 + \epsilon/2)\dot{\mathbf{x}} + (\tau_1\tau_2 + (\epsilon/2)(\tau_1 + \tau_2) + \epsilon^2/12)\ddot{\mathbf{x}}\right) + \ldots \end{aligned} \tag{5.38}$$

which is a refined version of, and alternate derivation for, Voelker et al. (2017a, equation 11).

5.1.4 Accounting for mismatch

This section has been adapted from Voelker and Eliasmith (2017b) which abstracts the approach taken by Voelker et al. (2017a) to account for time-constant variability in Braindrop. Specifically, due to transistor mismatch in analog hardware, a variety of synapse models emerge with log-normal distributions in time-constants (Benjamin and Boahen, 2019). Assuming these can be accurately measured, we show how they can be accounted for by Principle 3.

Now the i^{th} postsynaptic neuron has a distinct synaptic filter, $h_i(t)$, shared across each of its synapses, as is the case in Braindrop (Neckar et al., 2019):

$$H_i(s) = \frac{1}{\sum_{j=0}^{k_i} c_{ij} s^j}. \tag{5.39}$$

Recalling the intuition behind Principle 3, our approach is to separately drive each synapse h_i with the required signal $f^{h_i}(\mathbf{x}, \mathbf{u})$ such that each PSC becomes a projection of the desired representation $\mathbf{x}$. Thus, the connection weights to the i^{th} neuron should be determined by solving the decoder optimization problems for $f^{h_i}(\mathbf{x}, \mathbf{u})$ using the methods of section 2.2.3 with respect to the synapse model h_i. This can be repeated for each postsynaptic neuron, to obtain a full set of connection weights. While correct in theory, this approach displays two shortcomings in practice: (1) we must solve n optimization problems, where n is the number of postsynaptic neurons, and (2) there are $\mathcal{O}\left(n^2\right)$ weights, which eliminates the space and time efficiency of using factorized weight matrices.

Instead, we solve both issues simultaneously by taking advantage of the linear structure within f^{h_i} that is common between all h_i. Considering Theorem 5.1.3, we must drive the i^{th} synapse with the function:[3]

$$f^{h_i}(\mathbf{x}, \mathbf{u}) = \sum_{j=0}^{k_i} c_{ij} \mathbf{x}^{(j)}.$$

Let $\mathbf{d}^j$ be the set of decoders optimized to approximate $\mathbf{x}^{(j)}$, for all $j = 0 \ldots k$, where $k = \max_i k_i$. By linearity, the optimal decoders used to represent each $f^{h_i}(\mathbf{x}, \mathbf{u})$ may be decomposed as:

$$\mathbf{d}^{f^{h_i}} = \sum_{j=0}^{k_i} c_{ij} \mathbf{d}^j.$$

Next, we express our estimate of each variable $\mathbf{x}^{(j)}$ using the same activity vector $\mathbf{a}$:

$$\mathbf{x}^{(j)} \approx \mathbf{a}^\top \mathbf{d}^j.$$

Now, putting this all together, we obtain:

$$\mathbf{a}^\top \mathbf{d}^{f^{h_i}} = \sum_{j=0}^{k_i} c_{ij} \mathbf{a}^\top \mathbf{d}^j \approx \sum_{j=0}^{k_i} c_{ij} \mathbf{x}^{(j)} = f^{h_i}(\mathbf{x}, \mathbf{u}). \tag{5.40}$$

Therefore, we only need to solve $k+1$ optimization problems, decode the "matrix representation" $\left[\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k)}\right]$, and then linearly combine these $k+1$ different decodings as shown in equation 5.40—using the matrix of coefficients c_{ij}—to determine the input to each synapse. This approach reclaims the advantages of using factorized connection weight matrices, at the expense of a factor $\mathcal{O}(k)$ increase in space and time efficiency. Since Braindrop supports an efficient way of structuring the linear operations between pools (Neckar et al., 2019, Figure 5), we use this technique to account for time-constant mismatch in our simulations in section 7.1.

[3]We do this for Theorem 5.1.3 in particular, but this naturally applies to all other methods as well (both linear, nonlinear, discrete, and continuous).

5.1.5 Harnessing temporal heterogeneity

The methods in the previous section essentially "tame away" the heterogeneity in synaptic time-constants by correcting for the individual scaling factors, and in effect encoding the same state-vector across all synapses. However, we may also consider leveraging this temporal heterogeneity as a basis for dynamical computation. The intuition is as follows: a variety of encoding vectors in the NEF afford heterogeneity across space, while a variety of encoding *filters* afford heterogeneity across time.

Consider a connection between populations where the i^{th} postsynaptic neuron has a distinct filter applied to its input current, referred to as an "encoding filter", $h_i^e(t)$, and the j^{th} presynaptic neuron has a distinct filter applied to its spike-trains, referred to as a "decoding filter", $h_i^d(t)$. By linearity of encoding and decoding, the synapse model for the individual connection from the j^{th} presynaptic neuron to the i^{th} postsynaptic neuron becomes:

$$h_{ij}(t) = (h_i^e * h_j^d)(t). \tag{5.41}$$

If, for example, h_i^e and h_j^d are lowpass filters, then h_{ij} will be the double-exponential synapse. If further they are identical, then this becomes the alpha synapse.

This can be interpreted as a generalization of the NEF, which normally supposes $h_{ij}(t) = h(t)$ for all i and j within the same ensemble. Furthermore equation 5.41 can be interpreted as a *factorization* of the individual synapses, analogous to the factorization of connection weights $\omega_{ij} = \langle \mathbf{e}_i, \mathbf{d}_j \rangle$. In other words, we are supposing that the filters across all of the synapses may be factored (with rank one, but this can be generalized further to each spatial dimension), as is the case for the connection weights themselves. To be clear, the input current to the i^{th} neuron (equation 2.2) is now:

$$\begin{aligned} s_i(t) &= \alpha_i \left(\left[\mathbf{e}_i{}^\top \left(\sum_j (a_j * h_j^d) \mathbf{d}_j \right) \right] * h_i^e \right)(t) + \beta_i & (5.42) \\ &= \alpha_i \sum_j \omega_{ij} (a_j * h_{ij})(t) + \beta_i . & (5.43) \end{aligned}$$

However, equation 5.42 is more space- and time-efficient to implement than equation 5.43 for low-dimensional encoders and decoders, since the inner summation (which is also the decoded vector being represented) does not depend on i.

In order to solve for the decoders $\mathbf{d}$ which approximate this function, it suffices to explicitly simulate the ensemble on some training signal $\mathbf{x}(t)$, determine the corresponding target signal, and then solve for the linear decoders which minimize the root mean-square-error (RMSE) as usual in the NEF. This simulation involves encoding $\mathbf{x}(t)$ through the encoding filters $h_i^e(t)$ and encoding vectors $\mathbf{e}_i$, simulating the neuron models, and finally applying the decoding filters $h_i^d(t)$ to each spike train.

In Friedl et al. (2016), we use this technique to train a neuromorphic touch sensor to classify 18 metal surface textures scanned online. The architecture of the network is depicted in Figure 5.2. Encoding and decoding filters are employed throughout multiple levels of the network to improve performance. First, the input is encoded by nonlinear second-order differential filters, whose distribution of parameters are optimized to closely fit that of mechanoreceptor responses collected from human subjects (Kim et al., 2010). We also use adaptive neurons to improve the efficiency of this code. Second, a hidden layer uses a heterogeneous bank of decaying oscillators (i.e., bandpass filters) to extract a high-dimensional frequency representation from the previous layer. The decoding weights are optimized from the data using a support vector machine (SVM) to solve the least-squares problem as a classification problem instead. We find through cross-validation that the network performs best when it contains a wide variety of filters facilitating a rich feature expansion in time.

5.1.6 Solving for optimal filters

Now we consider the problem of analytically solving for the optimal linear filter, together with the optimal linear decoders, within the *same* least-squares optimization problem. This is useful for situations when we have freedom over the form of the synaptic filter (e.g., Loihi) and would like to find the best synapse(s) to represent some particular target signal (either feed-forward or recurrently). We begin by considering the architecture from Figure 5.3. In the scalar case, $x(t)$ is the known target representation, which as usual we may think of as the PSC without loss of generality, by moving the filtering to occur before the encoding vector, gain, and bias, by linearity. As usual, $a_i(t)$ is the spike-train of the i^{th} neuron encoding $x(t)$. The scalar b is a free parameter scaling the input signal $u(t)$. The decoders d_i are free parameters decoding the activities. Lastly, we'll suppose the synapse has the form:

$$H(s) = \frac{1}{1 + \sum_{i=1}^{k} c_i s^i}, \tag{5.44}$$

where $k \in \mathbb{N}_{\geq 1}$ is fixed and c_i $(1 \leq i \leq k)$ are free parameters for the chosen recurrent synapse. We fix $c_0 = 1$ so that $H(0) = 1 \iff$ its impulse response has an integral of 1. This means that the PSC is normalized (it is free to be scaled by d_i), and finite (i.e., not a perfect integrator). Lemma 5.1.1 may be used to put more general synapses into this form.

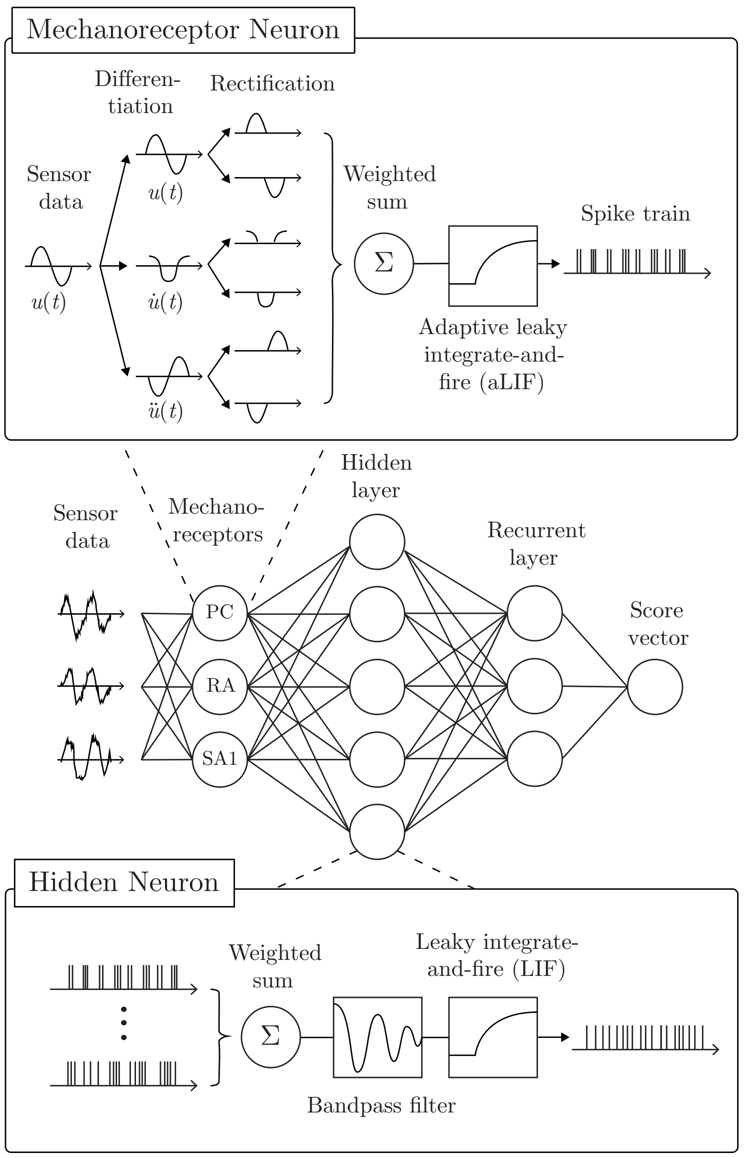


Figure 5.2: Architecture of the surface texture classifier. Three sensors are encoded into the spiking activity of mechanoreceptor neurons. A hidden layer extracts nonlinear features of its spike trains in the frequency domain using a bank of bandpass filters. A recurrently connected output (feedback arrows omitted) represents the classification scores, lowpass filtered over time. Weights are determined using the NEF. Reproduced from Friedl et al. (2016, Figure 1).

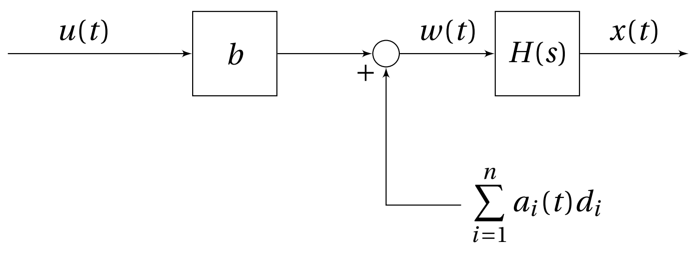


Figure 5.3: Architecture used to formulate the optimization problem that simultaneously solves for the decoding weights and the optimal linear filter.

We define $w(t)$ as the signal driving the synapse. Our goal is to satisfy:

$$
\begin{aligned}
& & \frac{X(s)}{W(s)} &= H(s) \\
\iff & & x(t) + \sum_{i=1}^{k} c_i x^{(i)}(t) &= b \cdot u(t) + \sum_{i=1}^{n} a_i(t) d_i \\
\iff & & b \cdot u(t) + \sum_{i=1}^{k} c_i \left(-x^{(i)}(t) \right) + \sum_{i=1}^{n} a_i(t) d_i &= x(t).
\end{aligned}
$$

We then rewrite this problem in matrix form $AD = Y$, where:

$$
A = \begin{bmatrix} u(t) & -x^{(1)}(t) & \ldots & -x^{(k)}(t) & a_1(t) & \ldots & a_n(t) \end{bmatrix}, \quad D = \begin{bmatrix} b \\ c_1 \\ \vdots \\ c_k \\ d_1 \\ \vdots \\ d_n \end{bmatrix}, \quad Y = \begin{bmatrix} x(t) \end{bmatrix},
$$

and each row of A and Y correspond to an evaluation point t from the dataset. This problem is now in the same form as the least-squares optimization problem of equation 2.9, written in matrix form for a sampled set of evaluation points.

For vector representations, $\mathbf{x}(t)$, we can repeat this independently for each dimension, to solve for a decoding matrix and a vector of synapses. Furthermore, if we substitute this target with the desired postsynaptic current, $\alpha_i \langle \mathbf{x}(t), \mathbf{e}_i \rangle + \beta_i$—analogous to a core idea from the methods of Tripp and Eliasmith (2006); Stöckel et al. (2018)—then we can repeat this optimization procedure for each neuron to obtain a full weight matrix and an individual synaptic filter for each neuron, while maintaining the desired encoding. This can be used to fully leverage hardware where synapses are configurable at the postsynaptic neuron level, in order to improve overall accuracy.

Since our A matrix contains spikes, we should regularize the system by filtering the columns of both A and Y by some filter (e.g., lowpass). Improved accuracy for sparse datasets may be achieved via generalized Tikhonov regularization. For example, by choosing some Bayesian prior over the poles of $H(s)$[4] and mapping this to its corresponding distribution over c_i ($1 \le i \le k$). This results in a regularization matrix with block-structure, with one block imposing the usual diagonal L2-regularization on the entries corresponding to the d_i parameters, and the other imposing some correlation structure on the c_i parameters given the chosen prior on the unknown time-constants of $h(t)$.

This same method also applies to the discrete-time domain to solve for the optimal digital filter (not shown). In general, this provides a way of rolling an unknown set of synapse models, $H(s)$ or $H(z)$, into the NEF's least-squares optimization problem – rather than assuming just a single synapse is fixed *a priori*.

5.2 Neurons

Although the majority of this chapter has focused on extending the NEF to a variety of synapse models, we acknowledge the importance of embracing intrinsic neural dynamics as a useful computational resource as well. On the one hand, Eliasmith and Anderson (2003, appendix F.1) argue that synaptic dynamics dominate that of LIF dynamics at the network level. But on the other hand, there is significant evidence that intracellular dynamics in more sophisticated neuron models play an important functional role in improving signal-to-noise (Eliasmith and Anderson, 2003, chapter 4) and in supporting network-level computations (Izhikevich, 2007). As stated in section 2.2.4, and demonstrated in section 5.1.5, one can explicitly simulate the spike-generation and solve for the decoders over time, thus allowing the optimization to exploit any relevant correlation structures in spike-timing across neurons and any information carried by the individual spike-timing of each neuron. However, such black box approaches do not assist the modeller, *a priori*, in picking the optimal neuron model for some useful function, configuring its parameters, and so forth. Rather, it becomes something of an art; making intuitive judgements about which neurons are doing what, and sweeping over various combinations of models and parameter settings with evolutionary search algorithms or similar. Work that considers the use of teaching signals to train recurrent networks of biophysically-detailed

[4]The time-constants of $h(t)$ are equal to $-p_i^{-1}$ where p_i is the i^{th} pole of $H(s)$.

neurons to implement integration, is a topic of active exploration (Duggins, 2017; Duggins et al., 2017). Making progress within a mathematical framework proves to be challenging, due to the intractability of nonlinear dynamical systems in general. We discuss the work that we have carried out in this direction, and highlight some promising directions for future work.

5.2.1 Adaptive models

In a report by Hunsberger (2016), it was shown that adapting neurons can be successfully decomposed into a linear-nonlinear model. That is, a linear filter followed by a static nonlinearity. Likewise, Lundstrom et al. (2008) have shown that pyramidal neurons can be decomposed in the same way, where the linear filter is a fractional differentiator of the form $H(s) = s^{\alpha}$ for some α. Such *decompositions* are extremely beneficial to the NEF, since they enable the modeller to then conceptually absorb that filter into the higher-order dynamics of the synapse model, and then apply any of our results from section 5.1 with respect to the new filter and the static nonlinearity that remains.

In general, the field of systems identification (Nelles, 2013) is concerned with learning the parameters of dynamical models, represented in various forms, that capture the essential underlying input-output response of some observed dynamical system, such as an adaptive neuron. Indeed, expressions such as the Volterra series, or its model form as a principal dynamic mode (PDM) decomposition (Eikenberry and Marmarelis, 2015), provide ways to characterize and then potentially leverage the principal dynamics of any given model.[5] Our proposed recipe is as follows:

1. Identify the behaviour of the neuron model, $G[\cdot]$, using some other model representation, such as a linear-nonlinear model or a PDM model. This model should capture the output response of the model, with respect to a typical input, and up to some degree of tolerance.
2. Extend the NEF with respect to this new model specification.

For the case of a linear-nonlinear model, this second step is accomplished by section 5.1. For more complicated model representations, such as PDM, we are less certain about what can be done. Essentially, this is about meeting in the middle, between what we know how to describe, and what we know how to exploit. The question then becomes: how complicated may the models in the first step become, before the second step too becomes intractable? We

[5]We remark that any nonlinear dynamical system can in theory be expressed arbitrarily well using the Volterra series – analogous to the Taylor series approximation of any function.

conjecture that this problem—assuming it is well-defined—is generally undecidable. Still, we view this as the penultimate goal of the NEF, and find it fruitful to pursue in specific cases.

Returning to the case of adaptive neurons, we can still say more. Consider the model used by Nengo (Camera et al., 2004). This model describes an additional state-variable, $k_i(t)$, for each neuron, that is equal to a lowpass filtered version of its own spike-train with some time-constant, τ_{adapt}. It then subtracts a scaled version, $\alpha_{\text{adapt}} k_i(t)$, from its own input current, which results in the effect where: the more it spikes, the more it inhibits itself in the future. This description suggests a natural decomposition of the adaptive LIF: express it as a LIF neuron, that is recurrently coupled to itself with weight $\omega_{i,i} = -\alpha_{\text{adapt}}$ and a synaptic filter described by a lowpass with a time-constant of τ_{adapt}, also known as an *autapse*. This provides an equivalent model description that the NEF can fully leverage. Specifically: include a recurrent diagonal weight matrix, where the i^{th} diagonal is α_{adapt}, and use a synapse model with time-constant τ_{adapt}. This then cancels out the adaptive term, and reduces the architecture back to the case of a LIF neuron. Using this technique, we have verified that Principle 3 can be implemented for the case of an adaptive LIF population with equivalent accuracy to that of its dual LIF population (not shown). Although this may seem like a pointless exercise, as we have merely gone around in a circle and back to where we started, this reveals an important equivalence relation, namely: *the adaptive LIF is equivalent to a LIF model with a diagonal inhibitory feedback loop*. A crucial observation here is that a diagonal matrix cannot be factored any further (it is full rank). In the context of the NEF, this implies that the network-effect of adaptive LIF models is to expand the dimensionality of the dynamical state-space from q to n, where the additional dimensions emerge from a sparse $(1/n)$-encoding with itself. This provides an explanation for the computational power of adaptive LIF neurons in the context of Principle 3, but only gets us so far due to the difficulty in analyzing an n-dimensional nonlinear dynamical system for large values of n. We have found, however, that this diagonal matrix can in many cases be approximated by the product of two low-rank matrices—analogous to encoders and decoders—whose state-space essentially represents a projection of the population's adaptive state, while maintaining essential properties of the network-level dynamics. This highlights a promising avenue for future research, namely the characterization of this additional state-vector and its subsequent harnessing for particular classes of dynamical systems.

5.2.2 Poisson models

Recalling our analysis from section 3.2.1, the NEF must cope with three sources of error in the neural representation: (1) "static distortion" arising from the use of decoders, (2) "spike noise" arising from the use of discontinuous spikes, and (3) "state discrepancy" arising from the substitution of a stateless model with a stateful model. The first two issues are resolved by scaling up the neuron count or total number of spikes, but the third is not, unfortunately. For the latter issue, the intuitive goal is for our neurons to be "uniformly ready to spike" at every given moment in time. We formalized this with our definitions of ISIP and state discrepancy (definition 3.2.1), and then proceeded to prove that the error converges to zero if the ISIPs are uniformly distributed (Theorem 3.2.1). The issue with LIF neurons is that, for non-constant inputs, this requirement is systematically violated (see Figure 3.4). Each neuron is biased away from the uniform distribution, tending towards the start of its ideal ISI (see Figure 3.3(d)), due primarily to leakage. This observation goes well beyond the scope of the LIF model, to any neuron model that has some internal memory or state that might lead it to violate the criteria of uniformity.

This discussion motivates the search for a "memoryless" spiking neuron model – one that cannot be biased towards a non-uniform probability of being ready to spike by its prior inputs or absence thereof. The exponential distribution is the only such continuous-time probability distribution. This distribution describes the arrival time between Poisson events. Therefore, a Poisson-spiking model is the only *probabilistic* type of neuron model that satisfies our criteria for eliminating this source of error. We thus consider Poisson-spiking neurons where the number of spikes within a given time-step follows a Poisson distribution with a rate given by some desired static response curve (here, equation 2.8) evaluated at the input current.

Poisson-spiking models have numerous benefits:

1. Cheap to simulate in both space and time, as they require no additional memory or updates to state.

2. May be applied to any static response curve without loss of generality.

3. There is no latency when processing inputs that jump discontinuously, and more generally when processing high-frequency input signals, as there is no start-up transient or requirement to initialize the model; the expected spike rate of each neuron always equals the desired instantaneous rate.

4. By solving the state discrepancy problem, the performance continues to scale with the number of neurons.

5. Scaling holds even with arbitrarily low firing rates, thus promoting a reduction in total power by way of fewer spikes needed to achieve high levels of precision in certain cases.

Integrate-and-fire (i.e., non-leaky LIF, or "ReLU") spiking neurons also share benefits 3, 4, and 5 – assuming its voltages are initialized to $v_i(0) \sim \mathcal{U}[0,1]$ at the beginning of the simulation.

Benefits 1 and 2 are lost, however, as the ReLU is a rectified linear (and unbounded) curve which may not provide the most efficient basis for some static nonlinearities, and it requires additional memory to maintain the state-variable (voltage) over time.

We validate these claims in Figures 5.4 and 5.5 by using each type of LIF model—Poisson-spiking, non-leaky, adaptive, and regular-spiking—evaluating the root-mean-squared error (RMSE) against the filtered ideal. In all cases the maximum firing rates of the neurons are limited to 20–40 Hz, and the states of the neurons are initialized to be mixed (i.e., uniform ISIPs). In both figures we display 95% confidence intervals bootstrapped from 10 trials in each condition.

In Figure 5.4, we demonstrate that precision remains constant for both Poisson-spiking and non-leaky neurons, even at frequencies greater than six times that of the maximum firing rates of any given neuron. For this, we set $n = 50f$, $\tau = 10f^{-1}$ ms, and $\mathtt{dt} = 1f^{-1}$ ms, as we scale the frequency of the sinusoid, f Hz. This choice of scaling is informed by equation 3.4, which tells us that we should balance $\tau = f^{-1}$ to keep $\mathcal{O}\left(2\pi\tau f\right)$ fixed, while increasing n in proportion to f to scale the total number of spikes with frequency. Due to the variability in a Poisson process, it is sub-optimal for constant inputs. However, performance remains constant at higher frequencies.

In Figure 5.5, we demonstrate that the precision continues to scale as $\mathcal{O}\left(\tau\sqrt{n}\right)$ for both Poisson-spiking and non-leaky neurons, but not for the other two. For this, we fix $f = 10$ Hz, $\tau = 1$ ms, and $\mathtt{dt} = 0.1$ ms, as we scale the neuron count. This demonstrates that Poisson spiking models (and ReLU) solve the state discrepancy problem, and thus continue to scale in accuracy with neuron count at increasingly-greater signal frequencies relative to the individual firing rates. The properties of the Poisson neuron model make it particularly appealing for neuromorphic architectures such as SpiNNaker 2. Specifically, SpiNNaker 2 has specialized hardware for computing exponentials (Partzsch et al., 2017) and uniform samples (Liu et al., 2018), and the architecture benefits enormously from the reduction of memory requirements and spike-traffic (Stromatias et al., 2013; Mundy et al., 2015). Since the Poisson spike-generator can be applied to any desired static response curve, this method can be applied to whatever is cheap to compute while still providing a suitable nonlinear basis for the desired transformation.

Despite the poor performance of the adaptive LIF model in these experiments, this could be an artifact of the particular adaptive mechanism (*subtracting* from its input current) combined with the statistics of the input signal (pure sinusoid). Adaptive neurons may still play an important role in efficient neural coding and the NEF, in particular by the complementary means of: (1) expanding the dimensionality of the dynamics, (2) providing higher-order encoding filters, (3) adjusting the static nonlinearities over time, and (4) maintaining the invariance of the ISIP criteria with respect to the statistics of the input signal using fewer spikes.

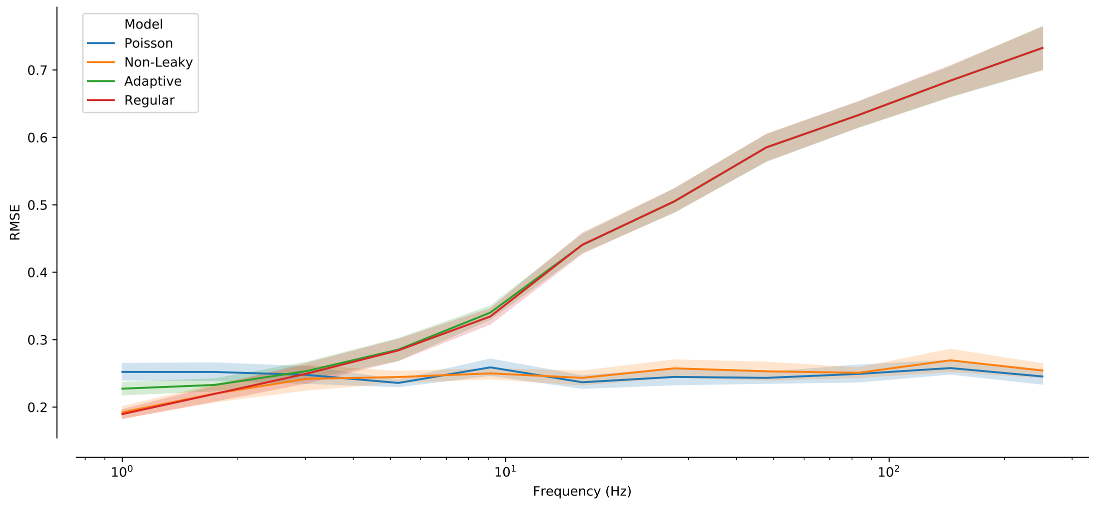


Figure 5.4: Scaling of LIF performance with frequency, for each type: Poisson-spiking, non-leaky (i.e., integrate-and-fire), adaptive, and regular-spiking. The input is a sinusoid with frequency f Hz. Due to the memoryless property of the Poisson process, and the uniformity of non-leaky LIF, both outperform regular LIF with constant precision as $f \to \infty$ (Theorem 3.2.1), assuming n and τ are scaled appropriately with frequency (see text for details).

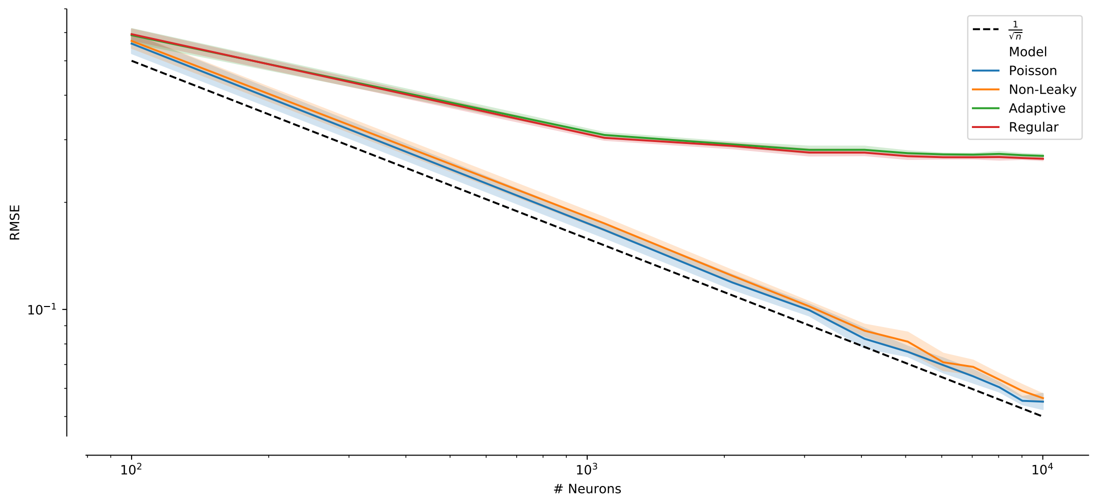


Figure 5.5: Scaling of LIF performance with neuron count ($f = 10$ Hz), for each type: Poisson-spiking, non-leaky (i.e., integrate-and-fire), adaptive, and regular-spiking. Regular-spiking LIF plateaus in performance for large numbers of neurons, due to the state discrepancy (definition 3.2.1). The precision of integrate-and-fire and Poisson-spiking both scale as $\sqrt{n}$ as $n \to \infty$ (Theorem 3.2.1), since each neuron is always ready to spike with the ideal probability.

Chapter 6

Delay Networks

This chapter has been adapted from, while significantly extending, the work of Voelker (2015a); Voelker and Eliasmith (2016, 2018, 2019, patent pending).

A particularly important dynamical system that has not been discussed before in the NEF literature is the pure *continuous-time delay* line. In order to implement such a system, one must represent a *rolling window* of input history, or, in other words, one must *buffer* the input signal into a memory that continuously slides alongside the current input. In this chapter, we provide a novel derivation of an optimal low-dimensional linear approximation to a continuous-time delay, and then realize this *Delay Network* (DN) using the NEF in a recurrent spiking network. We then investigate its computational properties, proving that the resulting network implements a well-defined nonlinear encoding of its input across the delay interval – isomorphic to a high-dimensional projection of the shifted Legendre polynomials. This network uses a scale-invariant representation, with a level of accuracy that depends on the input frequency, chosen dimensionality (i.e., the order of the approximation), and particular synapse model. To our knowledge, this work is the first to demonstrate that such a temporal code may be accurately implemented using a spiking dynamical network.

Reservoir Computing approaches, such as Liquid State Machines (Maass et al., 2002) and Echo State Networks (Jaeger, 2001), may be used to approximate a delay line. However, since these networks use randomly chosen feedback weights, we show in section 6.2.1 that they do so with relatively poor accuracy despite extensive hyper-optimization. Such networks instead represent a random variety of nonlinear memory traces (Lukoševičius, 2012b). Discrete approaches to short-term memory, such as those taken by White et al. (2004) and Ganguli et al. (2008), while optimal in an information-theoretic sense, rely fundamentally on single time-step delays between rate-based neurons. In contrast, the method that we propose here works independently of the simulation time-step, and is optimal assuming the population of spiking neurons—coupled with some model of the synapse—accurately represents a low-dimensional, low-frequency, vector space. Furthermore, we use our extensions from section 5.1, which improves our understanding of the relationship between synapse models and network-level computations.

We also find that the delay-line is a difficult function for FORCE networks to learn (section 6.2.2); re-encoding the delayed output as a teaching signal ends up "confusing" the network. Furthermore, we find that our network, when expressed as an RNN cell (without spikes), outperforms equivalently-sized stacked LSTMs on computing long delays, and predicting the Mackey-Glass dataset—a difficult chaotic time-series benchmark—in training time, inference time, and test accuracy. We then reveal some connections between the neural responses within our delay network, and "time cells" in the neuroscience literature. Lastly, we discuss a number of theoretical applications that are directly supported by, and extend, the computations of the delay network.

To begin, consider a continuous-time delay line of θ seconds:

$$y(t) = (u * \delta_\theta)(t) = u(t-\theta), \quad \theta > 0, \tag{6.1}$$

where δ_θ denotes the Dirac delta shifted forwards in time by θ. This system takes a time-varying scalar signal, $u(t)$, and outputs a purely delayed version, $u(t-\theta)$. The task of computing this function both accurately and efficiently in a biologically-plausible, spiking, dynamical network, is a significant theoretical challenge, that, to our knowledge, has previously remained unsolved.

The continuous-time delay is worthy of detailed consideration for several reasons. First, it is non-trivial to implement using continuous-time spiking dynamical primitives. Specifically, equation 6.1 requires that we maintain a *rolling window* of length θ (i.e., the history of $u(t)$, going θ seconds back in time). Thus computing a delay of θ seconds is just as hard as computing every delay of length θ', for all $0 \le \theta' \le \theta$. Since any finite interval of $\mathbb{R}$ contains an uncountably-infinite number of points, an exact solution for arbitrary $u(t)$ requires that we maintain an uncountably-infinite amount of information in memory. Second, the delay provides us with a window of input history from which to compute arbitrary nonlinear functions across time. For instance, the spectrogram of a signal may be computed by a nonlinear combination of delays, as may any finite impulse response (FIR) filter. Third, delays introduce a rich set of interesting dynamics into large-scale neural models, including: oscillatory bumps, traveling waves, lurching waves, standing waves, aperiodic regimes, and regimes of multistability (Roxin et al., 2005). Fourth, a delay line can be coupled with a single nonlinearity to construct a network displaying many of the same benefits as Reservoir Computing (Appeltant et al., 2011; Bai and Yi, 2018).

Many learning algorithms also tend to rely on sliding windows of input history. For instance, Leng et al. (2013) and Izzeldin et al. (2011) improved their neural networks' accuracy by incorporating sliding windows into their learning rules – but they did not offer any neural implementation for the component that performs the required buffering. Similarly, Ferreira and Ruano (2009) found sliding windows to be beneficial for the online learning of process models, in which the parameters of a nonlinear time-varying system are identified.

The computational power of delays is explained more abstractly by Takens' embedding theorem (Takens, 1981; Noakes, 1991).[1] This theorem provides a very deep connection between delays and dynamical systems, specifically: any chaotic attractor, with finite box counting dimension, k, can be reconstructed by a static nonlinear projection of $2k$ different delays of just *one* of its internal state variables. In other words, a basis of delays, when applied to just a single observed variable, provides enough information to infer its entire unobserved state. These insights have practical applications to time-series forecasting in the natural sciences (Sugihara et al., 2012). And, in particular, this provides theoretical grounding for why delays might be computationally useful in biological systems; their dynamics support the inference of high-dimensional states from low-dimensional observations over time.

Our delay network is also somewhat analogous to an autoencoder (Gulli and Pal, 2017) from machine learning. However, rather than compressing the input into a low-dimensional space that allows for it to be reconstructed spatially, we do this *temporally*. This leads to a number of applications that can, in effect, mentally travel back in time using just a handful of latent state-variables embedded within a spiking network. The resulting representations are fully interoperable with the semantic pointer architecture (SPA; Eliasmith et al., 2012), and thus afford the same manipulations as semantic pointers in Spaun, namely: binding into working memory, compressing information from cortex, internally routing information, and driving motor actions (Eliasmith, 2013).

At a higher level, any physical realization of a delay must necessarily be modelling the statistics of its input in some way. This is because it is physically impossible to buffer arbitrary signals in continuous time (details in section 6.1.1), and thus some sacrifice must be made. In essence, this sacrifice amounts to a sparse coding mechanism, in which a low-dimensional representation is constructed that maps onto the infinite-dimensional time window (Blumensath and Davies, 2009). Spikes in a neural network are akin to such a compression mechanism in time, as are decoders in space. Our solution leverages both methods of sparsification, but not in a way that one might come to expect from information theory or other approaches involving discrete samples; our NEF-based approach finds the optimal system for linearly mapping continuous-time windows onto a low-dimensional manifold.

[1] We thank Peter Suma for pointing out this connection.

6.1 Derivations

It is impossible in practice (i.e., given finite-order continuous-time resources) to implement an arbitrary delay. For instance, a white noise signal contains an uncountably-infinite amount of information within any finite window, that cannot be compressed any further (Cover and Thomas, 2012). We instead approximate $u(t)$ as a low-frequency signal, or, equivalently, approximate equation 6.1 as a low-dimensional system expanded about the zeroth frequency in the *Laplace domain*. Our choice of a zero-frequency approximation is informed by our analysis from section 3.1.6, which suggests that neural systems require energy that grows linearly in the representational frequency.

6.1.1 Optimal approximation

We begin by transforming equation 6.1 into the Laplace domain, $\mathcal{L}\left\{y(t)\right\} = \mathcal{L}\left\{u(t)\right\}\mathcal{L}\left\{\delta_\theta(t)\right\}$, and then using the fact that $\mathcal{L}\left\{\delta_\theta(t)\right\} = e^{-\theta s}$ to obtain:

$$F(s) := \frac{\mathcal{L}\left\{y(t)\right\}}{\mathcal{L}\left\{u(t)\right\}} = e^{-\theta s}, \tag{6.2}$$

where $F(s)$ is known as the *transfer function* of the system, defined as the ratio of the Laplace transform of the output to the Laplace transform of the input. Equation 6.2 should be understood as an equivalent way of expressing equation 6.1 in the Laplace domain, where the variable s denotes a complex frequency. Notably, we have not made any sacrifices at this point, as the ideal system is a linear dynamical system. Thus far, we have only described the transfer function that we would like the network to implement.

The state-space model discussed in section 2.2.3 (equation 2.12) is related to its transfer function by equation 5.2. Conversely, a transfer function can be converted into a state-space model satisfying equation 5.2 *if and only if* it can be written as a proper ratio of finite polynomials in s (Brogan, 1991). The ratio is proper when the degree of the numerator does not exceed that of the denominator. In such a case, the output will not depend on future input, and so the system is *causal*. The degree of the denominator corresponds to the dimensionality of the state-vector, and therefore must be finite. These two conditions align with physically realistic constraints where time may only progress forward, and neural resources are limited.

However, the pure delay (equation 6.2) has infinite order when expressed as a ratio of polynomials in s, and so the system is irrational, or infinite-dimensional. Consequently, no finite state-space model will exist for $F(s)$, which formalizes our previous remark that an exact solution is impossible for finite, continuous-time systems. To overcome this, we must approximate the irrational transfer function $e^{-\theta s}$ as a proper ratio of finite-order polynomials. We do so using its *Padé approximants*—the coefficients of a Taylor series extended to the ratio of two polynomials—expanded about $s = 0$ (Padé, 1892; Vajta, 2000):

$$\begin{aligned} [p/q]\, e^{-\theta s} &= \frac{\mathcal{B}_p(-\theta s)}{\mathcal{B}_q(\theta s)}, \\ \mathcal{B}_m(s) &:= \sum_{i=0}^{m} \binom{m}{i} \frac{(p+q-i)!}{(p+q)!} s^i. \end{aligned} \tag{6.3}$$

This provides the transfer function of order p in the numerator and order q in the denominator that optimally approximates equation 6.2 for low-frequency inputs (i.e., up to order $p + q$).

After choosing $0 \le p \le q$, we may numerically find a state-space model (A, B, C, D) that satisfies equation 5.2 using standard methods,[2] and then map this system onto the synapse using Principle 3. However, naïvely applying this conversion leads to numerical issues in the representation (i.e., dimensions that grow exponentially in magnitude), due in part to the factorials in equation 6.3.

To overcome this problem, we derive an equivalent yet normalized state-space model that we have not encountered elsewhere. We do so for the case of $p = q - 1$, since this provides the best approximation to the step-response (Vajta, 2000). We symbolically transform equation 6.3 into a normalized state-space model that avoids the need to compute any factorials. We begin by expanding equation 6.3:

$$\begin{aligned} [q-1/q]e^{-\theta s} &= \frac{\sum_{i=0}^{q-1} \binom{q-1}{i} (2q-1-i)!(-1)^i \theta^i s^i}{\sum_{i=0}^{q} \binom{q}{i} (2q-1-i)! \theta^i s^i} \\ &= \frac{\frac{1}{\theta^q (q-1)!} \sum_{i=0}^{q-1} \frac{(q-1)!}{(q-1-i)!i!} (2q-1-i)! \theta^i s^i (-1)^i}{s^q + \frac{1}{\theta^q (q-1)!} \sum_{i=0}^{q-1} \frac{q!}{(q-i)!i!} (2q-1-i)! \theta^i s^i} \\ &= \frac{\sum_{i=0}^{q-1} c_i s^i}{s^q + \sum_{i=0}^{q-1} d_i s^i}, \end{aligned}$$

where $d_i := \frac{q(2q-1-i)!}{(q-i)!i!} \theta^{i-q}$ and $c_i := (-1)^i \binom{q-i}{q} d_i$.

[2] For instance, the function `tf2ss` in MATLAB or SciPy.

This transfer function is readily converted into a state-space model in controllable canonical form:

$$A = \begin{pmatrix} -d_{q-1} & -d_{q-2} & \cdots & -d_0 \\ 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ 0 & 0 & 1 & 0 \end{pmatrix}, \qquad \begin{aligned} B &= \begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix}^\mathsf{T}, \\ C &= \begin{pmatrix} c_{q-1} & c_{q-2} & \cdots & c_0 \end{pmatrix}, \\ D &= 0. \end{aligned}$$

To eliminate the factorials in d_i and c_i, we scale the i^{th} dimension of the state-vector by d_{q-1-i}, for all $i = 0 \dots q-1$, which is equivalent to transforming the system using a diagonal state-space transformation. This is achieved without altering the transfer function by scaling each $(B)_j$ by d_{q-1-j}, each $(C)_i$ by $1/d_{q-1-i}$, and each $(A)_{ij}$ by d_{q-1-i}/d_{q-1-j}, which yields the equivalent state-space model:

$$A = \begin{pmatrix} -v_0 & -v_0 & \cdots & -v_0 \\ v_1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ 0 & 0 & v_{q-1} & 0 \end{pmatrix}, \qquad \begin{aligned} B &= \begin{pmatrix} v_0 & 0 & \cdots & 0 \end{pmatrix}^\mathsf{T}, \\ C &= \begin{pmatrix} w_0 & w_1 & \cdots & w_{q-1} \end{pmatrix}, \\ D &= 0, \end{aligned} \tag{6.4}$$

where $v_i := \dfrac{(q+i)(q-i)}{i+1}\theta^{-1}$ and $w_i := (-1)^{q-1-i}\left(\dfrac{i+1}{q}\right)$, for $i = 0 \dots q-1$. This follows from noting that $v_0 = d_{q-1}$ and $v_i := d_{q-1-i}/d_{q-i}$ for $i \geq 1$.

A similar derivation applies to the case where $p = q$, although it results in a passthrough $(D \neq 0)$ which is suboptimal for step-responses. For brevity, we omit this derivation, and instead simply state the result:

$$A = \begin{pmatrix} -v_0 & -v_0 & \cdots & -v_0 \\ v_1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ 0 & 0 & v_{q-1} & 0 \end{pmatrix}, \qquad \begin{aligned} B &= \begin{pmatrix} -v_0 & 0 & \cdots & 0 \end{pmatrix}^\mathsf{T}, \\ C &= \begin{pmatrix} 2(-1)^q & 0 & 2(-1)^q & 0 & \cdots & \cdots \end{pmatrix}, \\ D &= (-1)^q, \end{aligned}$$

where $v_i = \dfrac{(q+i+1)(q-i)}{i+1}\theta^{-1}$, for $i = 0 \dots q-1$.

In either case, A and B depend on the delay length solely by the scalar factor θ^{-1}. For this reason, and to keep quantities dimensionless, we often make the substitution $\mathbf{x}(t) \leftarrow \theta\mathbf{x}(t)$ to express the system in an alternative form (Neckar et al., 2019):

$$\theta\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + Bu(t), \quad A = \begin{pmatrix} -v_0 & -v_0 & \cdots & -v_0 \\ v_1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ 0 & 0 & v_{q-1} & 0 \end{pmatrix}, \quad B = \begin{pmatrix} v_0 & 0 & \cdots & 0 \end{pmatrix}^\mathsf{T}, \tag{6.5}$$

where $v_i := (q+i)(q-i)(i+1)^{-1}$ no longer depends on θ. As a side-effect, we may *control* the length of the delay by adjusting the gain on the integration time-constant (i.e., by scaling the input and feedback signals). The NEF can be used to build such controlled dynamical systems, without introducing multiplicative dendritic interactions or implausible on-the-fly connection weight scaling (Eliasmith and Anderson, 2000).

This model is now equivalent to equation 6.3, but without any factorials, and in the form of equation 2.12.[3] The choice of q corresponds to the dimensionality of the latent state-vector $\mathbf{x}(t)$ that is to be represented by Principle 1 and transformed by Principle 2. Principle 3 may then be used to map equation 6.4 onto a spiking dynamical network to accurately implement an optimal low-frequency approximation of the delay.

To demonstrate, we implement a 1 s delay of 1 Hz band-limited white noise using 1,000 recurrently connected spiking LIF neurons representing a 6-dimensional vector space (see Figure 6.1). The connections between neurons are determined by applying Principle 3 (section 2.2.3) to the state-space model derived above (equation 6.4, $q = 6$) via the Padé approximants of the delay. The normalized root-mean-squared error (NRMSE; normalized so that 1.0 would correspond to random chance) of the output signal is 4.8%. This is achieved without appealing to the simulation time-step (`dt` = 1 ms); in fact, as shown in section 6.2.7, the network accuracy improves as `dt` approaches zero due to the continuous-time assumption mentioned in section 2.2.3 (and relaxed in section 5.1.1).

6.1.2 Temporal representation

Although the delay network has its dynamics optimized for a single delay $\theta > 0$, we may still accurately decode any delay $0 \le \theta' \le \theta$ from the state of the same network, as intuitively it needs to be holding onto this memory. In other words, the network is representing a rolling window (i.e., history) of length θ. To compute these other delays, we must approximate $e^{-\theta' s}$ with a transfer function

$$F_{\theta \to \theta'}(s) := \frac{\mathcal{C}(s; \theta, \theta')}{\mathcal{D}(s; \theta)}$$

of order $[p/q]$, such that the denominator $\mathcal{D}(s;\theta)$ (which provides us with the recurrent transformation up to a change of basis) depends only on θ, while the numerator $\mathcal{C}(s;\theta,\theta')$ (which provides us with the output transformation up to a change of basis) depends on some relationship between θ' and θ.

[3] In section 4.1, we mention NengoLib's features for several other state-space realizations.

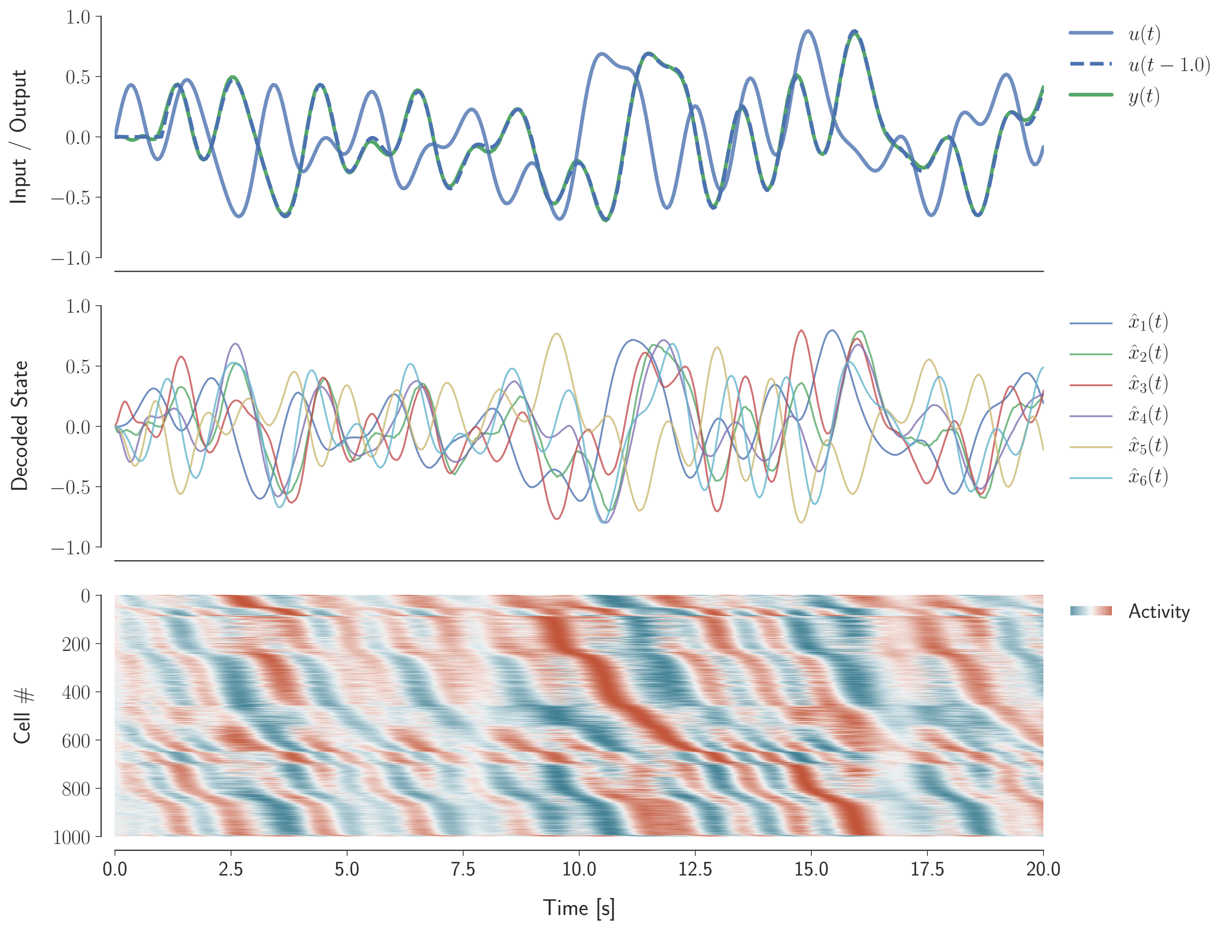


Figure 6.1: Delay of 1 s implemented by applying standard Principle 3 to equation 6.4 using $q = 6$, $\mathtt{dt} = 1$ ms, 1,000 spiking LIF neurons, and a lowpass synapse with $\tau = 0.1$ s. The input signal is white noise with a cutoff frequency of 1 Hz. The plotted spikes are filtered with the same $\tau = 0.1$ s, and encoded with respect to 1,000 encoders sampled uniformly from the surface of the hypersphere (sorted by time to peak activation). Reproduced from Voelker and Eliasmith (2018, Figure 3).

From equation 6.3, we may write the denominator as:

$$\mathcal{D}(s;\theta) = \sum_{i=0}^{q} d_i(\theta) s^i, \quad d_i(\theta) := \binom{q}{i} \frac{(p+q-i)!}{(p+q)!} \theta^i.$$

We then solve for the numerator, as follows:

$$\begin{aligned} [p/q] e^{-\theta' s} &= \sum_{i=0}^{\infty} \frac{(-\theta' s)^i}{i!} = \frac{\mathcal{C}(s;\theta,\theta')}{\mathcal{D}(s;\theta)} \\ \iff \quad \mathcal{C}(s;\theta,\theta') &= \left(\sum_{i=0}^{\infty} \frac{(-\theta' s)^i}{i!} \right) \left(\sum_{j=0}^{q} d_j(\theta) s^j \right) + \mathcal{O}\left(s^{p+1}\right). \end{aligned}$$

By expanding this product and collecting like terms, the correct numerator up to order $p \leq q$ is:

$$\mathcal{C}(s;\theta,\theta') = \sum_{i=0}^{p} c_i(\theta,\theta') s^i, \quad c_i(\theta,\theta') := \sum_{j=0}^{i} \frac{(-\theta')^{i-j}}{(i-j)!} d_j(\theta).$$

Therefore, the optimal readout for a delay of length θ', given the dynamics for a delay of length θ, is determined by the above linear transformation of the coefficients $\left(d_j(\theta)\right)_{j=0}^{p}$.

We remark that $c_i(\theta,\theta) = \binom{p}{i} \frac{(p+q-i)!}{(p+q)!} (-\theta)^i$, since $F_{\theta \to \theta}(s) = [p/q] e^{-\theta s}$, by uniqueness of the Padé approximants, and by equation 6.3. As a corollary, we have proven that the following combinatorial identity holds for all $p, q \in \mathbb{N}$ and $i \in \left[0, \min\{p,q\}\right]$:

$$\binom{p}{i} = \sum_{j=0}^{i} (-1)^j \binom{q}{j} \binom{p+q-j}{i-j}.$$

For the case when $p = q-1$, we may also apply the same state-space transformation used to derive equation 6.4 to obtain the normalized coefficients for the C transformation (i.e., with A, B, and D unchanged):

$$\begin{aligned} w_{q-1-i} &= \left(\sum_{j=0}^{i} \frac{(-\theta')^{i-j}}{(i-j)!} \binom{q}{j} \frac{(2q-1-j)!}{(2q-1)!} \theta^j \right) \left(\frac{(q-i)!i!(2q-1)!}{\theta^q (q-1)! q (2q-1-i)!} \theta^{q-i} \right) \\ &= \sum_{j=0}^{i} \binom{q}{j} \left(\frac{(2q-1-j)!}{(i-j)!(2q-1-i)!} \right) \left(\frac{(q-i)!i!}{q!} \right) \left(\theta^{j-i} \right) (-\theta')^{i-j} \\ &= \binom{q}{i}^{-1} \sum_{j=0}^{i} \binom{q}{j} \binom{2q-1-j}{i-j} \left(\frac{-\theta'}{\theta} \right)^{i-j}, \quad i = 0 \ldots q-1. \end{aligned}$$

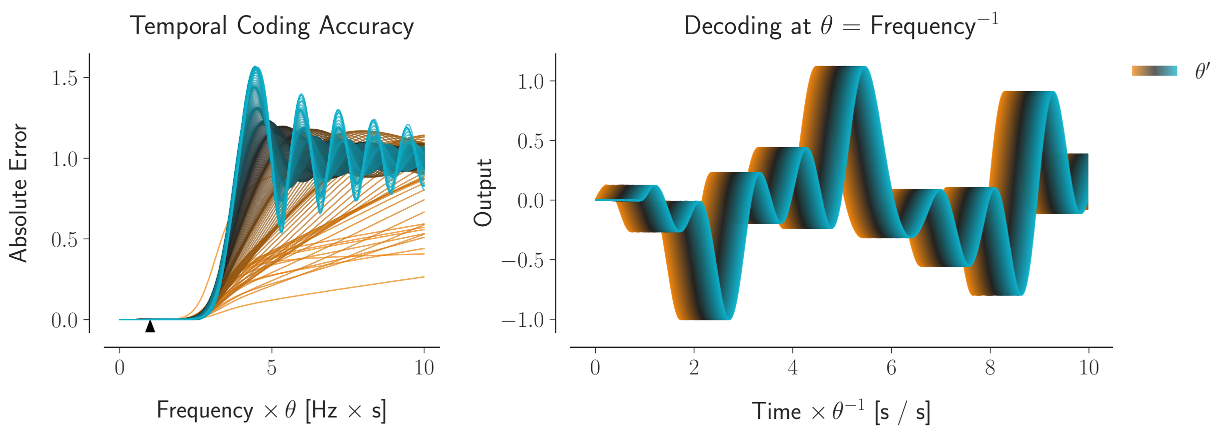


Figure 6.2: Decoding a rolling window of length θ. Each line corresponds to a different delay, ranging from 0 to θ, decoded from a single Delay Network ($q = 12$). (Left) Error of each delay, as the input frequency, f, is increased relative to θ. Shorter delays are decoded more accurately than longer delays at higher frequencies. A triangle marks $\theta = f^{-1}$. (Right) Example simulation decoding a rolling window of white noise with a cutoff frequency of θ^{-1} Hz. Reproduced from Voelker and Eliasmith (2018, Figure 4).

From this, we see that different decodings require different linear output transformations (C) for each θ', with the following coefficients:

$$w_{q-1-i} = \binom{q}{i}^{-1} \sum_{j=0}^{i} \binom{q}{j} \binom{2q-1-j}{i-j} \left(\frac{-\theta'}{\theta}\right)^{i-j}, \quad i = 0 \ldots q-1. \tag{6.6}$$

The underlying dynamical state remains the same. To be clear, the q-dimensional state-vector of the delay network represents a rolling window of length θ. That is, a single delay network with some fixed $\theta > 0$ may be used to accurately decode any delay of length θ' ($0 \leq \theta' \leq \theta$) by taking a linear transformation of its state-vector according to the coefficients of equation 6.6.

In Figure 6.2, we take different linear transformations of the same state-vector, by evaluating equation 6.6 at various delays between 0 and θ, to decode the rolling window of input from the state of the system.[4] This demonstrates that the delay network compresses the input's history (lasting θ seconds) into a low-dimensional state. In Figure 6.3, we sweep equation 6.6 across $\theta'\theta^{-1}$ to visualize the temporal "basis functions" of the delay network. This provides a means of understanding the relationship between the chosen state-space representation (i.e., the

[4]The optimization problem from equation 2.9 need only be solved once to decode $\mathbf{x}(t)$ from the neural activity. The same decoders may then be transformed by each C without loss in optimality (by linearity).

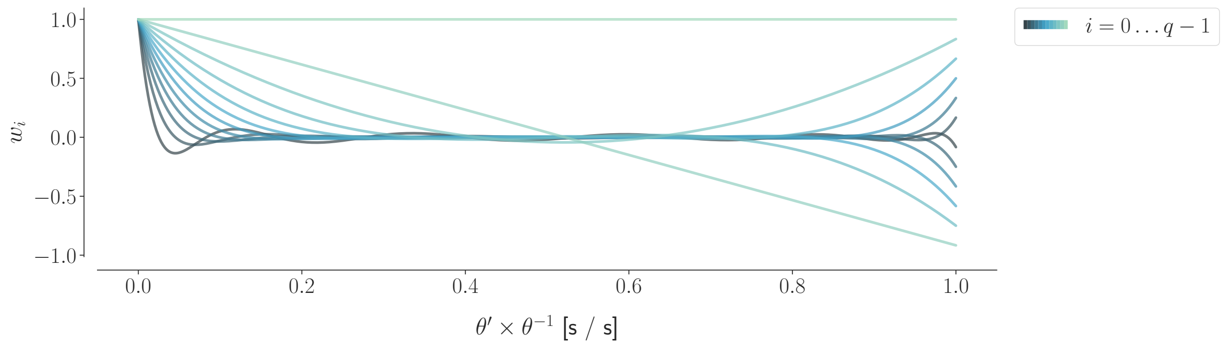


Figure 6.3: Temporal basis functions for the Delay Network ($q = 12$). Each line corresponds to the basis function of a single dimension (i) ranging from 0 (darkest) to $q-1$ (lightest). The i^{th} basis function is a polynomial over $\theta'\theta^{-1}$ with degree i (see equation 6.6). The state-vector of the Delay Network takes a linear combination of these q basis functions in order to represent a rolling window of length θ. Reproduced from Voelker and Eliasmith (2018, Figure 5).

q-dimensional $\mathbf{x}(t)$) and the underlying window representation (i.e., the infinite-dimensional $u(t)$). In particular, each basis function corresponds to the continuous window of history represented by a single dimension of the delay network. The instantaneous value of each dimension acts as a coefficient on its basis function, to contribute to the representation of the window at that point in time. Overall, the entire state-vector determines a linear combination of these q basis functions to represent the window. An intriguing consequence of this and equation 6.6, is that the temporal code employed by this network—in terms of the basis functions that it takes to reconstruct the window—is a linear transformation of the *shifted Legendre polynomials* up to the same order, as detailed in section 6.1.3. Thus the network projects its input onto the Legendre basis over time.

The encoder of each neuron can also be understood directly in these terms as taking a linear combination of the basis functions (via equation 2.2). Each neuron nonlinearly encodes a projection of the rolling window onto some "preferred window" determined by its own encoder. Since the state-vector is encoded by heterogeneous neural nonlinearities, the population's spiking activity supports the decoding of nonlinear functions across the entire window (i.e., functions that we can compute using Principles 1 and 2). Therefore, we may conceptualize the delay network as a *temporal coding* of the input stimulus, which constructs a low-dimensional state—representing an entire window of history—to encode the temporal structure of the stimulus into a nonlinear high-dimensional space of neural activities. We discuss this in more detail in the following sections.

To more thoroughly characterize the delay dynamics, we analyze the behaviour of the delay

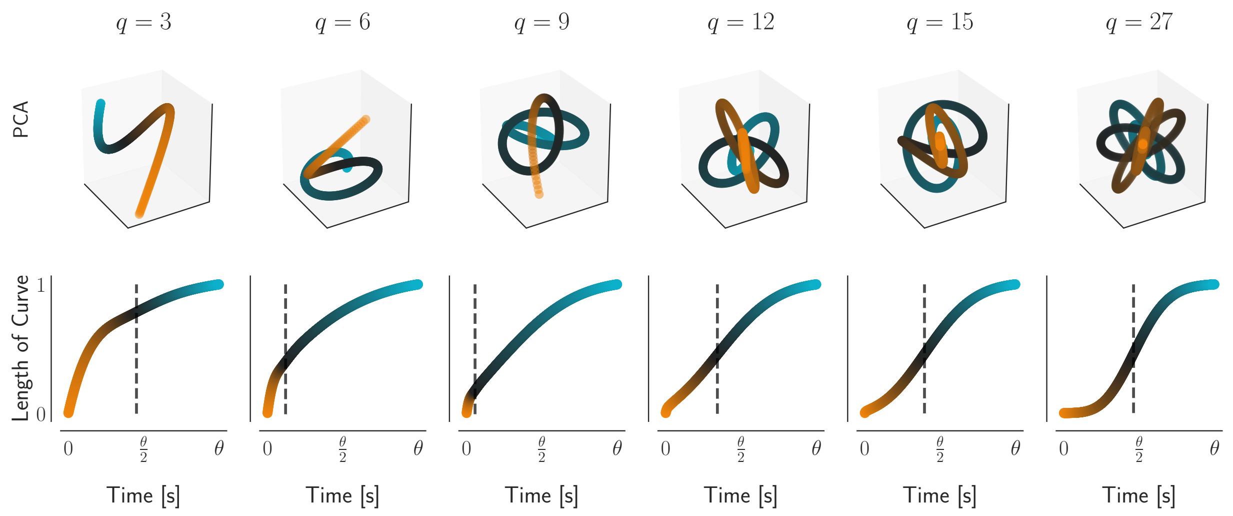


Figure 6.4: Impulse response of the Delay Network with various orders (q) of Padé approximants. (Top) The state-vector $\mathbf{x}(t)$ projected onto its first three principal components. (Bottom) The length of the curve $\mathbf{x}$ up to time t, computed using the integral $\int \|\dot{\mathbf{x}}(t)\|$ (normalized to 1 at $t = \theta$). This corresponds to the distance travelled by the state-vector over time. The dashed line marks the last inflection point, indicating when $\mathbf{x}(t)$ begins to slow down. Reproduced from Voelker and Eliasmith (2018, Figure 6).

network as the dimensionality is increased (see Figure 6.4). Specifically, we perform a standard principal component analysis (PCA) on the state-vector for the impulse response, and vary the order from $q = 3$ to $q = 27$. This allows us to visualize a subset of the state-vector trajectories, via projection onto their first three principal components (see Figure 6.4 (Top)). The length of this trajectory over time distinguishes different values of q (see Figure 6.4 (Bottom)). This length-curve is approximately logarithmic when $q = 6$, convex when $q \leq 12$, and sigmoidal when $q > 12$. To generate this figure we use a delay of $\theta = 10\,\text{s}$, but in fact this analysis is scale-invariant with time. This means that other delays will simply stretch or compress the impulse response linearly in time (not shown).

We remark that the delay network is scale-invariant with the delay length over input frequency, that is, the accuracy for a chosen order is a function of $f \times \theta$ (see units in Figure 6.2 for instance), where f is the input frequency. More specifically, for a fixed approximation error, the delay length scales as $\mathcal{O}\left(qf^{-1}\right)$. We elaborate on these properties in section 6.1.5, and demonstrate our extensions to harnessing arbitrary linear synapse models to improve accuracy in section 6.2.7.

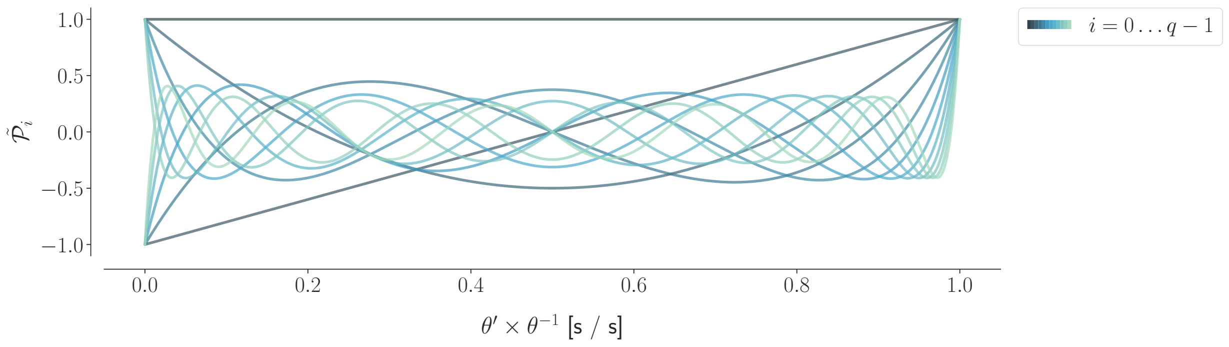


Figure 6.5: Temporal basis functions for the Legendre state-space realization of the Delay Network ($q = 12$). Each line corresponds to the basis function of a single dimension (i) ranging from 0 (darkest) to $q-1$ (lightest). The i^{th} basis function is the shifted Legendre polynomial over $\theta'\theta^{-1}$ with degree i (see equation 6.7). The state-vector of the Delay Network takes a linear combination of these q basis functions in order to represent a rolling window of length θ. These polynomials are a linear transformation of the same polynomials from Figure 6.3.

6.1.3 Padé–Legendre basis

In this section, we reveal a deep connection between the work of Legendre (1782) and Padé (1892). Specifically, the Padé approximants of the delay have an elegant state-space realization whose corresponding temporal basis functions are the shifted Legendre polynomials:

$$\tilde{\mathcal{P}}_i(r) = (-1)^i \sum_{j=0}^{i} \binom{i}{j} \binom{i+j}{j} (-r)^j = \mathcal{P}_i(2r-1), \quad r = \frac{\theta'}{\theta}, \tag{6.7}$$

where $\mathcal{P}_i(r)$ is the i^{th} Legendre polynomial (Rodrigues, 1816). These classical polynomials, shown in Figure 6.5, are *uniquely* defined as the set of orthogonal polynomials over the domain $0 \le r \le 1$ scaled such that $\tilde{\mathcal{P}}_i(1) = 1$. The Legendre polynomials have numerous applications in the physical sciences, for example in the solution to Schrödinger's equation for the hydrogen atom (Hermann and Fleck Jr, 1988), and in describing the function basis for a population of LIF tuning curves (Eliasmith and Anderson, 2003, p. 202).

Although previous connections have been made between these same polynomials and parameter identification in time-delay systems (Hwang and Chen, 1985), we believe this work is the first to discover a simple LTI system—derived by the Padé approximants—that projects a rolling window of its input history onto the shifted Legendre basis. To solve for this system, we determine the state-space transformation of equation 6.4 that rotates equation 6.6 into the form of equation 6.7. This is made possible by an inductive procedure and via linear independence of equation 6.6. This results in the following state-space realization of the Padé approximants for the delay transfer function, $[(q-1)/q]e^{-\theta s}$:

$$\theta\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + Bu(t)$$

$$A = [a]_{ij} \in \mathbb{R}^{q\times q}, \quad a_{ij} = (2i+1)\begin{cases} -1 & i < j \\ (-1)^{i-j+1} & i \geq j \end{cases} \tag{6.8}$$

$$B = [b]_i \in \mathbb{R}^{q\times 1}, \quad b_i = (2i+1)(-1)^i, \quad i, j \in [0, q-1].$$

To provide an example, for the case of $q = 6$, we have the following state-space matrices with linear scaling in dimension and alternating signs in B and in the lower triangle of A:

$$A = \begin{pmatrix} -1 & -1 & -1 & -1 & -1 & -1 \\ 3 & -3 & -3 & -3 & -3 & -3 \\ -5 & 5 & -5 & -5 & -5 & -5 \\ 7 & -7 & 7 & -7 & -7 & -7 \\ -9 & 9 & -9 & 9 & -9 & -9 \\ 11 & -11 & 11 & -11 & 11 & -11 \end{pmatrix}, \quad B = \begin{pmatrix} 1 \\ -3 \\ 5 \\ -7 \\ 9 \\ -11 \end{pmatrix}. \tag{6.9}$$

For the special case of $q = 1$, this reduces to the first-order lowpass $\theta\dot{x}(t) + x(t) = u(t)$, as in equation 2.13 ($\tau \leftarrow \theta$). For $q > 1$, the q^{th} dimension recursively extends the system for $q-1$ to represent a q^{th}-order polynomial of history. In general, the crucial result is that the state-space of this LTI system optimally reconstructs its input history via the shifted Legendre polynomials:

$$\boxed{u(t-\theta') \approx \sum_{i=0}^{q-1} \tilde{\mathcal{P}}_i\left(\frac{\theta'}{\theta}\right) x_i(t), \quad 0 \leq \theta' \leq \theta.} \tag{6.10}$$

This is proven by applying a state-space transformation to the polynomials of equation 6.6.[5] In summary, equation 6.8 has an identical transfer function to that of equation 6.4—namely, the Padé approximants of a pure delay—while providing a state-space that reconstructs the previous θ seconds of input via the Legendre polynomials up to order q.

Figure 6.6 illustrates equation 6.8, discretized using Euler's method, with $q = 6$ dimensions. This design exploits the alternation of signs, and reuses the intermediate computations within the upper and lower triangles of A, by decomposing them into two separate cascading chains

[5] *Cuius rei demonstrationem mirabilem sane detexi hanc marginis exiguitas non caperet.* ~ Pierre de Fermat

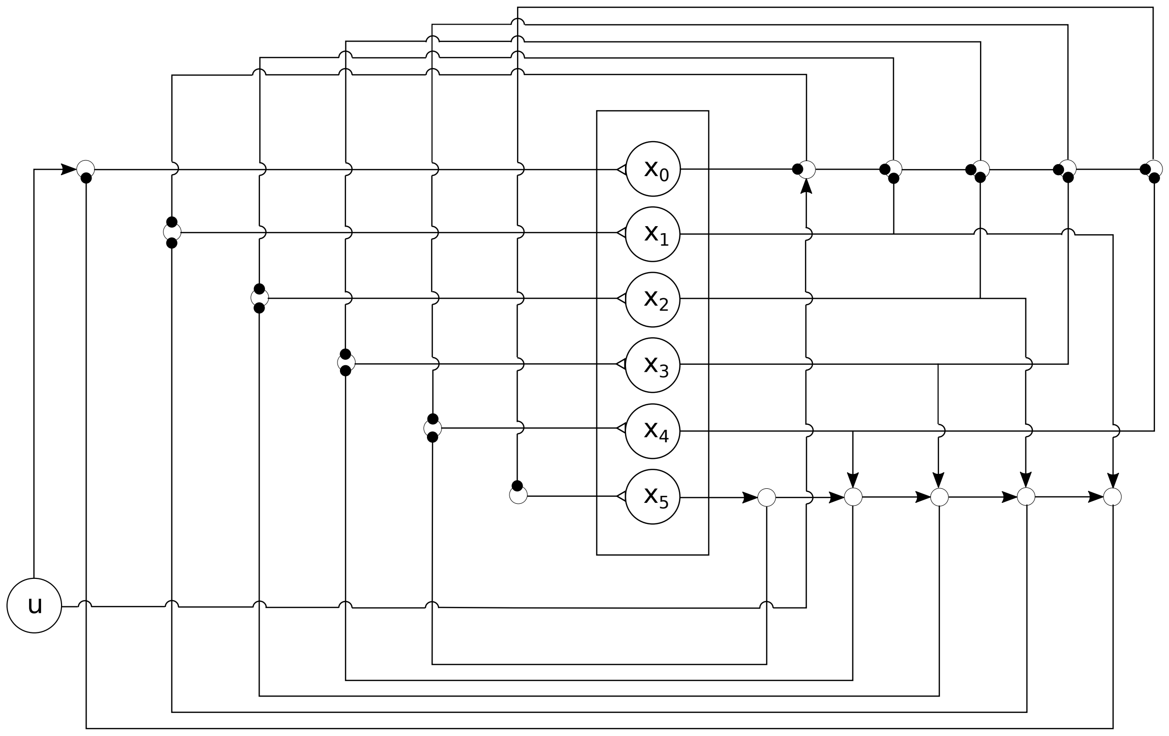


Figure 6.6: Circuit diagram that represents an efficient implementation of the Legendre realization of the Delay Network ($q = 6$). Large circles correspond to each dimension of **x**. Small circles indicate elements that add (arrow head) or subtract (circular head) their inputs. The i^{th} dimension scales its input (triangular head) by $\mathtt{dt}(2i+1)\theta^{-1}$, for $i = 0 \ldots q-1$. Recurrent self-loops, from each dimension back to itself, are omitted for simplicity.

of summations that are then combined by a feedback loop. These same computations are also reused to implement Bu by supplying u to the appropriate intermediate nodes. Increasing the dimensionality of the system by one requires appending $\mathcal{O}(1)$ wires, adders, and state-variables, to the existing circuitry. In total, this circuit requires $\mathcal{O}\left(q\right)$ wires, adders, and state-variables, thus making it linearly scalable in both space and time (see section 6.1.5).

6.1.4 Nonlinear characterization

A delay is a linear dynamical system. However, by virtue of encoding the delay network's state-vector into a heterogeneous population of neurons—as in the first two principles of the NEF—the network also affords the computation of arbitrary nonlinear functions across the rolling window. Recall that a continuous-time rolling window of length $\theta > 0$, which here we denote as $\left[u(t-\theta') : 0 \le \theta' \le \theta\right]$, $u(\cdot) \in \mathbb{R}$, may be *temporally compressed* into a low-dimensional state $\mathbf{x}(t) \in \mathbb{R}^q$, by the following linear time-invariant dynamical system that approximately delays an input signal by θ seconds (repeating equation 6.5, for clarity):

$$\theta\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + Bu(t), \quad A = \begin{pmatrix} -v_0 & -v_0 & \cdots & -v_0 \\ v_1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ 0 & 0 & v_{q-1} & 0 \end{pmatrix}, \quad B = \begin{pmatrix} v_0 & 0 & \cdots & 0 \end{pmatrix}^{\mathsf{T}},$$

where $v_i := (q+i)(q-i)(i+1)^{-1}$, for $i = 0 \ldots q-1$. This compression is derived by applying Padé approximants to $e^{-\theta s}$ about the complex point $s = 0$, which yields an optimal low-degree polynomial expansion of $\mathcal{L}\{u\}(s)$ about its zeroth frequency. The state $\mathbf{x}(t)$ represents the rolling window via a linear combination of time-invariant polynomial basis functions:

$$u(t-\theta') \approx \sum_{i=0}^{q-1} \mathcal{P}_{q-i-1,q}\left(\frac{\theta'}{\theta}\right) x_i(t), \quad 0 \le \theta' \le \theta. \tag{6.11}$$

Repeating the equation for these polynomials (equation 6.6), for completeness:

$$\mathcal{P}_{i,q}(r) = \binom{q}{i}^{-1} \sum_{j=0}^{i} \binom{q}{j} \binom{2q-1-j}{i-j} (-r)^{i-j}. \tag{6.12}$$

A set of polynomials $\{\mathcal{P}\}_{i,q}$ are shown in Figure 6.3. This system implicitly projects its window of history onto a linear transformation of polynomials up to some chosen order. In summary, equation 6.11 determines a linear map from the q-dimensional state, $\mathbf{x}(t)$, to the infinite-dimensional rolling window, $[u(t-\theta') : 0 \le \theta' \le \theta]$, such that equation 6.11 is the optimal pseudo-inverse of the linear compression defined by equation 6.5.

Moreover, every point along the window, along with its Taylor series expansion up to order q, as well as any other integral transform across the rolling window (e.g., Laplace transform), may be expressed using equation 6.11 and equation 6.12 as a fixed dot product with $\mathbf{x}(t)$. To solve for this, let $k(\theta')$ $(0 \le \theta' \le \theta)$ be any desired kernel (e.g., Fourier transform), then:

$$\begin{aligned}(u * k)(t) &= \int_{\theta'=0}^{\theta} u(t-\theta')k(\theta')\,d\theta' \\ &\approx \sum_{i=0}^{q-1} \underbrace{\left(\int_{\theta'=0}^{\theta} k(\theta')\mathcal{P}_{q-i-1,q}\left(\frac{\theta'}{\theta}\right)d\theta'\right)}_{k_i} x_i(t) \\ &= \langle \mathbf{x}(t), \mathbf{k} \rangle\,,\end{aligned} \tag{6.13}$$

where $\mathbf{k} = \left[k_0, \ldots, k_{q-1}\right]^{\mathsf{T}} \in \mathbb{R}^q$ is defined above independently of time. Hence, any integral transform is simply a dot product with $\mathbf{x}(t)$. Similarly, one may obtain Taylor expansions by taking dot products with vectors derived by repeatedly differentiating both sides of equation 6.11 with respect to θ' (not shown). Analogous statements hold for the Legendre realization from the previous section, whose set of polynomials are shown in Figure 6.5.

Crucially, such dot products are precisely what the encoders $\mathbf{e}_i$ compute in equation 2.2. This implies that the PSC of any neuron, by itself, necessarily represents a particular convolution[6] across the window, equivalent to the heterogeneous encoding filters proposed in section 5.1.5. The nonlinearity of each neuron thus provides a nonlinear function of some finite-width filter (Principle 1). More generally, the state $\mathbf{x}(t)$ is *encoded* by a heterogeneous population of neurons, which allows for arbitrary nonlinear functions across the window,

$$y(t) = f(\mathbf{x}(t)) \approx f\left(\left[u(t-\theta') : 0 \le \theta' \le \theta\right]\right)$$

to be *decoded* from its activity (Principle 2). For example, Figure 6.7 visualizes f within the space of $\mathbf{x} \in \mathbb{R}^3$ for the autocorrelation function, $y(t) = u(t-\theta)\,u(t)$.

6.1.5 Scalability considerations

In this section we address several fundamental questions regarding the scalability, or *memory capacity*, of the Delay Network (DN) with respect to q, θ, and the complex frequency of the input signal, $s = 2\pi i f$, for some real frequency, f. For the time being, we disregard the error arising from neural representation by assuming we can make n sufficiently large (see section 3.2.2). This means we match the model description precisely, even when considering more elaborate neuron or synapse models (see chapter 5). The only source of error remaining is our use of Padé

[6]There are infinitely many, since the system is overcomplete when working backwards from $\mathbf{k} \mapsto k(\cdot)$, but we can enforce uniqueness by imposing L2-regularization, for instance.

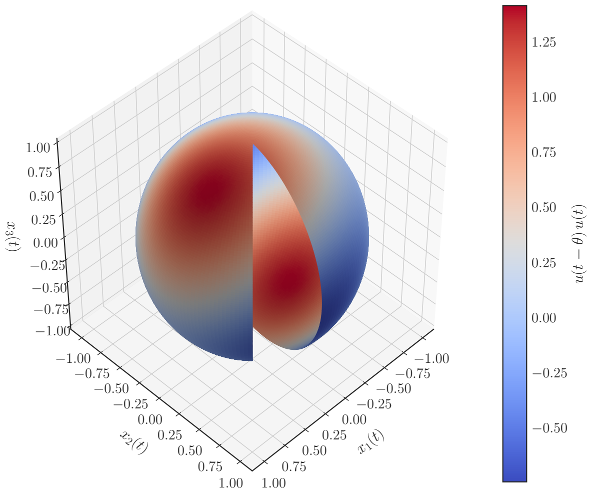


Figure 6.7: Visualizing the autocorrelation function $y(t) = u(t-\theta)\,u(t)$ with $q = 3$. Each point along the surface of the unit-sphere is $\mathbf{x}(t)$, and is coloured according to its corresponding value of $y(t)$ obtained from equation 6.11. A slice of the shell is cut away for visualization.

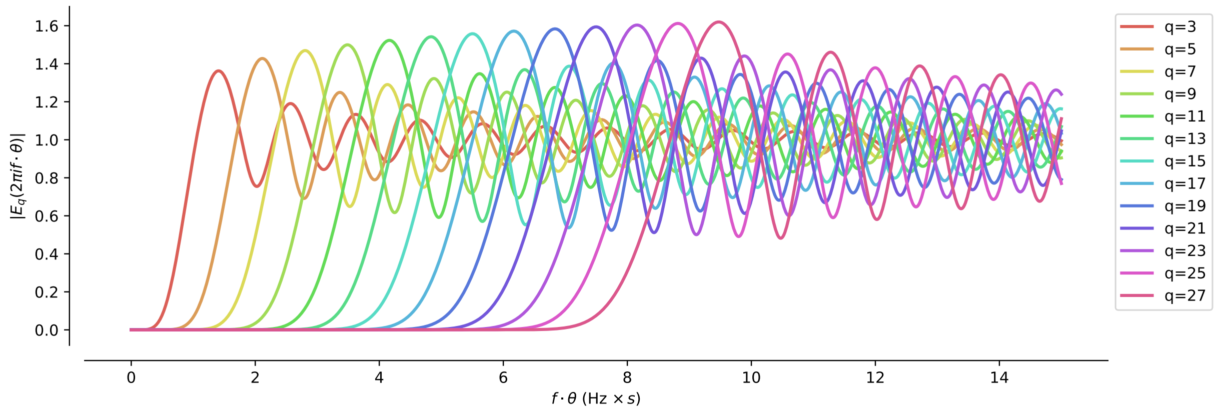


Figure 6.8: Visualizing the error in the Delay Network, $|E_q(2\pi i f\cdot\theta)|$ (see equation 6.14), while varying the dimensionless quantity $f\cdot\theta$ (input frequency multiplied by delay length), and the order of approximation, q.

approximants to render equation 6.2 finite-dimensional. This error term may be expressed in the Laplace domain as follows:

$$E_q(\theta s) = [q-1/q]e^{-\theta s} - e^{-\theta s}, \tag{6.14}$$

where $[q-1/q]e^{-\theta s}$ is defined by equation 6.3 (see Figure 6.8).[7] By linearity of the Laplace transform, it suffices to consider the error at individual frequencies s in order to fully characterize the error of the entire system for any given input. More precisely, by the convolution theorem, we can interpret equation 6.14 as a linear filter in the s-plane, to reveal how each input frequency is amplified, attenuated, or phase-shifted, to contribute to the spectrum of the overall error.[8]

Dimensional analysis quickly reveals an important property of this error: the dimensionless quantity, θs (i.e., the product of time and frequency), fully determines the error (for fixed q). In other words, the "difficulty" of computing a delay is a function of the length of the delay multiplied by the frequency of the input signal being delayed. For instance, delaying a 10 Hz signal for 0.1 s has the same approximation error as delaying a 0.1 Hz signal for 10 s. This is seen by simply rescaling the time-axis by a factor of 100. This is also consistent with the results from Figure 6.2.

This is deeply connected to the observation that in equation 6.4, the delay length θ scales inversely with the gain on A and B. The identification of this control factor can be derived from a more general property of the Laplace transform, $F(\theta s) = \theta^{-1}\mathcal{L}\{f(t/\theta)\}(s)$ for all $\theta > 0$, meaning that the time-scale of any linear filter is rescaled by a gain on the integration. We can exploit this fact more generally to modulate the time-scale of any linear filter on-the-fly (in this case affecting the length of delay; results not shown).

Next, as shown in Figure 6.8, the radius of convergence for the Padé approximants increases linearly with q. Thus, scaling q by some factor means that either the frequency of the input signal or the length of the delay can be scaled by the same factor, while achieving the same error. When θs is outside the radius of convergence, the absolute error is bounded between 0 and 2. This fact simply comes from the observations that $e^{-\theta s}$ is a unit-length complex number, and $[q-1/q]e^{-\theta s}$ is complex with magnitude less than 1. Thus, by the triangle inequality, the magnitude of their difference is bounded above by 2. In the limit as $\theta s \to \infty$, the magnitude of the error converges to 1 as $[q-1/q]e^{-\theta s} \to 0$.

We may interpret these results in the context of equation 6.14 as a linear filter. When θs is small (within the radius of convergence, which scales linearly with q), an input frequency at s radians is delayed almost perfectly for θ seconds. Outside the radius of convergence, the error contains frequencies that are phase-shifted and multiplied (with magnitude bounded above by 2) versions of the input frequencies in that range. As θs increase further, the error contains input frequencies that are simply phase-shifted, without any change in magnitude.

[7] For example, $|E_{21}(2\pi i \cdot 5)| \approx 0.003$ implies that a 21-dimensional DN delays a 1 Hz signal for 5 s with 0.3% error.

[8] Only frequencies that are present within the input can become present in the error.

Lastly, as revealed by equation 6.12, and as shown in Figure 6.3, the basis functions are also dimensionless, in the sense that they are defined over the range $r \in [0,1]$ which maps onto the time-interval $(t - \theta') \in [t - \theta, t]$, where $\theta' = r\theta$, for any $\theta > 0$. This has numerous consequences that are intertwined: (1) the relationship between the input signal and its state-vector is scale-invariant; scaling the input length by some factor, and the delay length by the same factor, results in the exact same state-vector corresponding to a rescaled input history, (2) we can perform computations on the representation of the DN, independently of scale; see section 6.2.4 for an example, and (3) we can reinterpret the same state-vector with respect to any time-scale without changing the state-vector.

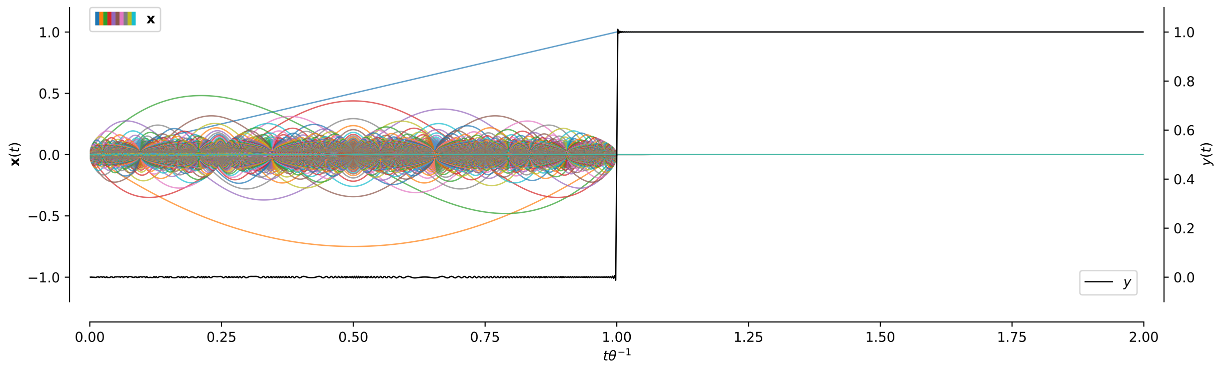


Figure 6.9: Scaling the Delay Network to $q = 10{,}240$ dimensions, simulated for 1,024,000,000 time-steps ($\theta = 5{,}120\,\mathrm{s}$, $\mathrm{dt} = 10^{-5}\,\mathrm{s}$). The input signal is $u(t) = 1$, and so $y(t) \approx u(t - \theta)$ is the Heavide step function centered at $t = \theta$. Each state-variable approximates the integral of a Legendre polynomial, and $y(t) = \sum_i x_i(t)$ sums across all dimensions at each moment in time.

Figure 6.9 reveals that the delay network scales to over 10^4 dimensions to efficiently maintain information across a simulation lasting more than 10^9 time-steps. This is demonstrated by implementing the design from Figure 6.6 in C++, which takes $\mathcal{O}(q)$ time and memory per simulated time-step using a traditional CPU.

Taking neural constraints into consideration, we do need to factor θs into its dimensional quantities. As shown in section 5.2.2, the frequency of the state-vector being represented can matter. This concern is resolved by the use of Poisson spiking, integrate-and-fire neurons, or any other solution to state discrepancy (section 3.2.1). As well, longer values of θ are more susceptible to the integration of spiking noise, in the form of “drift.” Lastly, when discretizing the system in time (e.g., ZOH) or using a non-canonical realization (e.g., balanced), larger values of q require denser interconnectivity ($q \times q$ transform matrices) between all q dimensions, leading to $\mathcal{O}(q^2)$ time and memory requirements. Such considerations are addressed in section 6.2.5 by stacking multiple DNs together to form a Deep Delay Network (DDN).

To summarize, the properties of scaling θ are physically coupled to considerations of which input frequencies need to be delayed, and to what degree of precision. The accuracy is a function of θs. And, for a fixed level of precision, θs scales linearly with q. In all cases, the representation employed by the DN (i.e., the linear relationship between the input history and the state-vector) is scale-invariant in θ.

6.2 Applications

We have developed rigorous theory to understand the Delay Network (DN) in terms of its linear dynamics, and its nonlinear encoding of time, and used the NEF to present a spiking example in Figure 6.1. Now, we turn to several concrete applications that are of relevance to those studying reservoir computing, FORCE learning, deep RNNs and stacked LSTMs, as well as the cognitive modelling and computational neuroscience communities at large.

6.2.1 Reservoir computing

Here we show that the NEF provides a principled way to optimize a particular RNN architecture, by mapping the desired dynamical state onto a latent representation within a factorable connection weight matrix (also see sections 2.1.4 and 2.2.4). This enables us to train networks that outperform Reservoir Computing (RC) methods—using either spiking (LSM; Maass et al., 2002) or rate-based (ESN; Jaeger, 2001) neurons—in memory capacity and nonlinear computation, while reducing the simulation time and space requirements by a factor of $\mathcal{O}(n)$. Our approach can learn both online and offline, readily incorporates detailed synapse models, supports various spiking and rate-based neuron models, and may be compiled onto analog and digital neuromorphic hardware.

Linear benchmark

Training a network to implement a delay is a natural way to measure the dynamical system's memory capacity—its ability to maintain a history of its input signal—which is needed to compute any function over past inputs. Indeed, this task was considered by ESNs in one of the earliest demonstrations of the RC paradigm (Jaeger, 2002). In its purest analog form, a delay line is attempting to encode an infinite amount of information, which is why it is a challenging and useful benchmark to consider. Theoretical and experimental results in the past have pointed to the limited capability of random feedback to maintain memory (Joshi and Maass, 2005; Mayor and Gerstner, 2005; Dambre et al., 2012; Wallace et al., 2013; Inubushi and Yoshimura, 2017; Schuecker et al., 2018), in particular finding that linear feedback *maximizes* memory capacity (Mitra and Stark, 2001)—as derived in section 6.1.1, and considering the linearity of a delay line—while being at odds with the nonlinearities that are required to support useful computations across the memory.[9] This is consistent with our findings below. Moreover, the DN poses a natural way out of this predicament, by using nonlinearities to approximate the required linear feedback, without sacrificing the ability to nonlinearly transform the window (section 6.1.4 and below).

For our benchmark task, weights were trained and validated using randomly sampled 25 Hz band-limited white noise inputs. In addition, full-spectrum white noise was added to the network during both training and testing. Accuracy was measured by normalizing the root-mean squared error against the root-mean squared target (NRMSE; Lukoševičius, 2012b). As well, 95% confidence intervals were bootstrapped and plotted using Seaborn (Waskom et al., 2015). We ported a Python implementation of the ESN from Lukoševičius and Jaeger (2009) to Nengo, and implemented it analogously to the DN. Nengo allows us to consider populations of either spiking LIF neurons or non-spiking neurons with various rate curves, without any additional changes to the model specification. In particular, LSMs were implemented by replacing the tanh units with LIF spiking neurons, making them comparable to our NEF networks but with full-rank weight matrices.

The hyperparameters of each method were optimized using 200 iterations of Hyperopt (Bergstra et al., 2013) with 3 trials per iteration, to maximize the validation error for 200 ms delays (see Table 6.1). Hyperparameters include an overall gain on the recurrent feedback matrix (gain), a normalization factor for the input (radius), time-constants on both the readout and recurrent filters (τ_{readout}, $\tau_{\text{recurrent}}$), L2-regularization parameters for applicable least-squares problems ($\sigma^2_{\text{readout}}$, $\sigma^2_{\text{recurrent}}$), and the dimensionality of the delay network (q).[10]

In all cases, we construct a reservoir of 1,500 neurons, and then train separate linear readouts to approximate various delays ranging from 100–200 ms ($\texttt{dt} = 1$ ms). For the NEF case, the prescribed dynamical system is a $\theta = 200$ ms delay, implemented by mapping equation 6.4 onto a discretized lowpass (see equation 5.9). In section 6.2.3 we show that the delay length can be trained from raw data.

[9]We thank Kwabena Boahen for pointing out this connection and a few of these references.

[10]Hyperopt was used to the benefit of LSMs and ESNs. All hyperparameters (apart from q) had minimal effect on the DN's performance compared to the usual defaults in Nengo.

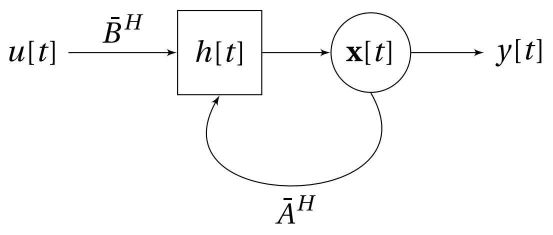


Figure 6.10: Delay Network (DN) model for a digital architecture. The synapse $h[t]$ is driven by $\bar{A}^H \mathbf{x}[t] + \bar{B}^H u[t]$ to yield the state $\mathbf{x}[t]$. This state is nonlinearly encoded by a heterogeneous population of neurons and subsequently decoded to estimate the desired $y[t]$.

Importantly, the same networks are used to compute all of the different delays reported. We thus conceptualize the DN as an optimized "low-dimensional reservoir." Compiling all of the equations of this network, for clarity (see Figure 6.10):

$$
\begin{aligned}
v_i &= \frac{(q+i)(q-i)}{i+1}\theta^{-1}, \quad i = 0 \ldots q-1 \\
A &= \begin{pmatrix} -v_0 & -v_0 & \cdots & -v_0 \\ v_1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ 0 & 0 & v_{q-1} & 0 \end{pmatrix} \in \mathbb{R}^{q \times q} \\
B &= \begin{pmatrix} v_0 & 0 & \cdots & 0 \end{pmatrix}^{\mathsf{T}} \in \mathbb{R}^q \\
\bar{A} &= \sum_{i=0}^{\infty} c_i A^i = \sum_{i=0}^{\infty} \frac{(A\mathrm{dt})^i}{i!} = e^{A\mathrm{dt}} \\
\bar{B} &= \left(\sum_{i=1}^{\infty} c_i A^{i-1} \right) B = A^{-1}\left(\bar{A} - I\right) B \\
a &= e^{-\frac{\mathrm{dt}}{\tau}} \\
\bar{A}^H &= \frac{1}{1-a}\left(\bar{A} - aI\right) \\
\bar{B}^H &= \frac{1}{1-a}\bar{B}.
\end{aligned}
\tag{6.15}
$$

We remind the reader that A and B may be transformed by any of the state-space realizations from section 4.1, or starting with the Legendre realization from section 6.1.3, before proceeding with the remaining numerical machinery. The final matrices $(\bar{A}^H, \bar{B}^H)$ determine the transformations to be implemented using Principles 1 and 2 of the NEF, which in turn will nonlinearly encode $\mathbf{x}[t]$ by our extensions to Principle 3.

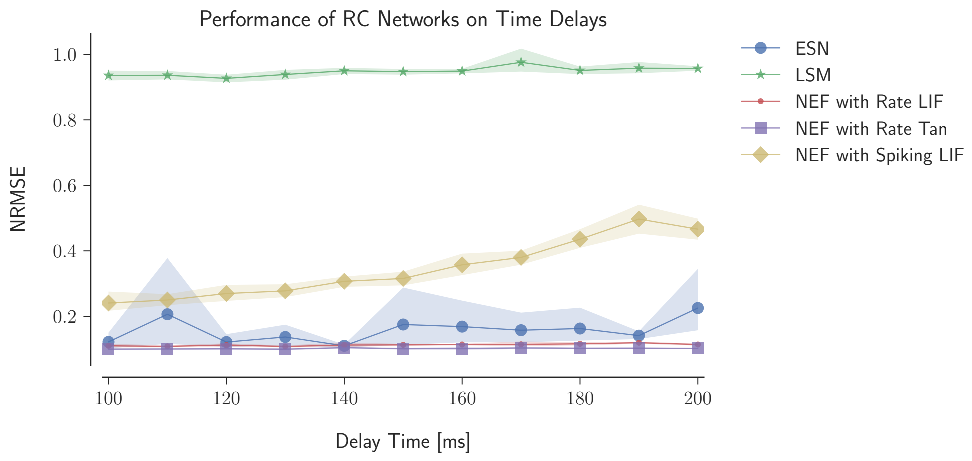


Figure 6.11: Performance from training various linear readouts to compute delays of 25 Hz band-limited white noise (bootstrapped across 10 trials). Each line corresponds to a single reservoir of 1,500 neurons, either randomly connected (in the case of ESNs and LSMs), or specifically engineered (in the case of the NEF).

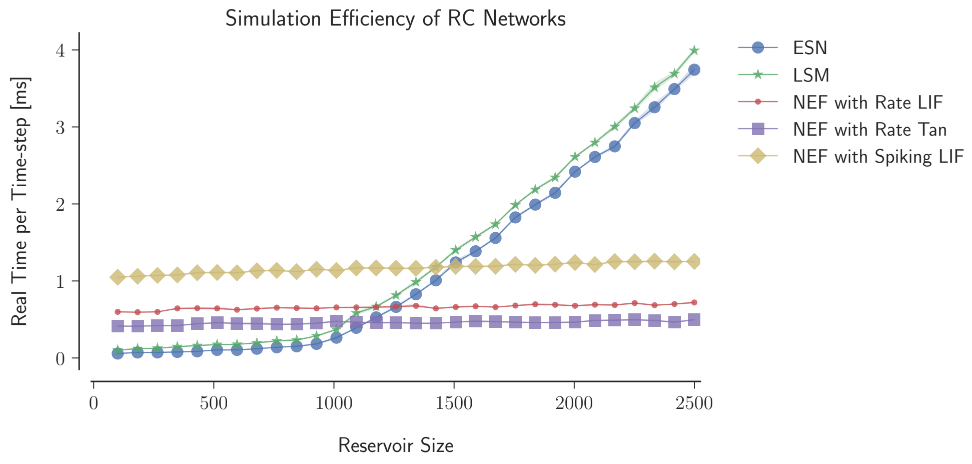


Figure 6.12: Cost of simulating each RC network as the reservoir size is increased (bootstrapped across 10 trials). Conventional RC approaches require $\mathcal{O}\left(n^2\right)$ space and time, while the NEF improves this to $\mathcal{O}\left(n\right)$ for constant dimensionality.

As shown in Figure 6.11, the NEF's performance is slightly better than ESNs for both LIF rate neurons and tanh rate neurons, and significantly better than LSMs for spiking LIF neurons. This demonstrates that, in terms of memory capacity, the DN as a low-dimensional reservoir not only outperforms RC in the rate-based case, but even performs comparably to ESNs when using spikes. The task is shown to be completely outside the grasp of LSMs (exceeding 90% error), due to the difficulty of the computation and the unreliability of randomized spiking feedback; Hyperopt minimizes both the gain and regularization hyperparameters of the LSM to keep its output close to zero, as this minimizes validation error.[11] The success of the DN should not be surprising given that we have mapped the ideal delay's dynamics onto the reservoir. Nevertheless, as we will show below, the readouts are capable of computing nonlinear functions across the delay interval, from the same reservoir. Informally, assigning some particular dynamics to the reservoir does not disable the network from computing other similar functions (thereby permitting nonlinear combinations of such functions to be trained from data).

These networks were also simulated while varying the number of neurons from 100 to 2,500, in order to measure the real-time cost of simulation (see Figure 6.12). We again note that traditional RC suffers from scaling issues since the recurrent weight matrices have $\mathcal{O}\left(n^2\right)$ coefficients. Consequently, the NEF is more resource efficient than these RC methods by a factor of $\mathcal{O}\left(n\right)$. RC networks often include a sparsity constraint of 20% connectivity, which is still $\mathcal{O}\left(n^2\right)$, in order to improve performance (Lukoševičius, 2012a,b). We also considered ESNs with constant sparsity that balance resource constraints with NEF networks, but found that they did not perform comparably (not shown). Furthermore, we found that the ESN breaks down for numbers of neurons as few as q neurons, while in fact q linear rate neurons (with linearly independent encoders) will suffice to perfectly implement equation 6.4, as q linear state-variables are in exact correspondence with the ideal linear system (after the appropriate discretization of equation 5.9) (see section 6.2.2).

Another key finding is that, as shown in Table 6.1, Hyperopt discovers that a radius of ≈ 24.3 performs the best with the ESN. This has the effect of scaling the domain of the tanh curve from $[-1, 1]$ to $\approx [-0.04, 0.04]$, which importantly is well-approximated by a straight line, that is, $\tanh x \approx x$ across this domain. Thus, Hyperopt is indirectly leveraging the fact that the ESN's memory capacity is maximized when its neurons *are linear*, consistent with Dambre et al. (2012). Crucially, this limits the ability of the ESN to perform nonlinear computations across the delay interval, as we now show.

[11]As additional validation, lower input frequencies or shorter delay lengths were possible with the LSM.

	ESN	LSM	NEF with Rate LIF	NEF with Rate Tan	NEF with Spiking LIF
Gain	1.33	3.15×10^{-3}	-	-	-
Radius	2.43×10^{1}	1.36	4.64	1.29×10^{1}	5.77×10^{-1}
τ_{readout}	6.87×10^{-3}	6.04×10^{-2}	6.06×10^{-2}	8.73×10^{-2}	2.18×10^{-2}
$\tau_{\text{recurrent}}$	2.14×10^{-3}	6.91×10^{-2}	6.26×10^{-2}	9.37×10^{-2}	7.40×10^{-2}
$\sigma^2_{\text{readout}}$	2.96×10^{-6}	3.29×10^{-2}	2.06×10^{-3}	4.80×10^{-3}	4.50×10^{-2}
$\sigma^2_{\text{recurrent}}$	-	-	2.00×10^{-4}	3.98×10^{-4}	3.76×10^{-2}
q	-	-	20	26	23

Table 6.1: Hyperopt parameters for the linear benchmark in section 6.2.1.

Nonlinear benchmark

To demonstrate our ability to compute nonlinear window functions, we consider the function from section 6.1.4, $y(t) = u(t-\theta)\,u(t)$, visualized in the context of a DN's state-vector in Figure 6.7. When integrated over time, this is the autocorrelation of u with lag θ, which has numerous applications in signal processing (e.g., detecting repeating events). We fix $\theta = 0.1$ s across all experiments. To compute this function accurately, we sample a proportion of the encoders from the diagonal combinations of $\mathcal{P}_{i,d}(0)$ and $\mathcal{P}_{i,d}(1)$ (Gosmann, 2015). However, the particular choice of function is not of importance, as the training can be data-driven, or analyzed using theory from sections 6.1.2 and 6.1.4.

Each input $u(t)$ is sampled white noise, band-limited with a cutoff frequency of 30 Hz. To optimize for the decoders, we map q-dimensional evaluation points onto desired target outputs using equation 6.11, and apply Nengo's regularized least-squares solver, which bypasses the need to explicitly simulate the network on any input signals.

To implement this system as a spiking (or non-spiking) recurrent neural network, we use the same set of transformations from equation 6.15. The model architecture of this delay network is, again, depicted in Figure 6.10, whose representation and transformations are realized using the NEF. For this experiment, we considered the use of both sigmoidal (non-spiking) neurons, and spiking LIF neurons.

We again used Hyperopt (Bergstra et al., 2015) to explore the space of model hyperparameters (e.g., q, τ, input gain, recurrent gain, L2-regularization) across 100 iterations containing 10 trials each. Each network consisted of 1,000 neurons, simulated with a time-step of $\mathrm{dt} = 1$ ms. Each trial used a training signal of length 10,200, a testing signal of length 2,200, and the first

200 outputs were discarded. We then cross-validated the best set of hyperparameters (in terms of mean NRMSE across all test signals) using another 25 trials.

We obtain a mean NRMSE of 5.9% for the sigmoid DN, 51.8% for the spiking DN, and 84.3% for the tanh ESN. Reducing the input frequency from 30 Hz to 15 Hz improves the ESN's accuracy to be on par with the non-spiking DN, and thus we attribute this difference to the inherent difficulty of autocorrelating a high-frequency signal (relative to θ) using random feedback weights, as opposed to using optimally derived weights as in the DN. In addition, trials took on average 5.10 s for the sigmoid DN, 6.44 s for the spiking DN, and 17.7 s for the ESN. This difference is a consequence of not simulating the DN for training, and from using factorized weight matrices (i.e., encoders and decoders) to simulate the DN. These results are consistent with that of the linear benchmark, except for the additional observation that here the spiking DN outperforms the rate ESN. This is because, as explained previously, the ESN's memory capacity requires linear tuning, which is at odds with the nonlinearities required by functions such as autocorrelation. LSMs were again unable to perform the task.

6.2.2 Force learning

We also compared our NEF solution to both FORCE (Sussillo and Abbott, 2009) and full-FORCE (DePasquale et al., 2018) networks (also see sections 2.1.4 and 2.2.4). FORCE networks take the same essential ingredient of RC networks, namely the inclusion of high-dimensional random feedback, except they make an important addition to the model: the target output is learned—online via RLS—to be simultaneously re-encoded back into the network (see Figure 2.3). Full-FORCE takes this a step further, by using the (pre-filtered) currents from from the previous FORCE network as a training signal for a separate "full-FORCE" network. In the first step, the FORCE network does not do any online learning, but instead receives the exact ideal teaching signal. The second step then allows for additional degrees of freedom to be optimized, to find the *full weights* that reconstruct the ideal currents from the first network – again, online using RLS. This is similar in many ways to Tripp and Eliasmith (2006), but maintains the same idea as in FORCE, in particular the re-encoding of the target state. The main difference from FORCE is that instead of learning a low-rank (one-dimensional) decode-encode of the target, full-FORCE learns a full-rank approximation to an encoding of that same target.

For this experiment, we first ported the same implementation described in DePasquale et al. (2018) to Nengo, and verified that it is works as intended (using the same models and parameters) by teaching the network to produce decaying oscillations in response to unit impulses. We compared this to the standard FORCE approach—which we refer to as "classic-

FORCE"—and verified that it performed slightly worse than the full-FORCE network, but still better than an equivalent reservoir computer. Since Nengo allows for the substitution of various spiking and non-spiking neuron models, we further validated both implementations with spiking neurons as well, and obtained reasonable levels of accuracy, similar to DePasquale et al. (2016); Thalmeier et al. (2016); Nicola and Clopath (2017).

However, we found that simply modifying the target signal to be a delayed version of its low-frequency input signal, posed a significant challenge to these networks. Thus, the learning rate was lowered from 1 to 10^{-3}, and the time-step set to $\texttt{dt} = 5\,\text{ms}$, which we found to help regularize the solution. We thus also considered a baseline approach: we took the original FORCE network, but removed the feedback loop that re-encodes the learned output. This makes it equivalent to an ESN with a slightly different method of distributing the weights, while learning the decoders online. We refer to this last method as "no-FORCE".

We now compare all three of these to an idealized NEF implementation of the DN (equation 6.4), consisting of just $n = 6$ linear units, coupled to one another by the discretized mapping of equation 5.9, as summarized by equation 6.15 ($q = 6$). In this scenario, the only error that remains is that of equation 6.14 arising from the use of Padé approximants to render the delay line finite-dimensional. Each FORCE network consists of $n = 500$ non-spiking tanh neurons (and $\mathcal{O}\left(n^2\right)$ weights). The network is given 10 s of training data per trial. In all cases, the training and test signals are randomly sampled 1 Hz band-limited white noise signals, with 5 s of test data per trial. We compare all four networks by sweeping θ across 0.01–1 s, with 10 trials at each value of θ (bootstrapped 95% confidence intervals).

The results in Figure 6.13 illustrate that the NEF's six rate neurons outperform all of the FORCE networks. The full-FORCE network performs well relative to classic-FORCE and no-FORCE for short delays ($\theta < 0.1$ ms). For longer delays ($\theta > 0.1$ s), the classic-FORCE network performs well relative to the other two, but still with error-rates approaching 100% as $\theta \to 1$ s, or 200 time-steps. This reveals a situation in which training the network to re-encode its target output can hinder performance. The NEF solution proposes a means of understanding this phenomenon. In particular, the target output is only one dimension among six orthogonal dimensions that must all be encoded into the state of the network. Focusing on this one dimension and letting the randomized feedback attempt to fill in the rest, leads to competing objectives between the low-dimensional linear feedback required for optimal memory capacity and the high-dimensional chaos induced by nonlinear random feedback.

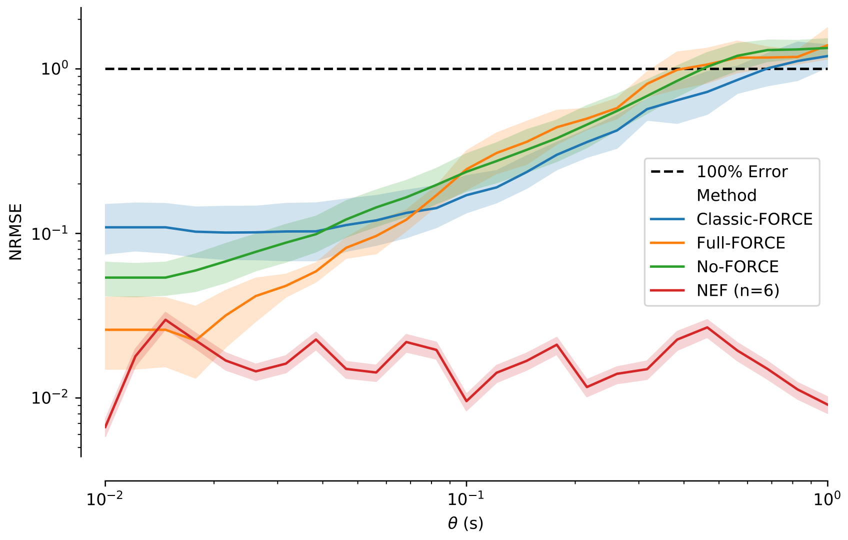


Figure 6.13: Comparison of several FORCE learning methods versus the NEF on a delay task. Networks are trained to delay a time-varying input by θ seconds. Each FORCE network consists of 500 tanh neurons, while the NEF network is 6 linear neurons. See text for details.

6.2.3 Long short-term memory

Long Short-Term Memory (LSTM; Hochreiter and Schmidhuber, 1997; Gers et al., 1999) networks address the problem of storing information across long intervals of time by using an explicit memory unit to alleviate the vanishing and exploding gradient problems (Bengio et al., 1994; Bengio and Frasconi, 1994). Each memory unit implements a small-scale dynamical system—similar *in spirit* to equation 6.15—referred to as an "LSTM cell," with a number of free parameters that control its nonlinear transfer function. This is replicated n times to create a single layer. Each layer is then stacked k times to form a "stacked LSTM," with each consecutive layer representing increasingly higher-order dynamical transformations of its previous inputs (Graves et al., 2013).

Since all of these mechanisms are differentiable, backpropagation through time (BPTT) may be applied end-to-end from raw data to optimize all of the parameters in a black-box manner (see section 2.1.1). There are many variants of LSTMs, but they support essentially the same class of computations (Jozefowicz et al., 2015). Recently, they have found numerous successes in sequence learning, machine translation, and speech (Graves et al., 2013; Sutskever et al., 2014; Cho et al., 2014b; Bahdanau et al., 2014), leading to a widespread adoption of LSTM networks as the RNN of choice (LeCun et al., 2015).

However, we find that when given a continuous-time delay task, the performance of stacked LSTMs suffer catastrophically in the number of time-steps that roll through the memory. This motivates the substitution of each "LSTM cell" with our Delay Network (DN)—explored extensively throughout this chapter—and its subsequent training via backpropagation in an equivalent manner. We now discuss these experimental results.

Memory capacity

Our first set of experiments are designed to evaluate the memory capacity of stacked LSTMs relative to stacked Delay Networks of equivalent resource usage. For this, we use an off-the-shelf implementation of a stacked LSTM (Keras; Gulli and Pal, 2017), and construct $k = 3$ layers with $n = 50$ cells each. Each layer is fully-connected to the next, and uses all of the default settings (e.g., tanh activations). The final layer likewise consists of a tanh activation unit for each output.

To evaluate the continuous-time memory capacity, the input data is white noise, band-limited to 30 Hz, starting at 0, and normalized to an absolute range of $[-1, 1]$. The output data is a 50-dimensional vector representing a uniform arrangement of delayed inputs between 0–0.2 s.[12] The data set consists of 256 samples, each 1 s long. This data is randomly partitioned into 50% training and 50% testing. The training data is further partitioned into a separate random 25% sample used to report validation accuracy during training. Backpropagation through time is carried out using the Adam optimizer (Kingma and Ba, 2014) with respect to the MSE loss function. Training is parallelized using Keras and TensorFlow (Abadi et al., 2016) across four Nvidia Titan Xp GPUs (12 GB each).[13]

We found that, for a time-step of $\mathtt{dt} = 2$ ms, backpropagation could find adequate parameters to solve this task – that is, the LSTM could in fact accurately represent the entire delay interval consisting of $\bar{\theta} = 100$ time-steps with an NRMSE of about 10% (not shown). However, after decreasing the time-step, by an order of magnitude, to $\mathtt{dt} = 200\ \mu$s—while increasing the length of data by the same factor so that the data still represents the exact same 1 s signals—the performance collapses; accuracy exponentially decays as a function of delay length across the $\bar{\theta} = 1{,}000$ time-step window (see Figure 6.14 (b)).

We then took the exact same code and network specification – but replaced each LSTM cell with a "Delay Network (DN) cell", defined as follows. We consider the q-dimensional LTI

[12]Therefore, the highest frequency component oscillates at most six times, within any given delay interval.

[13]This took approximately 14 hours in our largest experiment, in stark contrast to the sub-second time that it takes to train a spiking delay network using Nengo.

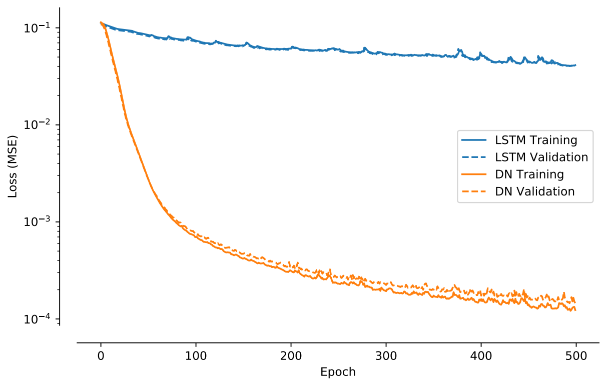
system described by equation 6.5, with a balanced state-space (section 4.1), and discretized using Euler's method:

$$\mathbf{x}[t+1] = \bar{\theta}^{-1}\left(A\mathbf{x}[t] + Bu[t])\right] + \mathbf{x}[t].$$

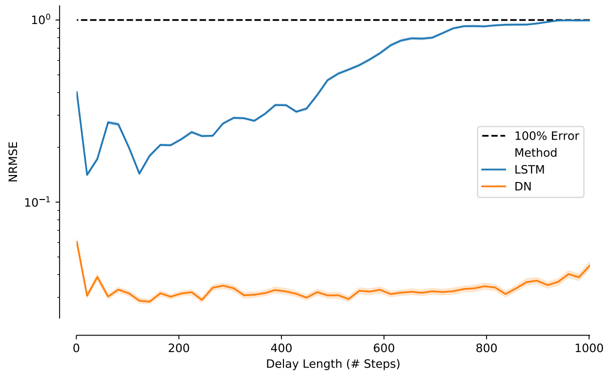
This discretization assumes $\bar{\theta}$ is sufficiently large, such that $\texttt{dt} \approx \bar{\theta}^{-1}$. This "trick" enables us to include $\bar{\theta}$ as a free parameter, unique to each cell, and backpropagate along it – thus learning the delay length of each cell. The unit of this discretized $\bar{\theta}$ is the number of time-steps. This is repeated across all cells within the same layer, while *sharing* the (A, B) matrices across each cell within the same layer (akin to weight-sharing in a convolutional neural network). Finally a tanh nonlinearity is added to each unit, that receives input from all state-variables across the same layer, thus supporting nonlinear computations across a mixture of scaled Legendre bases (see sections 6.1.3 and 6.1.4). For small values of q, the DN cell is roughly comparable to the resource requirements of the original stacked LSTM model, using q state-variables, $\mathcal{O}\left(q^2\right)$ multiply-adds, and 1 nonlinearity with qn weights, per cell.

Each DN cell receives a one-dimensional input. The trainable parameters are the weights between layers, and the delay lengths $\bar{\theta}$ within each cell. In this experiment, we disable training on the shared (A, B) weights. The overall architecture remains fixed with n cells stacked k times. The final output layer consists of linear activation units, since tanh has already been applied at this point. Finally, we fix $q = 9$, initialize the encoding weights of each cell to 1 for the first layer and n^{-1} for all subsequent layers (i.e., the reciprocal of the fan-in), distribute $\bar{\theta}$ values according to $\mathcal{U}[100, 1000]$, and set the weights projecting to each tanh using the C matrix from equation 6.5 with zero weights for all other state-variables from outside the cell. In other words, each cell is *initialized* to approximate $\tanh u[t - \bar{\theta}]$, where $u[\cdot]$ is the cell's mean input. Backpropagation then refines the values of $\bar{\theta}$ and learns to mix weighted nonlinear combinations of inputs and outputs between layers.

Running the exact same code and analysis, on the exact same training, validation, and testing data, reveals a dramatic difference in training time between the two approaches. We found that the stacked DN takes 52.5 s per epoch to train, compared to 102.6 s per epoch for the stacked LSTM. Furthermore, the DN outperforms the LSTM in every measure of accuracy. Figure 6.14 (a) demonstrates nearly three orders of magnitude reduction in MSE across both training and validation, while converging much more rapidly to the ideal solution. Figure 6.14 (b) displays the NRMSE on the test data as a function of delay length (unit of time-steps). We see that the DN architecture achieves nearly uniform accuracy across the delay interval, while the equivalently-sized LSTM cell architecture approaches 100% error rates towards the end of the window. This illustrates that the stacked LSTM struggles to memorize low-frequency signals (relative to the time-step) across long intervals of time. In contrast, this task is natural for the stacked DN, as its state represents a q-degree polynomial expansion of input history (see Figure 6.3). We now proceed to demonstrate that backpropagation enables



(a) Training & Validation



(b) Testing

Figure 6.14: Performance of a stacked LSTM versus a stacked Delay Network (DN) on a delay task. (a) Training and validation rapidly converge for the DN but not for the LSTM. (b) Testing NRMSE is nearly uniform across the delay interval for the DN, but grows exponentially for LSTMs (linear on a log-scale) up to a maximum of 100% error. See text for details.

stacked DNs to outperform stacked LSTMs even on tasks that are not readily supported by the initial configuration of the network.

Predicting chaotic time-series

To asses the performance of each network on a continuous-time prediction task, we consider a synthetic dataset called Mackey-Glass (MG; Mackey and Glass, 1977): a chaotic time-series described by a nonlinear delay-differential equation.[14] The MG data is generated using a discrete time-delay of $\bar{\tau} = 17$ (each time-step is 1 unit of time). The desired output is a look-ahead (prediction) of 15 time-steps in advance (see Figure 6.15). We simulate this for 5,000 time-steps after removing the first 100 step transient. We repeat this 128 times, each time starting from initial random conditions. The entire dataset is then centered to have a global mean of zero. Next, the dataset is randomly split into 32 training examples, 32 validation examples, and 64 testing examples.

We use the same networks from the previous experiment, but with $k = 4$ layers of $n = 100$ cells each. For the DN cells, we make all parameters trainable (including the A, B matrices shared across cells within the same layer). We set $q = 6$ and initialize $\bar{\theta} \in \mathcal{U}[25, 50]$ to account

[14]Our code was adapted from the 2015 Deep Learning Summer School: https://github.com/mila-iqia/summerschool2015/blob/master/rnn_tutorial/synthetic.py.

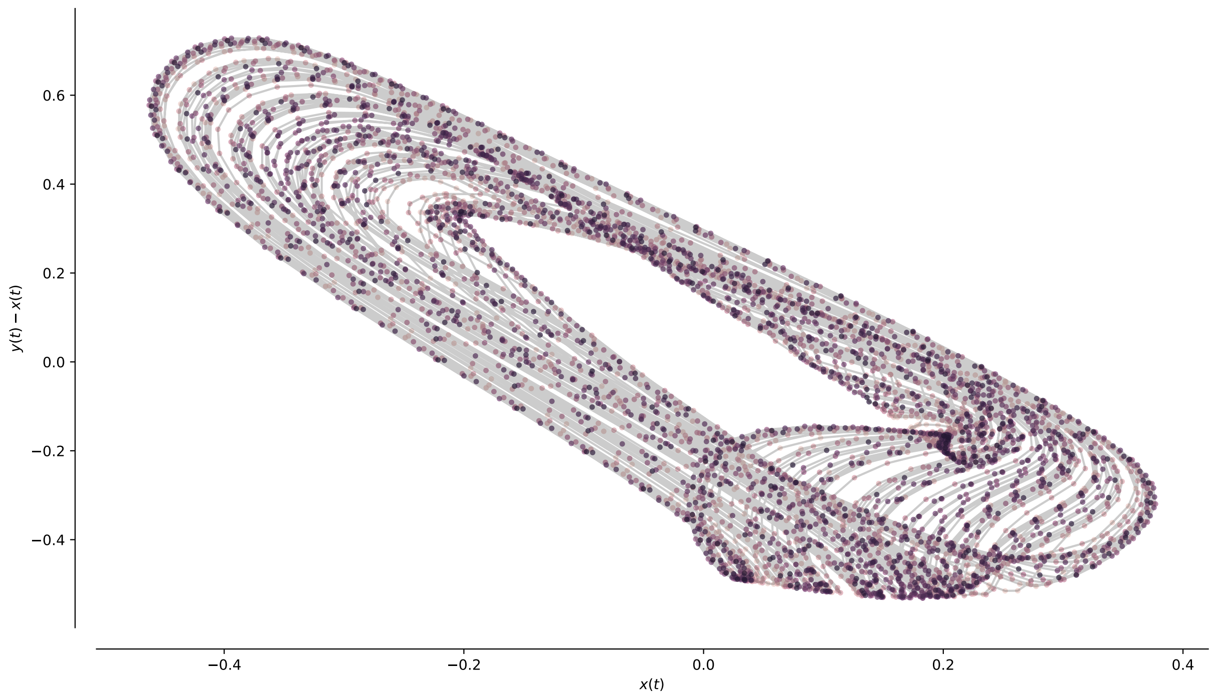


Figure 6.15: Example test case from the Mackey-Glass (MG) dataset ($\bar{\tau} = 17$). x is the input. y is the correct prediction 15 time-steps into the future. We plot x versus the difference between the required prediction and the input, $y - x$, which also shows a two-dimensional delay-embedding of the attractor. The signal consists of 5,000 time-steps (0 is lightest, 4,999 is darkest), with a line connecting adjacent time-points.

for the shorter time-scale of this dataset. We initialize the remaining weights using standard Keras weight initializers.

To successfully solve this problem, the RNN must implicitly do two things. First, it must model the dynamics of the state-trajectory to infer the underlying state from a one-dimensional observation. A theorem due to Takens (1981) states that this is made possible by delaying the input some constant number of times (dependent on $\bar{\tau}$) and then finding the correct static nonlinear transformation of this “delay embedding.” Second, it must predict the future state of this trajectory using its estimate of the current state. The last step is generally infeasible due to the chaotic nature of the MG system; *any* finite error in the state estimate is exponentially magnified over time (the “butterfly effect”; also see section 3.1.4). This makes a look-ahead of 15 steps particularly challenging for a relatively small network, as its reconstruction of the chaotic manifold must become exponentially more precise to see linear payoffs in accuracy.

Method	NRMSE	Parameters	Training time (s / epoch)
LSTM	7.084%	282,101	50.0
DN	6.783%	270,769	30.5
Hybrid	6.097%	291,285	40.5

Table 6.2: Test accuracy and training time for various RNNs on the Mackey-Glass (MG) dataset.

This insight motivates the inclusion of a third "hybrid" method that interleaves the two previous methods. Specifically as in '2121', where '2' is the DN cell and '1' is the LSTM cell. All three methods are trained across 500 epochs using the Adam optimizer. In this case, to minimize overfitting, we keep only the model from the epoch that has the highest validation score.

Test performance and training times are summarized in Table 6.2. We see that the DN outperforms the LSTM in accuracy and training time. We suspect that this is because the DN more readily supports a delay-embedding within its 6-dimensional state. However, the hybrid method outperforms both in accuracy and the LSTM in training time. Thus, the LSTM cell affords some dynamical primitive that may be leveraged by networks of DN cells (and vice-versa). This suggests that LSTMs and DNs may have complementary functions as building blocks within deep RNNs. And, as shown in our first experiment, the DN provides improved scaling through time with respect to lower frequencies across longer continuous time-intervals.

6.2.4 Detecting patterns

We now demonstrate an application of our theory of temporal representation in the delay network (section 6.1.2). Specifically, we show that the delay network can detect the existence of repeating patterns in its stimulus. Given an arbitrary scalar signal, $u(t)$, we define a measure of k-*periodicity* for natural $k \in \mathbb{N}_{\geq 1}$, with respect to some finite interval length $\theta > 0$, and sub-interval length $\gamma = \theta k^{-1}$, as follows:

$$p_k(t) = \sqrt{\frac{1}{\gamma} \int_{\tau=0}^{\gamma} \left(\frac{1}{k} \sum_{i=0}^{k-1} u\left(t - \theta + i\gamma + \tau\right) \right)^2 d\tau}. \tag{6.16}$$

This can be computed directly by: (1) partitioning the last θ seconds into k equally sized segments of length γ, (2) averaging them all together to obtain a superimposed signal of length γ, and (3) taking its root mean square (RMS). When $p_k(t)$ is large relative to the overall

RMS, this means that the k segments constructively interfere and thus all segments contain approximately the same pattern. When $p_k(t)$ is maximal (i.e., equal to the RMS of the input signal), we refer to the signal as being *periodic*, since each of the k segments must be exact copies of one another. Lower values of $p_k(t)$ imply some amount of deconstructive interference within the superposition, and thus we refer to such signals as being *aperiodic*.

We now present a method that can efficiently compute $p_k(t)$ with a SNN, by leveraging the computational properties of the DN. This has applications as a general low-power signal processing tool, and for modelling how nontrivially-repeating patterns could be detected by neural mechanisms in a robust and flexible manner.

Our goal is to recast equation 6.16 to be in terms of the DN's state-vector, $\mathbf{x}(t) \in \mathbb{R}^q$. We claim that we may compute $p_k(t)$ by taking the L2-norm of the state-vector after a linear transformation:

$$p_k(t) \mathrel{\underset{\sim}{\propto}} \|P_k \mathbf{x}(t)\| \tag{6.17}$$

$$P_k = \frac{Z_k}{k} \sum_{i=0}^{k-1} e^{Ai\gamma}, \tag{6.18}$$

where A is the recurrent state-space matrix from the balanced realization of equation 6.4,[15] and $Z_k \in \mathbb{R}^{q \times q}$ is a "zooming matrix" computed by:

$$Z_k = B(0,1)^+ B(1 - 1/k, 1), \tag{6.19}$$

where $(\cdot)^+$ denotes the Moore-Penrose pseudo-inverse, $B(\cdot,\cdot) \in \mathbb{R}^{m \times q}$ is a matrix corresponding to a slice of the realized basis functions (equation 6.12), sampled along m points, as in:

$$B(a,b) = \begin{pmatrix} \mathcal{P}_{q-1,q}(a) & \cdots & \mathcal{P}_{0,q}(a) \\ \vdots & \vdots & \vdots \\ \mathcal{P}_{q-1,q}(b) & \cdots & \mathcal{P}_{0,q}(b) \end{pmatrix} T. \tag{6.20}$$

and T is the similarity transform for the balanced realization. A visualization of Z_k is provided in Figure 6.16, using a diagonal transformation to reveal its checkerboard structure. Specifically, setting $k = 5$, $q = 20$, and with T taken by a diagonal realization that normalizes the system's Hankel singular values, and the Moore-Penrose pseudo-inverse taken while truncating the singular values below a cutoff ratio of 10%.

[15]The proportionality $\propto$ in equation 6.17 refers to a constant scaling in the balanced realization, while the approximation $\sim$ comes from the use of Padé approximants to render the system finite-dimensional, and from assuming the DN mapping is unitary. See text for details.

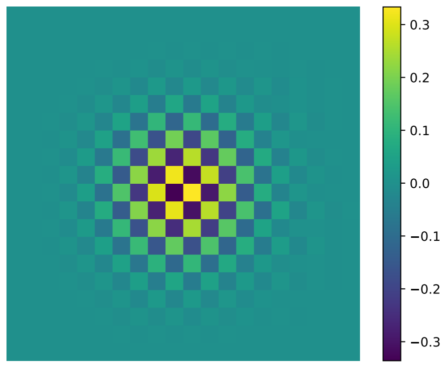


Figure 6.16: Visualizing the "zooming matrix", $Z_5 \in \mathbb{R}^{20\times 20}$ (see equation 6.19). This matrix linearly transforms the state-space of the Delay Network (DN) to "zoom in" on the oldest segment of input history partitioned into $k = 5$ segments.

We sketch a constructive proof of this claim. Since the buffered window of input history is a linear transformation of the DN's state-vector, we start by constructing a linear transformation of $\mathbf{x}(t)$ that contains the average of all segments. The matrix $e^{Ai\gamma}$ provides a linear transformation that emulates the effect of holding the input signal at zero for $i\gamma$ more seconds – a mathematical *fast-forwarding* of the system assuming no input. This can be seen by solving the differential equation $\dot{\mathbf{x}} = A\mathbf{x}$ with the initial condition $\mathbf{x}_0 = \mathbf{x}(t)$. Thus, $e^{Ai\gamma}\mathbf{x}(t)$ provides the state-vector that corresponds to some input signal where the i^{th} segment (ordered by seniority) has been shifted back in time to become the oldest segment. Note that $e^0 = I$, and so the oldest segment ($i = 0$) is not shifted. More generally, $k^{-1}\left(I + e^{A\gamma} + \ldots + e^{A(k-1)\gamma}\right)$ linearly transforms the state-vector to correspond to an input signal where the oldest segment is the average of all segments.

Next, we must "zoom in" on the oldest segment. This zooming operation can be accomplished by yet another linear transformation. In particular, Z_k is a linear transformation that takes us from a state-vector representing the entire interval of length θ to a state-vector representing the oldest sub-interval of length γ. This is derived by observing that $B(1-1/k, 1)$ maps $\mathbf{x}(t)$ onto the window of input history across the sub-interval $[t-\theta, t-\theta+\gamma]$—sampled along m points—by construction. Then $B(0,1)^+$ reinterprets this sampled signal as a state-vector in the context of the full interval $[t-\theta, t]$. To determine these matrices numerically, m should be sufficiently large so as to approximate the limit as $m \to \infty$ within some tolerance.[16]

This same technique works without loss of generality to zoom in on any slice of the window,

[16]We found $m = 1{,}000$ to be sufficient for our experiments.

and reinterpret that segment of input as a new state-vector. This is a specific instantiation of the more general observation that the relationship between the state-vector and the input history is linear, and so any linear operation across the window (e.g., slicing, shifting, convolution, integral transforms, etc.) must also be linear with respect to the state-vector. Moreover, by the scale-invariance of the delay network (see section 6.1.5), all of the matrices (e.g., Z_k, $B(\cdot, \cdot)$, P_k) are independent of θ. That is, these matrices only need to be precomputed once for some choice of k and q, and applied to any θ.

The last fact that we need is that the balanced realization is approximately unitary, in the sense that $\|\mathbf{x}(t)\|$ is proportional to the RMS of its corresponding window of history. This conveniently implies that it suffices to take the L2-norm of the resulting state-vector, as in equation 6.17, in order to obtain a proportional estimate of the RMS, $p_k(t)$, from equation 6.16. This completes our proof sketch. To summarize, since the superimposed-averaging computation is a linear transformation of the input window, it can be recast as a linear transformation of the state to provide a new state-vector corresponding to the averaged segment. And since the L2-norm of a balanced state reflects the RMS of its corresponding input signal, the former behaves as a proxy for the latter.

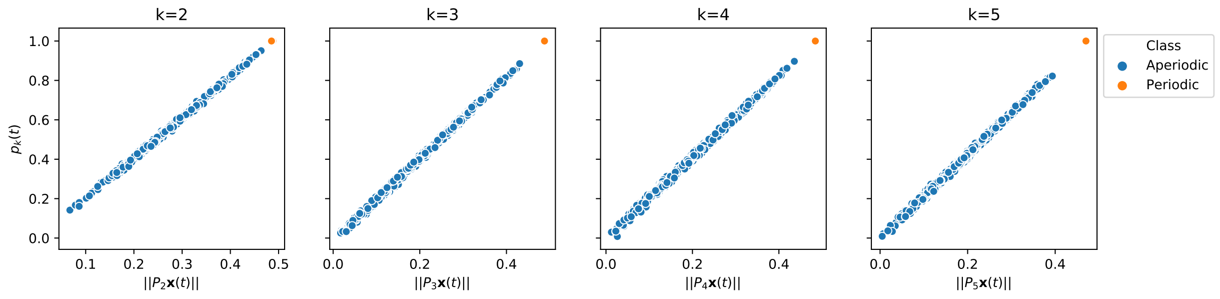


Figure 6.17: The Delay Network (DN) detecting repeating patterns over time, tested against 1,000 randomly generated input signals. The x-axis is the computed by the DN. The y-axis is the true measure of the input signal's k-periodicity—how closely it resembles some segment repeated k times—$p_k(t)$ (see equation 6.16), for two classes of input signals (500 periodic, and 500 aperiodic), for $k \in \{2, 3, 4, 5\}$. The DN perfectly separates the two classes in all cases.

We validate the above derivation by fixing $\theta = 0.24$ s, $q = 20$, and then sampling white noise signals band-limited at 22 Hz (such that the error in the Padé approximants, equation 6.14, is bounded above by 5%) with RMS=1 from the two possible classes: periodic and aperiodic. The periodic signals are generated by repeating a segment of length γ, k times, while the aperiodic signals are random across the interval of length θ. In each case, the system of equations 6.4 are balanced and simulated using zero-order hold (ZOH) with a time-step of $\mathrm{dt} = 1$ ms, to obtain the value of $\mathbf{x}(t)$ at $t = \theta$. The simulated value of $\|P_k(t)\|$ (equation 6.18) is then compared to the ideal value of $p_k(t)$ (equation 6.16). The results in Figure 6.17 empirically validate the claim of equation 6.17 for $k \in \{2,3,4,5\}$, by demonstrating that $\|P_k\mathbf{x}(t)\|$ proportionally-approximates the true k-periodicity.

Crucially, nothing has changed about the DN in order to support this form of temporal pattern recognition. The computation is merely a fixed linear transformation of the state, which is already being computed by the DN, followed by a nonlinear L2-norm operation. Thus, in order to detect temporal patterns, computations such as measures of k-periodicity are realized as fixed linear transformations between the DN and additional pools of neural nonlinearities via Principles 1 and 2 of the NEF.

6.2.5 Deep delay networks

We now consider an architecture that stacks multiple DNs on top of one another, to form a *Deep Delay Network* (DDN) chaining multiple delays together. This is the same as considering a single "column" from the architecture explored in section 6.2.3, although here we analyze it theoretically. For simplicity, we consider the case where each delay has the same length, γ, and each layer has the same dimensionality, q. Thus, k layers result in an overall delay of length $\theta = k\gamma$, and represent kq dimensions in total.

The state-vector of the i^{th} layer, is denoted $\mathbf{x}_i(t)$, where $i = 0$ corresponds to the deepest layer (i.e., the last delay), and $i = k-1$ corresponds to the shallowest layer (i.e., the first delay). We note that such an architecture realizes an alternative solution to computing the k-periodicity in section 6.2.4; in particular, since $\mathbf{x}_i(t)$ corresponds to the i^{th} segment of input history, we have:

$$P_k\mathbf{x}(t) = \frac{1}{k}\sum_{i=0}^{k-1} \mathbf{x}_i(t),$$

where P_k is the matrix defined by equation 6.18, and $\mathbf{x}(t)$ is the state-vector for an overall delay of length θ.

We now determine the error in the overall delay by extending our analysis from section 6.1.5. Since each layer implements the transfer function $[q-1/q]e^{-\gamma s}$ (equation 6.3), by the convolution theorem, the overall filter is the product of each transfer function. Therefore, the error is characterized by the filter:

$$E_{q,k}(\theta s) = \left([q-1/q]e^{-\gamma s}\right)^k - e^{-\theta s}. \tag{6.21}$$

When $k = 1$, this reduces to equation 6.14.

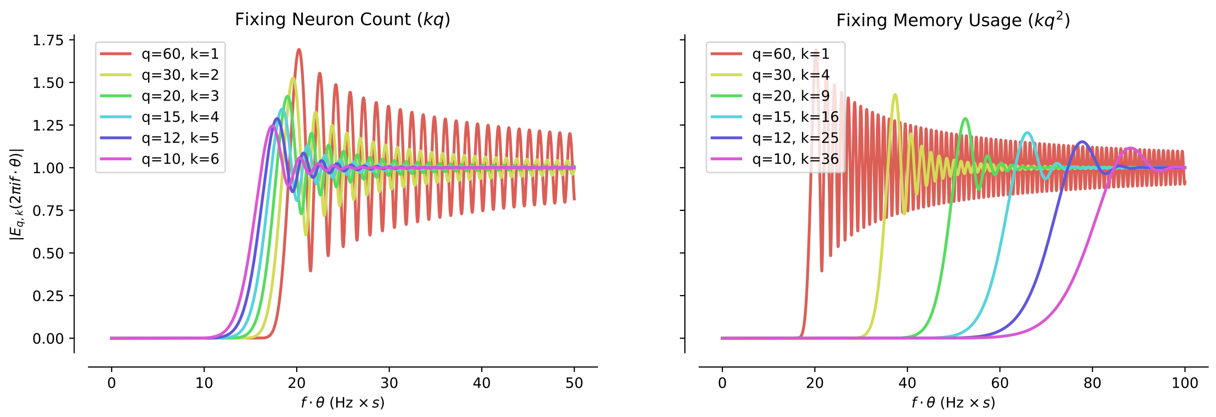


Figure 6.18: Visualizing the error in the Deep Delay Network (DDN) of width q and depth k (see equation 6.21) for two different resource-cost functions. (Left) Fixing the number of dimensions, kq, to be constant. Smaller k scales further in this case. (Right) Fixing the density of connections, kq^2, to be constant. Smaller q is more accurate in this case.

To gain insights into potential trade-offs between k and q, we require additional constraints to keep the comparison meaningful. If our main constraint is the number of neurons, then resource usage scales as $\mathcal{O}(kq)$ given a constant level of accuracy per dimension. However, if our main constraint is the number of multiply-adds and memory usage (i.e., connection density), then these scale as $\mathcal{O}(kq^2)$ assuming the use of factorized connection-weight matrices (as in the standard NEF formulation, and as realized on both SpiNNaker and Braindrop, but not Loihi) and assuming the use of balanced state-space realizations. $\mathcal{O}(kq^2)$ is also the number of internal cell parameters that need to be learned in the architecture of section 6.2.3, since these are shared across all columns. In both cases, we evaluate $E_{q,k}(\theta s)$ while varying (q, k) in such a way that keeps the resource-cost fixed (see Figure 6.18).

Depending on which resource-cost function is considered, we obtain very different trade-offs for the amount of error at some desired operating point θs (see Figure 6.8). In the former case of minimizing neural resources, we should set $k = 1$ and minimize q such that θs falls within the radius of convergence. This should come as no surprise, as the DN has been derived to optimally approximate the delay line, and so there is no benefit to adding additional layers if we are free to scale q. However, for the latter case of minimizing connectivity or the number of trainable parameters, then we should primarily minimize q and maximize k. In other words, deeper delay structures provide a considerable payoff when the cost is $\mathcal{O}(kq^2)$.

Furthermore, note that in Figure 6.18, the error oscillates with dampened amplitude beyond the radius of convergence for larger k. This can be seen from equation 6.21 by noticing that

$[q - 1/q]e^{-\gamma s}$ is a complex number with magnitude less than 1, that is then exponentiated to the power of k. By the same triangle-inequality argument in section 6.1.5, this effectively regularizes the error k times towards 1 outside the radius of convergence.

The final consideration that should be made in picking (q, k) is in determining the ideal nonlinear support for any function(s) to be computed across the window of history. Since each dimension is encoded by a heterogeneous pool of neural nonlinearities, this supports the decoding of nonlinear functions with respect to the coefficient on the corresponding basis function (equation 6.12) via Principles 1 and 2. Deeper networks effectively partition the basis functions into individual segments of input history, which enhances the complexity of the nonlinearities with respect to each segment, while limiting nonlinear interactions between segments. All of this should be systematically taken into account when choosing the state-space realization, the delay lengths of each layer, the dimensionality of each layer, and the number of layers.

6.2.6 Acausal deconvolution

In general, if one takes a communication channel, $f(u) = u$, constructed using normal NEF methods, and stacks it k times, then the i^{th} layer will represent $\mathcal{L}^{-1}\left\{H(s)^k U(s)\right\}(t)$, that is, the input $u(t)$ convolved k times with $h(t)$. This is demonstrated in Figure 6.19 (Top), which encodes a 10 Hz band-limited white noise signal through 8 layers of 2,500 spiking LIF neurons ($\tau = 100$ ms). As we see, deeper layers become progressively more lowpass-filtered in time. This has the often[17] undesirable effect of losing information within the frequency content of the input. This phenomenon contributes to the misconception that the NEF does not support high-speed transmission of information through networks, as discussed in section 3.1.5.

To solve this problem, we are free to scale τ arbitrarily small, so long as n is scaled as $\mathcal{O}\left(\tau^{-2}\right)$ (see Theorem 3.2.1) to maintain the same level of feed-forward precision. Alternatively, if τ is fixed, then one can use Principle 3 to implement a lowpass filter $(\theta s + 1)^{-1}$ with arbitrarily small θ, which likewise requires $\mathcal{O}\left(\theta^{-2}\right)$ neurons (see Table 3.1). Our solution can be viewed as a generalization of the latter.

A natural solution falls out of the DN, given by equation 6.11: the current value of $u(t)$ is represented by the population that encodes $\mathbf{x}(t)$. It is not obvious that this should be the case, as $u(t)$ has been filtered by the synapse model (e.g., a lowpass filter) to produce a filtered version (e.g., phase-shifted) of its input. Nevertheless, the state-vector is reconstructing an unfiltered

[17] Goldman (2009) has shown that repeated lowpass filtering can be usefully exploited to implement an integrator, by summing across all of the filters.

version of the window of input history, which includes the current moment in time. Such a reconstruction is also known as a deconvolution operation (i.e., the inverse convolution), and is an acausal operation in general. That is, to perform deconvolution in general for arbitrary inputs, one requires future knowledge of the input. The same applies to constructing Taylor series approximants at the current moment in time.

The low-frequency approximation of the DN essentially models the statistics of the input, and provides a robust estimate of the current $u(t)$ from the spiking activity of the population. As can be seen in Figure 6.3 at $\theta' = 0$, the transformation to do so (ignoring alternative state-space realizations) is simply a summation across the state:

$$u(t) \approx \sum_{i=0}^{q-1} x_i(t). \tag{6.22}$$

We use this fact in Figure 6.19 (Bottom) to instantaneously propagate the input through 8 layers, using the same neurons and synapses as in (Top). The difference between these two simulations is that the recurrence, local to each layer, effectively undoes the filtering by using its internal model of the input's history. This demonstrates the utility in including recurrence at each layer, not only to support dynamical computations, but to maintain the frequency content of the input signal while facilitating high-speed computation through deep neural structures.

6.2.7 Higher-order synapses

This section has been adapted from Voelker and Eliasmith (2018), and applies the extensions from section 5.1 to the case of the delay network from section 6.1.

We begin by making the practical point that it is crucial to account for the effect of the simulation time-step in digital simulations, if the time-step is not sufficiently small relative to the time scale of the desired network-level dynamics. To demonstrate this, we simulate a 27-dimensional delay network using 1,000 spiking LIF neurons, implementing a 0.1 s delay of 50 Hz band-limited white noise. We vary the simulation time-step (`dt`) from 0.1 ms to 2 ms. The accuracy of our extension does not depend on `dt` (see Figure 6.20 (Left)). When `dt` = 1 ms (the default in Nengo), the standard Principle 3 mapping (equation 5.8) obtains a NRMSE of 1.425 (43% worse than random chance), versus 0.387 for the discrete lowpass mapping which accounts for `dt` (equation 5.9)—a 73% reduction in error. As `dt` approaches 0 the two methods become equivalent.

More to the point, we can analyze the delay network's frequency response when using a continuous lowpass synapse and an axonal delay of λ (equation 5.10) instead of the canonical

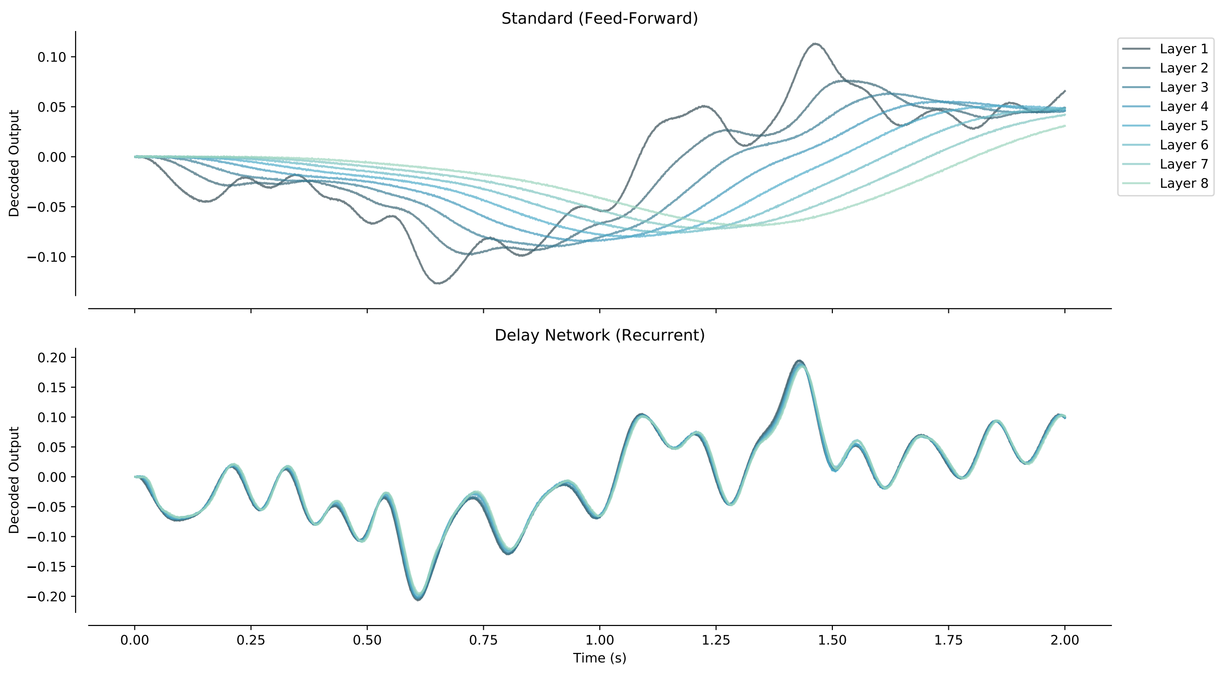


Figure 6.19: Demonstrating the ability of the Delay Network (DN) to approximate a synaptic deconvolution operation, realizing a pure communication channel. The test input $u(t)$ is a white noise signal band-limited at 10 Hz. (Top) A standard feed-forward communication channel, implemented by decoding the identity function at each layer. (Bottom) A recurrent communication channel, implemented by a Deep Delay Network (DDN; section 6.2.5) approximating $u(t)$ at each layer. Each layer is a DN with $\theta = 50$ ms and $q = 6$. In either case, there are 8 layers of 2,500 spiking LIF neurons, with each layer communicating spikes to the next through a lowpass synapse with time-constant $\tau = 100$ ms.

lowpass (equation 4.11) as the dynamical primitive. This provides a direct measure of the possible improvement gains when using the extension. Figure 6.20 (Right) compares the use of Principle 3 (which accounts for τ but ignores λ), to our extension (which fully accounts for both; see equation 5.12) when $\lambda = \tau$. The figure reveals that increasing the dimensionality improves the accuracy of our extension, while magnifying the error from Principle 3. In the worst case, the Principle 3 mapping has an absolute error of nearly 10^{15}. In practice, saturation from the neuron model bounds this error by the maximum firing rates. Regardless, it is clearly crucial to account for axonal transmission delays to accurately characterize the network-level dynamics.

In neuromorphic hardware such as Loihi, delays are configurable on a per-axon or per-

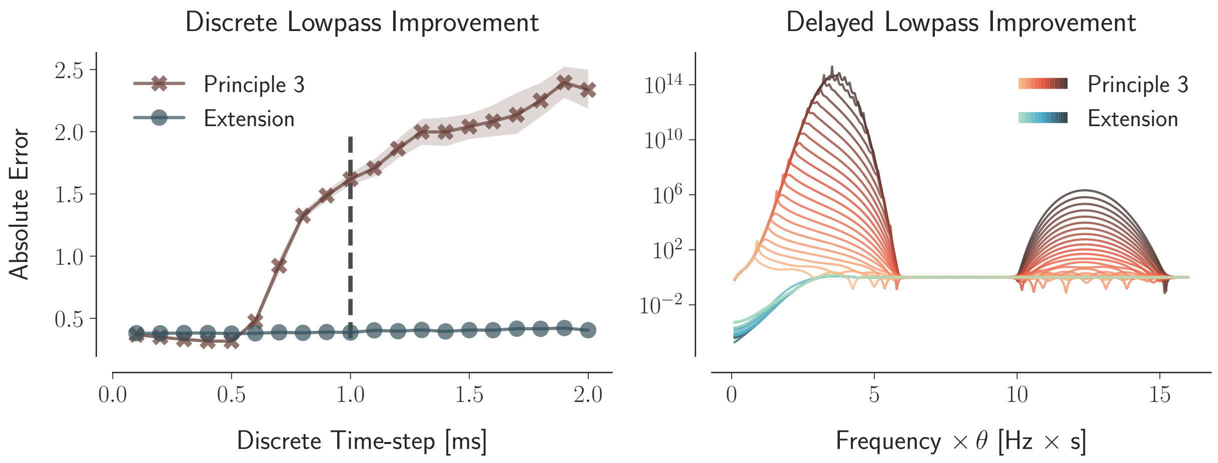


Figure 6.20: Comparing standard Principle 3 to our NEF extensions. (Left) Error from mapping a 27-dimensional 0.1 s delay onto 1,000 spiking LIF neurons, while varying the simulation time-step (dt). The input to the network is white noise with a cutoff frequency of 50 Hz. Unlike our extension, the standard form of Principle 3 does not account for dt. A dashed vertical line indicates the default time-step in Nengo. Error bars indicate a 95% confidence interval bootstrapped across 25 trials. (Right) Mapping the delay system onto a delayed continuous lowpass synapse (with parameters $\tau\theta^{-1} = 0.1$ and $\lambda\tau^{-1} = 1$). The order of the delay system (q) is varied from 6 (lightest) to 27 (darkest). Each line evaluates the error in the frequency response, $\left|e^{-\theta s} - F^H(H(s)^{-1})\right|$, where F^H is determined by mapping the delay of order q onto equation 5.10 using one of the two following methods. The method of our extension—which accounts for the axonal transmission delay—has monotonically increasing error that stabilizes at 1 (i.e., the high frequencies are filtered). The standard Principle 3—which accounts for τ but ignores λ—alternates between phases of instability and stability as the frequency is increased. Reproduced from (Voelker and Eliasmith, 2018, Figure 8).

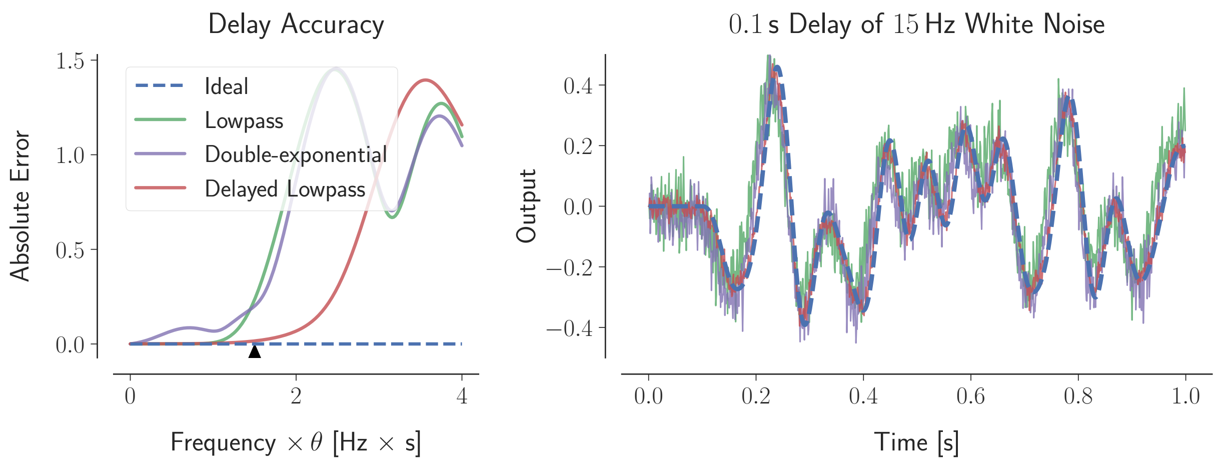


Figure 6.21: The pure delay mapped onto spiking networks with various synapse models (with parameters $q = 6$, $\tau\theta^{-1} = 0.1$, $\lambda\tau^{-1} = 1$, $\tau_1 = \tau$, and $\tau_1\tau_2^{-1} = 5$). (Left) Error of each mapping in the frequency domain. This subfigure is scale-invariant with θ. (Right) Example simulation when $\theta = 0.1$ s and the input signal is white noise with a cutoff frequency of 15 Hz, corresponding to the triangle (over 1.5) from the left subfigure. We use a time-step of 0.01 ms (10 μs) and 2,000 spiking LIF neurons. Reproduced from (Voelker and Eliasmith, 2018, Figure 9).

synapse basis (Davies et al., 2018). We would not only like to account for the existence of such delays, but leverage them as dynamical primitives for higher-order computation. Thus, to more broadly validate our NEF extensions, we map the delay system onto: (1) a continuous lowpass synapse; (2) a delayed continuous lowpass synapse; and (3) a continuous double-exponential synapse (see section 5.1.1). We apply each extension to construct delay networks of 2,000 spiking LIF neurons. To compare the accuracy of each mapping, we make the time-step sufficiently small ($\mathrm{dt} = 10\,\mu$s) to emulate a continuous-time setting. We use the Padé approximants of order [5/6] for both equations 5.12 and 6.3. For the delayed lowpass, we again fix $\tau\theta^{-1} = 0.1$ and $\lambda\tau^{-1} = 1$. For the double-exponential, we fix $\tau_1 = \tau$ and $\tau_1\tau_2^{-1} = 5$. Expressing these parameters as dimensionless constants keeps our results scale-invariant with θ.

Figure 6.21 reveals that axonal delays may be effectively "amplified" 10-fold while reducing the NRMSE by 71% compared to the lowpass (see Figure 6.21 (Right); NRMSE for lowpass=0.702, delayed lowpass=0.205, and double-exponential=0.541). The double-exponential synapse outperforms the lowpass, despite the additional poles introduced by the ZOH assumption in equation 5.22 that we analyze in section 5.1.2 . This is because the double-exponential filters the spike-noise twice. Likewise, by exploiting an axonal delay, the same level of performance (e.g., 5% error) may be achieved at approximately 1.5 times higher frequencies, or equivalently for 1.5 times longer network delays, when compared to the lowpass synapse (see Figure 6.21 (Left)). In summary, accounting for higher-order synaptic properties allows us to harness the axonal transmission delay to more accurately approximate network-level delays in spiking dynamical networks.

Together, these results demonstrate that our extensions can significantly improve the accuracy of high-level network dynamics. Having demonstrated this for delays, in particular, suggests that the extension is useful for a wide variety of biologically relevant networks, such as those discussed in the following section.

6.2.8 Time cell data

This section has been reproduced from Voelker and Eliasmith (2018).

We now describe a connection between the delay network from this chapter and recent neural evidence regarding time cells. Time cells were initially discovered in the hippocampus and proposed as temporal analogs of the more familiar place cells (Eichenbaum, 2014). Similar patterns of neural activity have since been found throughout striatum (Mello et al., 2015) and cortex (Luczak et al., 2015), and have been extensively studied in the rodent mPFC (Kim et al., 2013; Tiganj et al., 2016).

Interestingly, we find that our delay network produces qualitatively similar neural responses to those observed in time cells. This is shown in Figure 6.22, by comparing neural recordings from mPFC (Tiganj et al., 2016, Figure 4 C,D) to the spiking activity from a network implementing a delay of the same length used in the original experiments. Specifically, in this network, a random population of 300 spiking LIF neurons maps a 4.784 s[18] delay onto an alpha synapse ($\tau = 0.1$ s) using our extension. The order of the approximation is $q = 6$ (see equation 6.4), and the input signal is a rectangular pulse beginning at $t = -1$ s and ending at $t = 0$ s (height = 1.5). The simulation is started at $t = -1$ s and stopped at $t = 5$ s.

We also note a qualitative fit between the length-curve for $q = 6$ in Figure 6.4 and the peak response-times in Figure 6.22. Specifically, Figure 6.4 (Bottom) models the non-uniform distribution of the peak response-time of the cells as the length of the trajectory of $\mathbf{x}(t)$ through time. Implicit to this model are the simplifying assumptions that encoders are uniformly distributed, and that the L2-norm of the state-vector remains constant throughout the delay period. Nevertheless, this model produces a qualitatively similar curve when $q = 6$ to both peak response-times from Figure 6.22 (Right) (see overlay).

[18]This value comes from Tiganj et al. (2016).

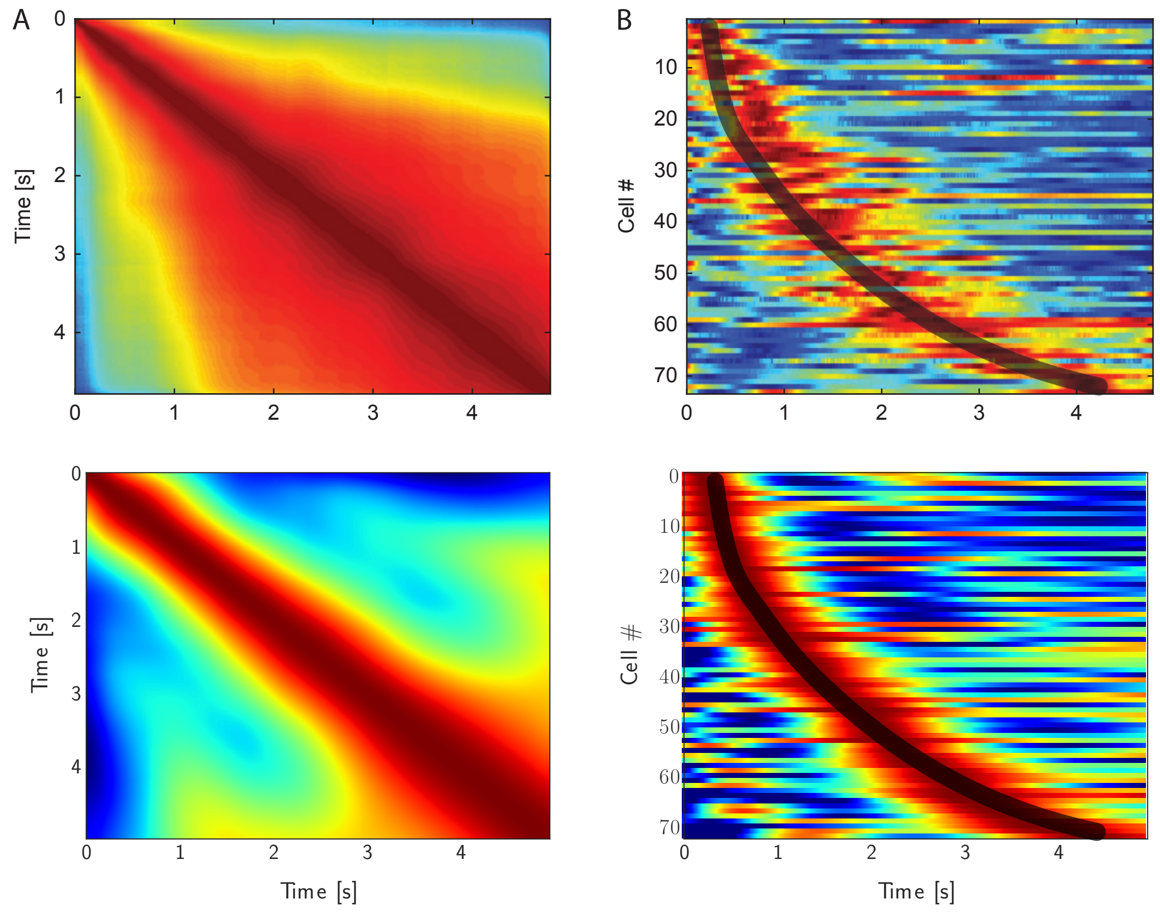


Figure 6.22: Comparison of time cells to a Delay Network. (Top) Spiking activity from the rodent mPFC (reproduced from Tiganj et al., 2016, Figures 4C and 4D, by permission of Oxford University Press). Neural recordings were taken during a maze task involving a delay period of 4.784 s. (Bottom) Delay network implemented using the NEF (see text for details). 73 time cells are selected by uniformly sampling encoders from the surface of the hypersphere. (A) Cosine similarity between the activity vectors for every pair of time-points. The diagonal is normalized to the warmest colour. The similarity spreads out over time. (B) Neural activity sorted by the time to peak activation. Each row is normalized between 0 (cold) and 1 (warm). We overlay the curve from Figure 6.4 (Bottom) ($q = 6$) to model the peak-response times. Reproduced from Voelker and Eliasmith (2018, Figure 10).

More quantitatively, we performed the same analysis on our simulated neural activity as Tiganj et al. (2016) performed on the biological data to capture the relationship between the peak and width of each time cell. Specifically, we fit the spiking activity of each neuron with a Gaussian to model the peak time (μ_t) and the standard deviation (σ_t) of each cell's "time field".[19] This fit was repeated for each of the 250 simulated spiking LIF neurons that remained after selecting only those that had at least 90% of their spikes occur within the delay interval. The correlation between μ_t and σ_t had a Pearson's coefficient of $R = 0.68$ ($\rho < 10^{-34}$), compared to $R = 0.52$ ($\rho < 10^{-5}$) for the biological time cells. An ordinary linear regression model linking μ_t (independent variable) with σ_t (dependent variable) resulted in an intercept of 0.27 ± 0.06 (standard error) and a slope of 0.40 ± 0.03 for our simulated data, compared to 0.27 ± 0.07 and 0.18 ± 0.04 respectively for the time cell data. We note that we used the same bin size of 1 ms, modeled the same delay length, and did not perform any parameter fitting beyond the informal choices of 90% cutoff, dimensionality ($q = 6$), area of the input signal (1.5), and synaptic time-constant ($\tau = 0.1$ s).

Neural mechanisms previously proposed to account for time cell responses have either been speculative (Tiganj et al., 2016), or rely on the precision of gradually changing firing rates from a bank of arbitrarily long, ideally spaced, lowpass filters (Shankar and Howard, 2012; Howard et al., 2014; Tiganj et al., 2015, 2017, 2018).[20] It is unclear if such methods can be implemented accurately and scalably using heterogeneous spiking neurons. We suspect that robust implementation is unlikely given the high precision typically relied upon in these abstract models.

In contrast, our proposed spiking model has its network-level dynamics derived from first principles to optimally retain information throughout the delay interval, without relying on a particular synapse model or bank of filters. All of the neurons recurrently work together in a low-dimensional vector space to make efficient use of neural resources. By using the methods of the NEF, this solution is inherently robust to spiking noise and other sources of uncertainty. Furthermore, our explanation accounts for the nonlinear distribution of peak firing times as well as its linear correlation with the spread of time fields.

The observation of time cells across many cortical and subcortical areas suggests that the same neural mechanisms may be used in many circuits throughout the brain. As a result, the neural activity implicated in a variety of delay tasks may be the result of many networks optimizing a similar problem to that of delaying low-frequency signals recurrently along a low-dimensional manifold. Such networks would thus be participating in the temporal coding of a stimulus, by representing its history across a delay interval.

[19] We set $a_1 = P = S = 0$ in equation 1 from Tiganj et al. (2016), since we have no external variables to control.

[20] One may view this as a generalization of the main idea from Goldman (2009).

Chapter 7

Applications to Neuromorphic Hardware

Nengo and the NEF provide a useful toolkit for programming recurrently coupled networks of spiking neurons to carry out the computations of algorithms formulated as dynamical systems. The CPU backend is immensely useful for rapidly prototyping ideas, testing hypotheses, and experimenting with small-scale models on the order of a few thousand neurons. However, for much larger models such as Spaun (Eliasmith et al., 2012), its 2.5 million neurons require about 2.5 hours of compute time per 1 second of simulation time, that is ≈ 9,000 times slower than real-time (Stewart and Eliasmith, 2014; Mundy, 2016). As a result, one must turn to specialized hardware such as GPUs, FPGAs, or neuromorphic architectures, in order to carry out much larger simulations of spiking neural networks (section 2.3).

For example, in our work in collaboration with Knight et al. (2016), we demonstrate that a heteroassociative memory network deployed on SpiNNaker takes a *constant* amount of time per association, while the exact same network takes a linear amount of time per association on a CPU.[1] At its largest instantiation of 100,000 neurons, the SpiNNaker simulation is more than 150 times faster than a CPU, due to the massive parallelism afforded by the architecture. This demonstration is encouraging to see, as it illustrates the potential for these methods to scale up and take full advantage of the distributed nature of spiking computation, while leveraging the power of dynamical systems to learn synaptic weights with both unsupervised (section 4.2.2) and supervised (section 4.2.3) learning rules.

However, there are very few examples of *functional* large-scale models running on neuromorphic hardware. We believe that this surprising lack of scale—in a field that was essentially created to solve a problem in scaling—is primarily due to a lack of co-designed NEF-like

[1]The number of neurons were scaled linearly in the number of associations.

frameworks for translating computations, specified in some high-level language (e.g., coupled differential equations), onto distributed networks of physically coupled devices. These frameworks must be correct, scalable, complete, robust, extensible, and finally realized on the physical hardware itself, in order to be ultimately useful (section 3.2). In the case of the NEF, this criteria has been validated by many of the extensions and analyses throughout this thesis, as well as by past work compiling networks onto Neurogrid (Dethier et al., 2011), SpiNNaker (Mundy et al., 2015; Mundy, 2016; Knight et al., 2016), TrueNorth (Fischl et al., 2018), and many other architectures (Naylor et al., 2013; Bekolay et al., 2014; Corradi et al., 2014; Wang et al., 2014; Berzish et al., 2016; Wang et al., 2017; Rasmussen, 2018; Blouw et al., 2018).

However, at time of writing, the Nengo *backends* for Braindrop (Neckar et al., 2019) and Loihi (GitHub, 2019b) are brand new—as are the chips themselves (Neckar, 2018; Davies et al., 2018)—and thus currently fall short of their promise to realize *large-scale* functional SNNs in neuromorphic hardware. For the case of Braindrop, its shortcoming is mainly by design: the chip is 0.65 mm^2 and implements 4,096 neurons. It is a proof-of-concept prototype for research, that can in principle be tiled to scale to much larger models in the future and tailored towards the requirements of some application space. For the case of Loihi, due to a combination of a lack of hardware support for factorized weight matrices, and limitations on connectivity and memory, the maximum recurrently connected pool size is 342 neurons.[2] And with current software workarounds, the feed-forward precision stops scaling in Nengo-Loihi after about 400 neurons. Nevertheless, this is the first and only software abstraction to currently make use of Loihi (Blouw et al., 2018) outside of Intel (Lin et al., 2018b), and it is under active development. We expect this to get significantly better with time.

Nevertheless, these two architectures—Braindrop and Loihi—represent significant milestones in the evolution of neuromorphic hardware. The first, Braindrop, consumes energy roughly equivalent to 381 fJ per synaptic operation, for typical network configurations, or about 10–20 times more than the human brain (see section 2.3). The second, Loihi, consumes about 24 pJ per synaptic operation, or about 50–100 times more than Braindrop, but offers determinism and an unparalleled degree of flexibility given its power budget. At the same time, these two neuromorphic hardware architectures are about as different from one another as two could be in the space of neuromorphic computing, with each posing very different sets of challenges when it comes to compiling NEF networks.

The goal of this chapter is to demonstrate that the fundamentals for systematically programming neuromorphic hardware are in place. With the theoretical guarantees provided by chapter 3, the methods for programming dynamical systems in chapter 4, the extensions of chapter 5, and the functional capabilities of dynamical systems such as those explored in

[2]Determined empirically using the `nengo-loihi==0.5.0` emulator.

chapter 6 – we are confident that in principle these methods can scale to Spaun's 6.6 million neurons (Choo, 2018) and beyond. Thus, we focus on demonstrating the principles of the NEF, and a few of its extensions, applied to two fundamental—but admittedly small-scale—dynamical systems. These dynamical systems are described in the same high-level language of Nengo, but mapped onto two vastly different state-of-the-art neuromorphic architectures: Braindrop and Loihi.

7.1 Integrator

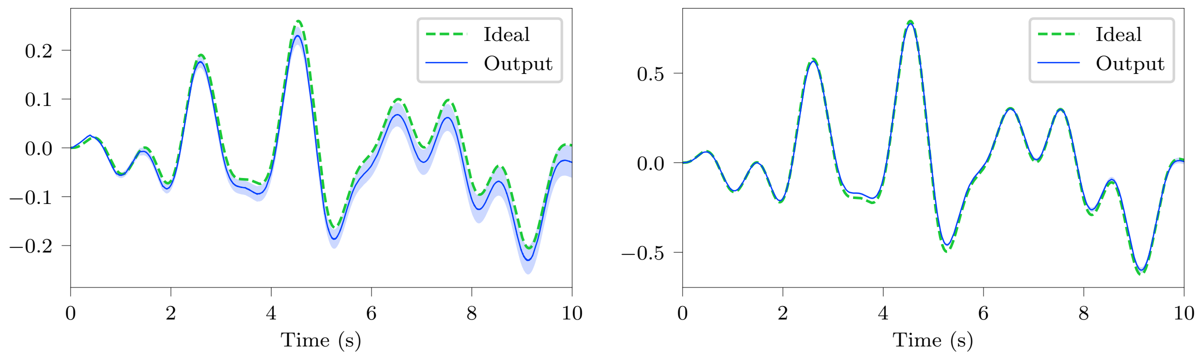
The integrator, $\theta \dot{\mathbf{x}}(t) = \mathbf{u}(t)$, is a dynamical system used extensively by Spaun (Eliasmith et al., 2012) and many other NEF and SPA models (Singh and Eliasmith, 2004; Trujillo, 2014; Rasmussen et al., 2017, to name a few) for cognitive tasks involving working memory (also see section 4.2 for a brief review). This system persists information about the history of an input signal, starting from t_0 and extended indefinitely throughout time, as characterized by the solution to its differential equation,

$$\mathbf{x}(t) = \mathbf{x}(t_0) + \frac{1}{\theta} \int_{t'=t_0}^{t} \mathbf{u}(t')\, dt'.$$

The parameter θ is a system-level time-constant that controls how quickly the memory integrates new information.[3] For example, beginning from an initial state of $\mathbf{x}(t_0) = \mathbf{0}$, if we hold the input constant at $\mathbf{u}(t) = \mathbf{v}$ for θ seconds ($t_0 < t \le t_0 + \theta$), and then clamp the input to $\mathbf{u}(t) = \mathbf{0}$ thereafter ($t > t_0 + \theta$), then $\mathbf{x}(t) = \mathbf{v}$ will "store" the vector $\mathbf{v}$ indefinitely.

More generally, any finite set of nonlinear differential equations can be described as the integration of an input vector that is some nonlinear function of the state-vector—augmented to also include $\mathbf{u}(t)$—as in $\theta \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t))$. The nonlinearity $\mathbf{f}(\cdot)$ may then be supported by a pool of neurons encoding the state-vector, as explained in section 2.2. Indeed, this is the essential observation of Principle 3; dynamical systems may be reduced to integration. Thus, the integrator is a basic component used to implement sophisticated nonlinear dynamical transformations, such as those involved in adaptive motor control (DeWolf et al., 2016). We therefore use the integrator, implemented by a pool of spiking neurons using the methods of the NEF, as a benchmark for evaluating the ability of neuromorphic hardware to implement generic dynamical systems.

[3]This parameter is useful for dimensional analysis, when considering the transfer function, $\frac{\mathbf{X}(s)}{\mathbf{U}(s)} = (\theta s)^{-1}$, and in determining the precision of the system: n scales as $\mathcal{O}\left(\theta^{-2}\right)$ (see Table 3.1).



(a) Braindrop (b) Loihi

Figure 7.1: An integrator running on state-of-the-art neuromorphic hardware. The ideal solution is plotted against the mean's 95% confidence interval, bootstrapped from 200 trial simulations. (a) Nengo Braindrop implementation, reproduced from Neckar et al. (2019, Figure 15), using 1,024 neurons. (b) Nengo Loihi (v0.5.0) implementation, using 256 neurons. Loihi simulations performed by Xuan Choo from Applied Brain Research, Inc. See text for details.

In Figure 7.1, we instantiate a one-dimensional integrator ($\theta = 1\,\text{s}$) on both Braindrop (1024 neurons) and Loihi (256 neurons). In both cases, the input signal is a white noise test signal, band-limited to 1 Hz. Both the ideal and the spiking activity are filtered with a lowpass ($\tau = 200\,\text{ms}$). We report the mean output's 95% confidence intervals, bootstrapped from 200 trials lasting 10 seconds each. We run a large number of trials with random configurations of the network on the same input, in order to assess overall robustness, and to reveal any systemic biases in the error or drift.

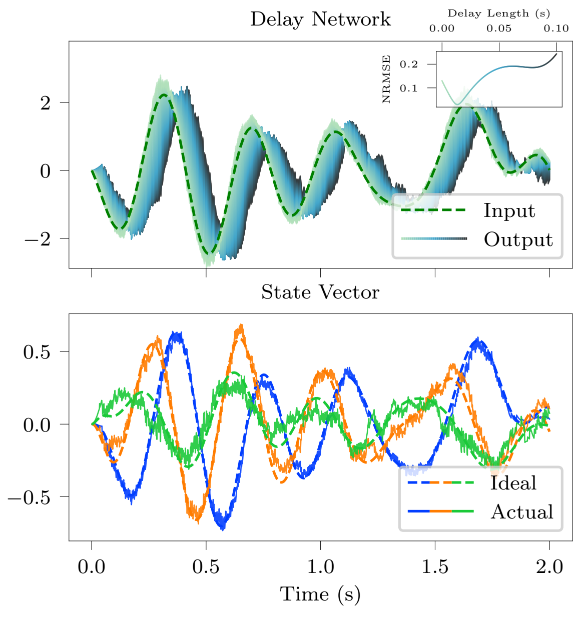
On Braindrop (see Figure 7.1 (a), reproduced from Neckar et al. (2019, Figure 15)), we compensate for the distribution of synaptic time-constants, arising from transistor mismatch in the analog circuitry, using the methods of Voelker et al. (2017a), detailed in section 5.1.3 and 5.1.4. We do so by first measuring the time-constants, and then targeting them individually with the specific transform given by our extensions. The chip is configured to maximize the synaptic time-constants; the empirically measured mean is 179.3 ms, with a standard deviation of 53.8 ms. To help account for temperature-induced variability—manifesting as slowly-changing shifts to the effective tuning curves (Kauderer-Abrams et al., 2017)—we also retrain the weights every 10 simulations directly from the spike data, collected using a separate training signal. The rest of the mapping is handled by the Nengo-Braindrop backend (Neckar et al., 2018, 2019).

On Loihi (see Figure 7.1 (b)), we set $\tau = 200\,\text{ms}$, and use the methods of section 5.1.1 to discretize the integrator according to the simulation time-step ($\texttt{dt} = 1\,\text{ms}$). We also use non-leaky neurons (integrate-and-fire), as this provides the best accuracy (see section 5.2.2). Weight matrices are left unfactored. Lastly we move the input synapse onto the spike-generator from host to chip, and uniformly initialize the states of the spike-generators to satisfy the uniform ISIP criteria (Theorem 3.2.1). The rest of the mapping is handled by the Nengo-Loihi backend (GitHub, 2019b).

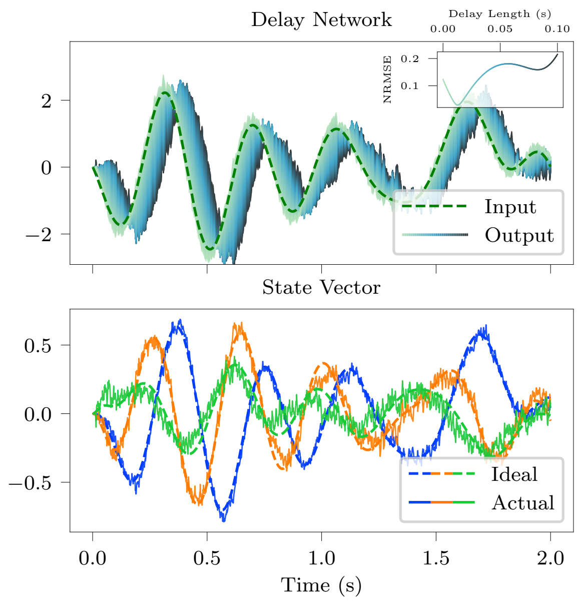
In both cases, the 95% confidence intervals include the ideal across nearly the entire 10 s simulation. This implies that the sum of any error, whether related to spiking, representation, or otherwise, has a mean value of approximately zero. We remark that Loihi gives much more consistent trial-to-trial results, due to the all-digital nature of the chip, determinism, and lack of temperature-induced variability. Advanced methods that can be used to provide temperature-invariant decodes in Braindrop have been recently developed by Reid et al. (2019), although they require external measurements of the device's temperature, and thus were not employed in this experiment.

7.2 Delay Network

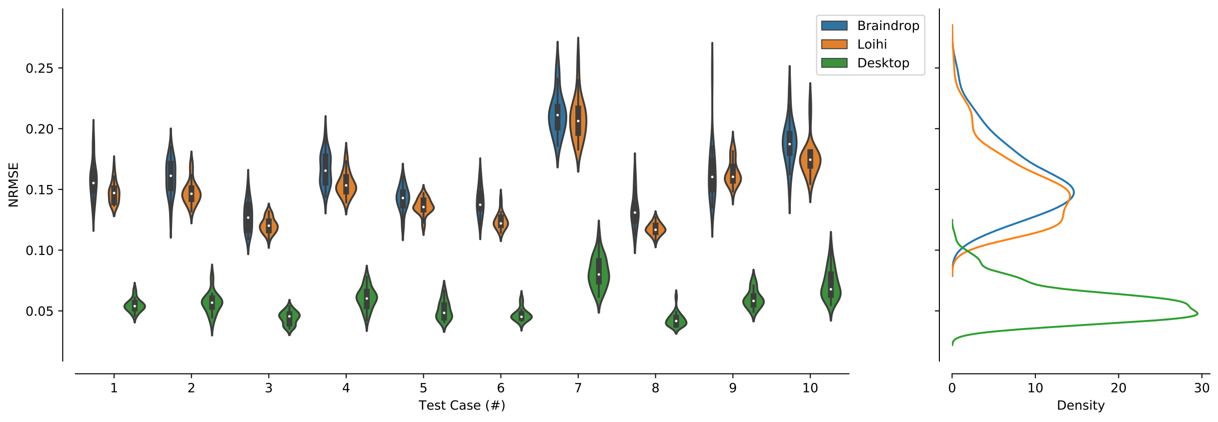
We now instantiate the same Delay Network (DN) from chapter 6 on state-of-the-art neuromorphic hardware (see Figure 7.2). This system is an entirely different kind of dynamical system that persists not the input's sum, but finite windows of the input's history. Specifically, the DN *buffers* a sliding window of the input, thus serving as a continuous-time memory, and enabling the computation of arbitrary nonlinear transformations across a window of time. This can be used as a basic building block for working memory models that must represent not only *what* has occurred, but also *when* it has occurred.



(a) Braindrop



(b) Loihi



(c) Overall Error

Figure 7.2: Delay Network (DN; $q = 3$, $\theta = 100\,\text{ms}$; chapter 6) running on state-of-the-art neuromorphic hardware. (a) Nengo Braindrop implementation, reproduced from Neckar et al. (2019, Figure 16). (b) Nengo Loihi (v0.5.0) implementation. (c) Overall error (NRMSE) for Braindrop, Loihi, and a standard desktop CPU. The simulations of (a) and (b) correspond to a randomly chosen trial from the first test case from (c). Loihi simulations performed by Xuan Choo from Applied Brain Research, Inc. See text for details.

This system is described by (repeating equations 6.5–6.12, for clarity):

$$\begin{aligned}
\theta\dot{\mathbf{x}}(t) &= A\mathbf{x}(t) + Bu(t) \\
A &= \begin{pmatrix} -v_0 & -v_0 & \cdots & -v_0 \\ v_1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ 0 & 0 & v_{d-1} & 0 \end{pmatrix} \\
B &= \begin{pmatrix} v_0 & 0 & \cdots & 0 \end{pmatrix}^\top \\
v_i &= (q+i)(q-i)(i+1)^{-1} \\
u(t-\theta') &\approx \sum_{i=0}^{q-1} \mathcal{P}_{i,q}\left(\frac{\theta'}{\theta}\right) x_{q-1-i}(t), \quad 0 \le \theta' \le \theta \\
\mathcal{P}_{i,q}(r) &= \binom{q}{i}^{-1} \sum_{j=0}^{i} \binom{q}{j}\binom{2q-1-j}{i-j}(-r)^{i-j}.
\end{aligned}$$

In addition, the state-space realization is balanced and normalized (see section 4.1), and the polynomial basis functions are rotated accordingly.

To implement the DN, three pools, each containing 128 spiking LIF neurons, are recurrently coupled to each other (and to themselves), and trained to optimally buffer a white noise test signal—band-limited to 3 Hz—across a 100 ms sliding time-window. Output spikes are filtered using a lowpass synapse with $\tau = 20$ ms, and weighted to decode both the state-vector and the window of history via $\mathcal{P}_{i,3}(\cdot)$.

For this experiment, we use identical Nengo model code for both neuromorphic backends. On Braindrop (see Figure 7.2 (a), reproduced from Neckar et al. (2019, Figure 16)), the chip is configured to use the default distribution of synaptic time-constants (mean $\tau \approx 18$ ms). For Loihi (see Figure 7.2 (b)), the recurrent time-constant is set to $\tau = 10$ ms, and weight matrices are unfactored. We also compare this to the reference Nengo simulator ($\tau = 10$ ms) running the exact same model on a conventional desktop CPU, to obtain a baseline level of performance. The overall error is evaluated across the entire delay interval $\theta' \in [0, \theta]$ using the corresponding analytical readout, $\mathcal{P}_{i,3}(\cdot)$.

Table 7.1 reports the bootstrapped 95% confidence intervals across 25 trials of 10 separate test cases. Given the inherent variability in Braindrop's analog computation, and its incredibly low power-consumption relative to both Loihi and the CPU solutions, it is surprisingly accurate given our anecdotal experience simulating noisy small-scale dynamical spiking networks. However, these results reveal an unexpected drop in precision on Loihi relative to the reference CPU solution. We attribute this to a combination of: the input's encoding into spikes, the

Method	95% Confidence Interval
Braindrop	[0.156, 0.163]
Loihi	[0.146, 0.153]
Nengo-Loihi Emulator	[0.145, 0.151]
Reference CPU	[0.055, 0.059]

Table 7.1: Performance of the Delay Network (DN) running on state-of-the-art neuromorphic hardware: Braindrop and Loihi. We also include Nengo's emulation of the Loihi hardware, and Nengo's CPU reference backend. All four simulations use the same model code and test signals.

LIF neuron model's discretization in time, quantization errors in neural states and recurrent weights, and the uniform ISIP criteria being systematically violated leading to state discrepancy (definition 3.2.1). Notably, this problem is revealed by fewer neurons per dimension, faster input frequencies, a faster system-level time-constant (θ), and faster synaptic time-constant (τ), relative to the previous experiment in section 7.1. It is difficult to explore these issues without direct access to the hardware. Future work is needed to ensure that high-bandwidth signals are being adequately and scalably encoded as per our analysis in chapter 3.

Chapter 8

Conclusions

We have discussed a number of theoretical and practical results involving the synthesis of dynamical systems, their implementation in spiking neural networks, and deployment on neuromorphic hardware. We now summarize our main contributions.

First, section 3.1 observes a number of computational "sub-principles" that follow from the adoption of the NEF's three principles. Arbitrary network topologies reduce to a single recurrently connected layer with sparse encoders, decoders with block structure isomorphic to the original graph structure, and a partitioning of time-constants. Heterogeneous dynamical primitives form the basis for neural computation, and may be expressed within a unified language to facilitate mathematical analyses that leverage the interchangeability and composability of such primitives. Chaotic strange attractors emerge from even the simplest spiking implementations of such dynamical systems. The NEF represents state-vectors by linearly projecting them onto the postsynaptic currents of neurons, independent of any considerations of what it means to use a "rate code" or a "spike-time code." Likewise, these state-vectors represent frequency content that grows linearly with the relative amount of energy required to drive the synapses of each postsynaptic neuron, for a fixed level of precision.

Second, section 3.2 addresses the suitability of the NEF as a framework for compiling SNNs onto neuromorphic hardware. Correctness is guaranteed by Theorem 3.2.1, which teases apart three sources of error in the NEF, and provides a novel proof of the NEF's precision, conditioned upon a specific criteria that characterizes the distribution of neural states. Scalability is guaranteed by the previous theorem, in conjunction with a number of prior observations made about time, space, and energy requirements that carefully consider the physical dimensions of relevant quantities (Table 3.1). Completeness is provisioned by the Turing-completeness of dynamical systems, which justifies our assertion that spiking neural networks—trained with the NEF to obey some dynamics—represent powerful models of computation. Robustness is ensured by a volume of prior work, together with our observation that the NEF is robust to white noise bounded by the diameter of its neural basin of attraction. Extensibility is demonstrated by a large number of Nengo backends supporting a variety of seemingly disparate architectures, in addition to the extensions summarized in point four below.

Third, section 4.2 provides a number of novel perspectives on various algorithms realized by dynamical systems, in contexts that do not traditionally take a dynamical systems-based approach at the state-space level. In particular, winner-take-all networks may be given their inputs sequentially, in which case the ideal solution becomes a nonlinear dynamical system – as does the case when all inputs are provided simultaneously. Unsupervised learning of encoders is a dynamical system, local to each synapse, that can be used to learn fixed-points memorizing its encoded vectors. Supervised learning of decoders is likewise a dynamical system, that can be unified with system-level dynamics, and subsequently exploited to implement higher-order transfer functions with only a single spiking neuron (see Lemma 4.2.1 and Theorem 4.2.2). Both dynamical systems are simultaneously realized in a SpiNNaker simulation to learn a heteroassociative memory in a network of 100,000 neurons, while running 150 times faster than a CPU. Lastly, we observe that a large variety of important routines in linear algebra and gradient-based optimization problems may be cast as dynamical systems that resolve the correct solution over time.

Fourth, section 5.1 exposes several theoretical extensions to the third principle of the Neural Engineering Framework, thus enabling spiking networks to leverage the computations of higher-order synapses in both continuous- and discrete-time domains. We thoroughly characterize linear dynamical systems using the transfer function representation, which allows for axonal spike-delays, such as those in Loihi, to be harnessed, while appealing to Lemma 5.1.1 to prove the most general case in Theorem 5.1.2. Nonlinear extensions are supported by Theorems 5.1.3 and 5.1.4, and culminate in an application that exploits heterogeneous mixed-signal synapses in Braindrop. We then demonstrate that heterogeneous dynamical primitives may in principle be exploited to accurately classify surface textures using a tactile robotic fingertip. And finally we extend the NEF's optimization problem to simultaneously solve for decoding weights and the optimal linear filter in the context of Principle 3.

Fifth, section 5.2 considers the challenges in extending the NEF to various neuron models, while reviewing the work done thus far. In particular, the role of adaptive neurons and biophysical neurons may be characterized as decomposing a nonlinear transfer function into increasingly-sophisticated model representations. In doing so, we observe that adaptive LIF neurons are isomorphic to an n-dimensional expansion of the dynamical state vector, and we provide a conceptual roadmap for how progress can be made to harness such models systematically within the NEF. Poisson-spiking models are compared to regular-spiking and adaptive neuron models in the context of Principle 1, which validates our proof of Theorem 3.2.1 – namely, that uniform neural states guarantee the scaling of precision, even at arbitrarily high representational frequencies. This validates our theory, and recommends that Nengo should consider the role of neural-state distributions very carefully in its abstractions.

Sixth, section 6.1 derives a solution to a difficult task for spiking neural networks: maintaining a sliding window of input history. We derive from first-principles an optimal low-frequency approximation of dimension q to a continuous-time delay, and realize this efficiently using a spiking neural network. This solution is called the Delay Network (DN). We show that the DN supports a decoding of the entire window, with weights given by the shifted Legendre polynomials up to order q. Next, the neural encoding implements a nonlinear kernel convolution of the time window, thus enabling the population of neurons within the delay network to compute arbitrary nonlinear functions across the input's q-degree projection over time. This leads into a discussion on the scalability of the network, in particular highlighting the importance of the relationship between each physical quantity: time and frequency.

Seventh, section 6.2 exploits the DN in a number of very different contexts. We find that none of Reservoir Computing, FORCE computing, nor stacked LSTMs can match the performance of the Delay Network given a low-frequency signal relative to the time-step. Furthermore, LSTM cells can be substituted with DN cells, which results in a stacked DN that recovers performance, while being able to predict chaotic time-series data with improved accuracy and reduced training times. We see that the detection of periodic patterns within an input stimulus reduces to taking the L2-norm of a linear transformation of the DN. And, we theoretically justify the utility in stacking Deep Delay Networks (DDNs) by analyzing the error's transfer function and discovering that deep structures provide the better trade-off when controlling for the number of connection weights. Likewise, we show that DDNs can instantaneously propagate signals through multiple layers with relatively long synaptic time-constants. We exploit our earlier extensions to improve the accuracy of discrete-time simulations of continuous neural dynamics. We also demonstrate more accurate implementations of delay networks with a variety of synapse models, thus allowing systematic exploration of the relationship between synapse- and network-level dynamics. Finally we suggest that these methods provide new insights into the observed temporal properties of individual neurons; biological time cell responses during a delay task are well-approximated by a delay network constructed using these methods.

Eighth, sections 7.1 and 7.2 utilize many of the aforementioned extensions together with Nengo to deploy dynamical systems onto state-of-the-art neuromorphic hardware: Braindrop and Loihi. We find that, by applying a few compensatory techniques, both Braindrop and Loihi may be used to implement an integrator with little systematic drift averaged over 10 s trials. When combined with nonlinear feedback, this system serves as the basic building block for arbitrary dynamical systems. Braindrop has higher variability due to analog and temperature-induced variability, but consumes orders of magnitude less power. Next, a small-scale instantiation of the DN is compiled onto both architectures, using identical Nengo code, and 128 neurons per dimension – obtaining nearly equal performance to one another.

Among these contributions, we would like to draw final attention to the following overarching narrative. Recurrent neural networks, especially their spiking instantiations, are notoriously difficult to train. Backpropagation has little to say about: discovering what computational structures are useful to consider; imposing prior knowledge on the initial configuration of the network; harnessing the dimension of time within the dynamics of the computational primitives themselves; and appropriately setting all of the hyperparameters. Sidestepping these matters inevitably poses a challenge to deep learning when placed next to the extraordinary scale of the human brain: 10^{14} connections, capturing temporal dependencies spanning continuous time-scales on the order of seconds to years, while using only 20 W of power. Thus, we view mathematical theories and frameworks as paramount to the future success of deep learning and artificial intelligence, assuming it is ever to become embodied in a similar form to that of biological intelligence. This motivates our careful exploration into the Neural Engineering Framework and its methods of harnessing dynamical systems to realize useful computations latent within the distributed activity of biologically-plausible recurrent spiking neural networks. We extend this framework, and derive a class of useful dynamical systems that have been largely ignored in the past: delay networks. The linear dynamics of these systems optimally project its input history onto a low-dimensional state-space that becomes nonlinearly encoded into the high-dimensional activity spaces of neurons. This approach outperforms equivalently-sized and trained state-of-the-art recurrent neural networks, including stacked LSTMs, in basic measures of continuous-time information processing capacity, prediction of a chaotic time-series, and training time. Finally, this same system is implemented on the neuromorphic hardware architectures of Braindrop and Loihi, with plenty of room to grow.

References

Aamir, S. A., P. Müller, G. Kiene, L. Kriener, Y. Stradmann, A. Grübl, J. Schemmel, and K. Meier
2018a. A mixed-signal structured AdEx neuron for accelerated neuromorphic cores. *IEEE Transactions on Biomedical Circuits and Systems*, (99):1–11.

Aamir, S. A., Y. Stradmann, P. Müller, C. Pehle, A. Hartel, A. Grübl, J. Schemmel, and K. Meier
2018b. An accelerated LIF neuronal network array for a large-scale mixed-signal neuromorphic architecture. *IEEE Transactions on Circuits and Systems I: Regular Papers*, (99):1–14.

Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al.
2016. TensorFlow: A system for large-scale machine learning. In *OSDI*, volume 16, Pp. 265–283.

Abbott, L., B. DePasquale, and R.-M. Memmesheimer
2016. Building functional networks of spiking model neurons. *Nature Neuroscience*, 19(3):350–355.

Abbott, L. and W. G. Regehr
2004. Synaptic computation. *Nature*, 431(7010):796–803.

Adrian, E. D.
1928. *The basis of sensation.* Christophers; London, 22 Berners Steeet, W. 1.

Alemi, A., C. K. Machens, S. Denève, and J.-J. Slotine
2018. Learning nonlinear dynamics in efficient, balanced spiking networks using local plasticity rules. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Ambrogio, S., P. Narayanan, H. Tsai, R. M. Shelby, I. Boybat, C. Nolfo, S. Sidler, M. Giordano, M. Bodini, N. C. Farinha, et al.
2018. Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature*, 558(7708):60.

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

Amodei, D., S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, et al.
2016. Deep Speech 2: End-to-end speech recognition in English and Mandarin. In *International Conference on Machine Learning,* Pp. 173–182.

Anderson, J. R. and C. Lebiere
1998. *The atomic components of thought.* Erlbaum Associates.

Andreou, A. G., K. A. Boahen, P. O. Pouliquen, A. Pavasovic, R. E. Jenkins, and K. Strohbehn
1991. Current-mode subthreshold MOS circuits for analog VLSI neural systems. *IEEE Transactions on neural networks*, 2(2):205–213.

Appeltant, L., M. C. Soriano, G. Van der Sande, J. Danckaert, S. Massar, J. Dambre, B. Schrauwen, C. R. Mirasso, and I. Fischer
2011. Information processing using a single dynamical node as complex system. *Nature communications*, 2:468.

Armstrong-Gold, C. E. and F. Rieke
2003. Bandpass filtering at the rod to second-order cell synapse in salamander (Ambystoma tigrinum) retina. *The Journal of Neuroscience*, 23(9):3796–3806.

Arthur, J. V. and K. Boahen
2011. Silicon-neuron design: A dynamical systems approach. *IEEE Transactions on Circuits and Systems*, 58(5):1034–1043.

Aubin, S.
2017. Representing and combining dynamics in biologically plausible neurons. Technical report, Centre for Theoretical Neuroscience, Waterloo, ON.

Aubin, S.
2018. Learning and leveraging neural memories. Masters thesis, University of Waterloo.

Aubin, S., A. R. Voelker, and C. Eliasmith
2017. Improving with practice: A neural model of mathematical development. *Topics in Cognitive Science*, 9(1):6–20.

Azghadi, M. R., S. Moradi, D. B. Fasnacht, M. S. Ozdas, and G. Indiveri
2015. Programmable spike-timing-dependent plasticity learning circuits in neuromorphic VLSI architectures. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 12(2):17.

Bahdanau, D., K. Cho, and Y. Bengio
2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bai, K. and Y. Yi
2018. DFR: An energy-efficient analog delay feedback reservoir computing system for brain-inspired computing. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 14(4):45.

Bartolozzi, C., R. Benosman, K. Boahen, G. Cauwenberghs, T. Delbrück, G. Indiveri, S.-C. Liu, S. Furber, N. Imam, B. Linares-Barranco, et al.
1999. Neuromorphic systems. *Wiley Encyclopedia of Electrical and Electronics Engineering*, Pp. 1–22.

Bekolay, T.
2011. Learning in large-scale spiking neural networks. Master's thesis, University of Waterloo, Waterloo, ON.

Bekolay, T., J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, D. Rasmussen, X. Choo, A. R. Voelker, and C. Eliasmith
2014. Nengo: A Python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, 7(48).

Bekolay, T., C. Kolbeck, and C. Eliasmith
2013. Simultaneous unsupervised and supervised learning of cognitive functions in biologically plausible spiking neural networks. In *35th Annual Conference of the Cognitive Science Society*, Pp. 169–174. Cognitive Science Society.

Bellec, G., D. Salaj, A. Subramoney, R. Legenstein, and W. Maass
2018. Long short-term memory and learning-to-learn in networks of spiking neurons. *arXiv preprint arXiv:1803.09574*.

Bengio, Y. and P. Frasconi
1994. Credit assignment through time: Alternatives to backpropagation. In *Advances in Neural Information Processing Systems*, Pp. 75–82.

Bengio, Y., P. Simard, and P. Frasconi
1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Benjamin, B. V. and K. Boahen
2019. Mismatch and temperature aware compact MOS model for subthreshold circuit design. *(submitted) IEEE Transactions on Circuits and Systems I: Regular Papers*.

Benjamin, B. V., P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen
2014. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, 102(5):699–716.

Bennett, C. H.
1990. Undecidable dynamics. *Nature*, 346:606–607.

Bennett, C. H.
1995. Universal computation and physical dynamics. *Physica D: Nonlinear Phenomena*, 86(1-2):268–273.

Bergstra, J. and Y. Bengio
2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

Bergstra, J., B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox
2015. Hyperopt: A Python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008.

Bergstra, J., D. Yamins, and D. D. Cox
2013. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning*. ICML.

Berry, A. C.
1941. The accuracy of the Gaussian approximation to the sum of independent variates. *Transactions of the american mathematical society*, 49(1):122–136.

Berzish, M., C. Eliasmith, and B. Tripp
2016. Real-time FPGA simulation of surrogate models of large spiking networks. In *International Conference on Artificial Neural Networks (ICANN)*.

Binas, J., D. Neil, G. Indiveri, S.-C. Liu, and M. Pfeiffer
2016. Precise deep neural network computation on imprecise low-power analog hardware. *arXiv preprint arXiv:1606.07786*.

Blondel, V. D., O. Bournez, P. Koiran, and J. N. Tsitsiklis
2001. The stability of saturated linear dynamical systems is undecidable. *Journal of Computer and System Sciences*, 62(3):442–462.

Blouw, P., X. Choo, E. Hunsberger, and C. Eliasmith
2018. Benchmarking keyword spotting efficiency on neuromorphic hardware. *arXiv preprint arXiv:1812.01739*.

Blumensath, T. and M. E. Davies
2009. Sampling theorems for signals from the union of finite-dimensional linear subspaces. *IEEE Transactions on Information Theory*, 55(4):1872–1882.

Boahen, K.
2017. A neuromorph's prospectus. *Computing in Science & Engineering*, 19(2):14–28.

Boahen, K. A., P. O. Pouliquen, A. G. Andreou, and R. E. Jenkins
1989. A heteroassociative memory using current-mode MOS analog VLSI circuits. *IEEE Transactions on Circuits and Systems*, 36(5):747–755.

Boerlin, M. and S. Denève
2011. Spike-based population coding and working memory. *PLOS Computational Biology*, 7(2):e1001080.

Boerlin, M., C. K. Machens, and S. Denève
2013. Predictive coding of dynamical variables in balanced spiking networks. *PLOS Computational Biology*, 9(11):e1003258.

Boybat, I., M. Le Gallo, S. Nandakumar, T. Moraitis, T. Parnell, T. Tuma, B. Rajendran, Y. Leblebici, A. Sebastian, and E. Eleftheriou
2018. Neuromorphic computing with multi-memristive synapses. *Nature communications*, 9(1):2514.

Brette, R.
2015. Philosophy of the spike: rate-based vs. spike-based theories of the brain. *Frontiers in Systems Neuroscience*, 9:151.

Brink, S., S. Nease, and P. Hasler
2013. Computing with networks of spiking neurons on a biophysically motivated floating-gate based neuromorphic integrated circuit. *Neural Networks*, 45:39–49.

Brockett, R. W.
1991. Dynamical systems that sort lists, diagonalize matrices, and solve linear programming problems. *Linear Algebra and its Applications*, 146:79–91.

Broersma, H., S. Stepney, and G. Wendin
2018. Computability and complexity of unconventional computing devices. In *Computational Matter*, Pp. 185–229. Springer.

Brogan, W. L.
1991. *Modern Control Theory (3rd Edition)*. Prentice-Hall.

Broomhead, D. S. and D. Lowe
1988. Radial basis functions, multi-variable functional interpolation and adaptive networks. Technical report, Royal Signals and Radar Establishment Malvern (United Kingdom).

Cady, N., K. Beckmann, W. Olin-Ammentorp, G. Chakma, S. Amer, R. Weiss, S. Sayyaparaju, M. Adnan, J. Murray, M. Dean, et al.
2018. Full CMOS-memristor implementation of a dynamic neuromorphic architecture. In *GOMACTech Conference*.

Camera, G. L., A. Rauch, H.-R. Lüscher, W. Senn, and S. Fusi
2004. Minimal models of adapted neuronal response to in vivo-like input currents. *Neural Computation*, 16(10):2101–2124.

Cao, Y., Y. Chen, and D. Khosla
2015. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113(1):54–66.

Cassidy, A. S., R. Alvarez-Icaza, F. Akopyan, J. Sawada, J. V. Arthur, P. A. Merolla, P. Datta, M. G. Tallada, B. Taba, A. Andreopoulos, et al.
2014. Real-time scalable cortical computing at 46 giga-synaptic OPS/watt with ~100x speedup in time-to-solution and ~100,000x reduction in energy-to-solution. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Pp. 27–38. IEEE.

Cassidy, A. S., J. Georgiou, and A. G. Andreou
2013. Design of silicon brains in the nano-CMOS era: Spiking neurons, learning synapses and neural architecture optimization. *Neural Networks*, 45:4–26.

Chang, T., Y. Yang, and W. Lu
2013. Building neuromorphic circuits with memristive devices. *IEEE Circuits and Systems Magazine*, 13(2):56–73.

Chen, G. K., R. Kumar, H. E. Sumbul, P. C. Knag, and R. K. Krishnamurthy
2018a. A 4096-neuron 1M-synapse 3.8-pJ/SOP spiking neural network with on-chip STDP learning and sparse weights in 10-nm finFET CMOS. *IEEE Journal of Solid-State Circuits*.

Chen, T. Q., Y. Rubanova, J. Bettencourt, and D. K. Duvenaud
2018b. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, Pp. 6572–6583.

Cheng, R., U. Goteti, and M. C. Hamilton
2019. Superconducting neuromorphic computing using quantum phase-slip junctions. *IEEE Transactions on Applied Superconductivity*.

Chklovskii, D. B. and D. Soudry
2012. Neuronal spike generation mechanism as an oversampling, noise-shaping A-to-D converter. In *Advances in Neural Information Processing Systems*, Pp. 503–511.

Cho, K., B. Van Merriënboer, D. Bahdanau, and Y. Bengio
2014a. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio
2014b. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Choo, X.
2010. The ordinal serial encoding model: Serial memory in spiking neurons. Master's thesis, University of Waterloo, Waterloo, ON.

Choo, X.
2018. *Spaun 2.0: Extending the World's Largest Functional Brain Model*. Phd thesis, University of Waterloo.

Choudhary, S., S. Sloan, S. Fok, A. Neckar, E. Trautmann, P. Gao, T. Stewart, C. Eliasmith, and K. Boahen
2012. Silicon neurons that compute. In *International Conference on Artificial Neural Networks*, volume 7552, Pp. 121–128. Springer.

Chung, J., C. Gulcehre, K. Cho, and Y. Bengio
2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Cohen, A., R. Douglas, C. Koch, T. Sejnowski, S. Shamma, T. Horiuchi, G. Indiveri, et al.
2001. Report to the National Science Foundation: Workshop on neuromorphic engineering. Technical report.

Conway, J. H. and N. J. A. Sloane
1999. *Sphere packings, lattices and groups*, volume 290. Springer Science & Business Media.

Corless, R. M., G. H. Gonnet, D. E. Hare, D. J. Jeffrey, and D. E. Knuth
1996. On the Lambert W function. *Advances in Computational Mathematics*, 5(1):329–359.

Corradi, F., C. Eliasmith, and G. Indiveri
2014. Mapping arbitrary mathematical functions and dynamical systems to neuromorphic VLSI circuits for spike-based neural computation. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, Melbourne.

Costa, R., I. A. Assael, B. Shillingford, N. de Freitas, and T. Vogels
2017. Cortical microcircuits as gated-recurrent neural networks. In *Advances in Neural Information Processing Systems*, Pp. 272–283.

Coulter, W. K., C. J. Hillar, G. Isley, and F. T. Sommer
2010. Adaptive compressed sensing–a new class of self-organizing coding models for neuroscience. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Pp. 5494–5497. IEEE.

Cover, T. M. and J. A. Thomas
2012. *Elements of information theory*. John Wiley & Sons.

Crawford, E., M. Gingerich, and C. Eliasmith
2015. Biologically plausible, human-scale knowledge representation. *Cognitive Science*.

Crunchbase
2018. Femtosense | Crunchbase. https://www.crunchbase.com/organization/femtosense. Accessed: 2018-11-29.

Cunningham, J. P. and M. Y. Byron
2014. Dimensionality reduction for large-scale neural recordings. *Nature Neuroscience*, 17(11):1500–1509.

Daffron, C., J. Chan, A. Disney, L. Bechtel, R. Wagner, M. E. Dean, G. S. Rose, J. S. Plank, J. D. Birdwell, and C. D. Schuman
2016. Extensions and enhancements for the DANNA neuromorphic architecture. In *SoutheastCon 2016*, Pp. 1–4. IEEE.

Dambre, J., D. Verstraeten, B. Schrauwen, and S. Massar
2012. Information processing capacity of dynamical systems. *Scientific Reports*, 2:514.

Davies, M., N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, et al.
2018. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99.

Davies, S., T. C. Stewart, C. Eliasmith, and S. Furber
2013. Spike-based learning of transfer functions with the SpiNNaker neuromimetic simulator. In *International Joint Conference on Neural Networks (IJCNN)*.

Dayan, P. and L. F. Abbott
2001. *Theoretical neuroscience*, volume 806. Cambridge, MA: MIT Press.

De Vries, B. and J. C. Principe
1992. The gamma model – a new neural model for temporal processing. *Neural Networks*, 5(4):565–576.

Denève, S. and C. K. Machens
2016. Efficient codes and balanced networks. *Nature Neuroscience*, 19(3):375–382.

DePasquale, B., M. M. Churchland, and L. Abbott
2016. Using firing-rate dynamics to train recurrent networks of spiking model neurons. *arXiv preprint arXiv:1601.07620*.

DePasquale, B., C. J. Cueva, K. Rajan, L. Abbott, et al.
2018. full-FORCE: A target-based method for training recurrent networks. *PLOS ONE*, 13(2):e0191527.

Destexhe, A., Z. F. Mainen, and T. J. Sejnowski
1994a. An efficient method for computing synaptic conductances based on a kinetic model of receptor binding. *Neural Computation*, 6(1):14–18.

Destexhe, A., Z. F. Mainen, and T. J. Sejnowski
1994b. Synthesis of models for excitable membranes, synaptic transmission and neuromodulation using a common kinetic formalism. *Journal of Computational Neuroscience*, 1(3):195–230.

Destexhe, A., Z. F. Mainen, and T. J. Sejnowski
1998. Kinetic models of synaptic transmission. *Methods in Neuronal Modeling*, 2:1–25.

Dethier, J., P. Nuyujukian, C. Eliasmith, T. C. Stewart, S. A. Elasaad, K. V. Shenoy, and K. A. Boahen
2011. A brain-machine interface operating with a real-time spiking neural network control algorithm. In *Advances in Neural Information Processing Systems*, Pp. 2213–2221.

Detorakis, G., C. A. Sadique Sheik, S. Paul, B. U. Pedroni, N. Dutt, J. Krichmar, G. Cauwenberghs, and E. Neftci
2018. Neural and synaptic array transceiver: A brain-inspired computing framework for embedded learning. *Frontiers in Neuroscience*, 12.

DeWolf, T. and C. Eliasmith
2017. Trajectory generation using a spiking neuron implementation of dynamic movement primitives. *27th Annual Meeting for the Society for the Neural Control of Movement.*

DeWolf, T., T. C. Stewart, J.-J. Slotine, and C. Eliasmith
2016. A spiking neural model of adaptive arm control. *Proceedings of the Royal Society B*, 283(48).

Diehl, P. U., G. Zarrella, A. Cassidy, B. U. Pedroni, and E. Neftci
2016. Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, Pp. 1–8. IEEE.

Duggins, P.
2017. Incorporating biologically realistic neuron models into the NEF. Master's thesis, University of Waterloo.

Duggins, P., T. C. Stewart, X. Choo, and C. Eliasmith
2017. Effects of guanfacine and phenylephrine on a spiking neuron model of working memory. *Topics in Cognitive Science.*

Ebrahimi Kahou, S., V. Michalski, K. Konda, R. Memisevic, and C. Pal
2015. Recurrent neural networks for emotion recognition in video. In *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, Pp. 467–474. ACM.

Eichenbaum, H.
2014. Time cells in the hippocampus: A new dimension for mapping memories. *Nature Reviews Neuroscience*, 15(11):732–744.

Eikenberry, S. E. and V. Z. Marmarelis
2015. Principal dynamic mode analysis of the Hodgkin–Huxley equations. *International Journal of Neural Systems*, 25(02):1550001.

Eliasmith, C.
2005. A unified approach to building and controlling spiking attractor networks. *Neural Computation*, 7(6):1276–1314.

Eliasmith, C.
2013. *How to build a brain: A neural architecture for biological cognition.* Oxford University Press.

Eliasmith, C. and C. H. Anderson
1999. Developing and applying a toolkit from a general neurocomputational framework. *Neurocomputing*, 26:1013–1018.

Eliasmith, C. and C. H. Anderson
2000. Rethinking central pattern generators: A general approach. *Neurocomputing*, 32–33:735–740.

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

Eliasmith, C., J. Gosmann, and X. Choo
2016. BioSpaun: A large-scale behaving brain model with complex neurons. *arXiv preprint arXiv:1602.05220*.

Eliasmith, C., T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen
2012. A large-scale model of the functioning brain. *Science*, 338(6111):1202–1205.

Esseen, C.-G.
1942. On the Liapunov limit error in the theory of probability. *Ark. Mat. Astr. Fys.*, 28:1–19.

Esser, S. K., R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha
2015. Backpropagation for energy-efficient neuromorphic computing. In *Advances in Neural Information Processing Systems*, Pp. 1117–1125.

Esser, S. K., P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, et al.
2016. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences*, 113(41):11441–11446.

Evans, D. J. and D. J. Searles
1996. Causality, response theory, and the second law of thermodynamics. *Physical Review E*, 53(6):5808.

Fairhall, A. L., G. D. Lewen, W. Bialek, and R. R. d. R. van Steveninck
2001. Efficiency and ambiguity in an adaptive neural code. *Nature*, 412(6849):787.

Fang, K. T. and Y. Wang
1994. *Number-Theoretic Methods in Statistics*. Chapman & Hall.

Feinstein, D. I.
1988. The hexagonal resistive network and the circular approximation.

Fernando, C. and S. Sojakka
2003. Pattern recognition in a bucket. In *European Conference on Artificial Life*, Pp. 588–597. Springer.

Ferreira, P. M. and A. E. Ruano
2009. Online sliding-window methods for process model adaptation. *IEEE Transactions on Instrumentation and Measurement*, 58(9):3012–3020.

Fischl, K. D., T. C. Stewart, K. L. Fair, and A. G. Andreou
2018. Implementation of the Neural Engineering Framework on the TrueNorth neurosynaptic system. In *IEEE Biomedical Circuits and Systems Conference (BioCAS)*, Pp. 587–590. IEEE.

Fok, S.
2018. *Communicating and Computing with Spikes in Neuromorphic Systems*. PhD thesis, Stanford University.

Fok, S. and K. Boahen
2018. A serial H-tree router for two-dimensional arrays. In *24th IEEE International Symposium on Asynchronous Circuits and Systems*. IEEE.

Fok, S., A. Neckar, and K. Boahen
2019. Weighting and summing spike trains by accumulative thinning. In *(submitted) The Thirteenth International Conference on Neuromorphic Systems (ICONS)*.

Frady, E. P. and F. T. Sommer
2019. Robust computation with rhythmic spike patterns. *arXiv preprint arXiv:1901.07718*.

Frenkel, C., M. Lefebvre, J.-D. Legat, and D. Bol
2018. A 0.086-mm^2 12.7-pJ/SOP 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28nm CMOS. *IEEE Transactions on Biomedical Circuits and Systems*.

Friedl, K. E., A. R. Voelker, A. Peer, and C. Eliasmith
2016. Human-inspired neurorobotic system for classifying surface textures by touch. *Robotics and Automation Letters*, 1(1):516–523.

Funahashi, K. and Y. Nakamura
1993. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, 6(6):801–806.

Furber, S.
2012. To build a brain. *IEEE Spectrum*, 49(8).

Furber, S. B., F. Galluppi, S. Temple, and L. A. Plana
2014. The SpiNNaker project. *Proceedings of the IEEE*, 102(5):652–665.

Furber, S. B., D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown
2013. Overview of the SpiNNaker system architecture. *IEEE Transactions on Computers*, 62(12):2454–2467.

Galluppi, F., S. Davies, S. Furber, T. Stewart, and C. Eliasmith
2012. Real time on-chip implementation of dynamical systems with spiking neurons. In *International Joint Conference on Neural Networks (IJCNN)*, Pp. 1–8. IEEE.

Galluppi, F., C. Denk, M. Meiner, T. C. Stewart, L. Plana, C. Eliasmith, S. Furber, and J. Conradt
2014. Event-based neural computing on an autonomous mobile platform. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong.

Ganguli, S., D. Huh, and H. Sompolinsky
2008. Memory traces in dynamical systems. *Proceedings of the National Academy of Sciences*, 105(48):18970–18975.

Gao, P., B. V. Benjamin, and K. Boahen
2012. Dynamical system guided mapping of quantitative neuronal models onto neuromorphic hardware. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 59(10):2383–2394.

Gautrais, J. and S. Thorpe
1998. Rate coding versus temporal order coding: A theoretical approach. *Biosystems*, 48(1-3):57–65.

Gers, F. A., J. Schmidhuber, and F. Cummins
1999. Learning to forget: Continual prediction with LSTM.

Gerstner, W.
1999. Spiking neurons. *Pulsed Neural Networks*, 4:3–54.

Gerstner, W. and R. Naud
2009. How good are neuron models? *Science*, 326(5951):379–380.

Gibney, E.
2016. Google AI algorithm masters ancient game of Go. *Nature News*, 529(7587):445.

Gilra, A. and W. Gerstner
2017. Predicting non-linear dynamics: A stable local learning scheme for recurrent spiking neural networks. *arXiv preprint arXiv:1702.06463*.

GitHub
2019a. nengo/nengo-fpga==0.1.0: Nengo extension to connect to FPGAs. `https://github.com/nengo/nengo-fpga/`. Accessed: 2019-01-20.

GitHub
2019b. nengo/nengo-loihi==0.5.0: Run nengo models on Intel's Loihi chip. `https://github.com/nengo/nengo-loihi/`. Accessed: 2019-01-20.

Glackin, B., J. Harkin, T. M. McGinnity, and L. P. Maguire
2009. A hardware accelerated simulation environment for spiking neural networks. In *International Workshop on Applied Reconfigurable Computing*, Pp. 336–341. Springer.

Glatz, S., J. N. Martel, R. Kreiser, N. Qiao, and Y. Sandamirskaya
2018. Adaptive motor control and learning in a spiking neural network realised on a mixed-signal neuromorphic processor. *arXiv preprint arXiv:1810.10801*.

Glorot, X., A. Bordes, and Y. Bengio
2011. Deep sparse rectifier neural networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, Pp. 315–323.

Glover, K. and J. R. Partington
1987. Bounds on the achievable accuracy in model reduction. In *Modelling, Robustness and Sensitivity Reduction in Control Systems*, Pp. 95–118. Springer.

Goldman, M. S.
2009. Memory without feedback in a neural network. *Neuron*, 61(4):621–634.

Gosmann, J.
2015. Precise multiplications with the NEF. Technical report, Centre for Theoretical Neuroscience, Waterloo, ON.

Gosmann, J.
2018. *An Integrated Model of Contex, Short-Term, and Long-Term Memory*. Phd thesis, University of Waterloo.

Gosmann, J. and C. Eliasmith
2017. Automatic optimization of the computation graph in the Nengo neural network simulator. *Frontiers in Neuroinformatics*, 11:33.

Gosmann, J., A. R. Voelker, and C. Eliasmith
2017. A spiking independent accumulator model for winner-take-all computation. In *Proceedings of the 39th Annual Conference of the Cognitive Science Society*, London, UK. Cognitive Science Society.

Graves, A., A.-R. Mohamed, and G. Hinton
2013. Speech recognition with deep recurrent neural networks. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Pp. 6645–6649. IEEE.

Graves, A. and J. Schmidhuber
2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5-6):602–610.

Graves, A., G. Wayne, and I. Danihelka
2014. Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Gulli, A. and S. Pal
2017. *Deep Learning with Keras*. Packt Publishing Ltd.

Halko, N., P.-G. Martinsson, and J. A. Tropp
2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288.

Häusser, M. and A. Roth
1997. Estimating the time course of the excitatory synaptic conductance in neocortical pyramidal cells using a novel voltage jump method. *The Journal of Neuroscience*, 17(20):7606–7625.

Haykin, S.
1991. *Adaptive Filter Theory*. Prentice Hall.

Hazan, H., D. J. Saunders, H. Khan, D. T. Sanghavi, H. T. Siegelmann, and R. Kozma
2018. BindsNET: A machine learning-oriented spiking neural networks library in Python. *Frontiers in Neuroinformatics*, 12:89.

Hermann, M. R. and J. Fleck Jr
1988. Split-operator spectral method for solving the time-dependent Schrödinger equation in spherical coordinates. *Physical Review A*, 38(12):6000.

Hochreiter, S. and J. Schmidhuber
1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Hoeppner, S., B. Vogginger, Y. Yan, A. Dixius, S. Scholze, J. Partzsch, F. Neumaerker, S. Hartmann, S. Schiefer, G. Ellguth, et al.
2019. Dynamic power management for neuromorphic many-core systems. *arXiv preprint arXiv:1903.08941*.

Hopfield, J. J.
1982. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558.

Hoppensteadt, F.
2017. Applied physics: A new spin on nanoscale computing. *Nature*, 547(7664):407.

Hornik, K., M. Stinchcombe, and H. White
1989. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

Howard, M. W., C. J. MacDonald, Z. Tiganj, K. H. Shankar, Q. Du, M. E. Hasselmo, and H. Eichenbaum
2014. A unified mathematical framework for coding time, space, and sequences in the hippocampal region. *The Journal of Neuroscience*, 34(13):4692–4707.

Huh, D. and T. J. Sejnowski
2018. Gradient descent for spiking neural networks. In *Advances in Neural Information Processing Systems*, Pp. 1440–1450.

Hunsberger, E.
2016. System identification of adapting neurons. Technical report, Centre for Theoretical Neuroscience, Waterloo, ON.

Hunsberger, E.
2018. *Spiking Deep Neural Networks: Engineered and Biological Approaches to Object Recognition.* Phd thesis, University of Waterloo.

Hunsberger, E. and C. Eliasmith
2015. Spiking deep networks with LIF neurons. *arXiv preprint arXiv:1510.08829*.

Hunsberger, E. and C. Eliasmith
2016. Training spiking deep networks for neuromorphic hardware. *arXiv preprint arXiv:1611.05141*.

Hurzook, A.
2012. A mechanistic model of motion processing in the early visual system. Masters thesis, University of Waterloo, Waterloo, Ontario.

Hwang, C. and M.-Y. Chen
1985. Analysis and parameter identification of time-delay systems via shifted Legendre polynomials. *International Journal of Control*, 41(2):403–415.

Ijspeert, A. J., J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal
2013. Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Computation*, 25(2):328–373.

Indiveri, G., B. Linares-Barranco, T. J. Hamilton, A. Van Schaik, R. Etienne-Cummings, T. Delbruck, S.-C. Liu, P. Dudek, P. Häfliger, S. Renaud, et al.
2011. Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience*, 5:73.

Inubushi, M. and K. Yoshimura
2017. Reservoir computing beyond memory-nonlinearity trade-off. *Scientific Reports*, 7(1):10199.

Izhikevich, E. M.
2007. *Dynamical systems in neuroscience*. MIT press.

Izzeldin, H., V. S. Asirvadam, and N. Saad
2011. Online sliding-window based for training MLP networks using advanced conjugate gradient. In *7th International Colloquim on Signal Processing and its Applications (CSPA)*, Pp. 112–116. IEEE.

Jaeger, H.
2001. The "echo state" approach to analysing and training recurrent neural networks. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148:34.

Jaeger, H.
2002. Short term memory in echo state networks. Technical report, Fraunhofer Institute for Autonomous Intelligent Systems.

Jaeger, H.
2014. Controlling recurrent neural networks by conceptors. *arXiv preprint arXiv:1403.3369*.

Jing, L., C. Gulcehre, J. Peurifoy, Y. Shen, M. Tegmark, M. Soljacic, and Y. Bengio
2018. Gated orthogonal recurrent units: On learning to forget. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*.

Joshi, P. and W. Maass
2005. Movement generation with circuits of spiking neurons. *Neural Computation*, 17(8):1715–1738.

Jozefowicz, R., W. Zaremba, and I. Sutskever
2015. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, Pp. 2342–2350.

Kalchbrenner, N., I. Danihelka, and A. Graves
2015. Grid long short-term memory. *arXiv preprint arXiv:1507.01526*.

Kauderer-Abrams, E. and K. Boahen
2017. Calibrating silicon-synapse dynamics using time-encoding and decoding machines. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, Pp. 1–4. IEEE.

Kauderer-Abrams, E., A. Gilbert, A. R. Voelker, B. V. Benjamin, T. C. Stewart, and K. Boahen
2017. A population-level approach to temperature robustness in neuromorphic systems. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, Baltimore, MD. IEEE.

Khaisongkram, W. and D. Banjerdpongchai
2007. On computing the worst-case norm of linear systems subject to inputs with magnitude bound and rate limit. *International Journal of Control*, 80(2):190–219.

Kim, J., J.-W. Ghim, J. H. Lee, and M. W. Jung
2013. Neural correlates of interval timing in rodent prefrontal cortex. *The Journal of Neuroscience*, 33(34):13834–13847.

Kim, J., J. Koo, T. Kim, and J.-J. Kim
2018. Efficient synapse memory structure for reconfigurable digital neuromorphic hardware. *Frontiers in Neuroscience*, 12.

Kim, R., Y. Li, and T. J. Sejnowski
2019. Simple framework for constructing functional spiking recurrent neural networks. *bioRxiv*, P. 579706.

Kim, S.-J., K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky
2007. An interior-point method for large-scale ℓ_1-regularized least squares. *IEEE Journal of Selected Topics in Signal Processing*, 1(4):606–617.

Kim, S. S., A. P. Sripati, and S. J. Bensmaia
2010. Predicting the timing of spikes evoked by tactile stimulation of the hand. *Journal of Neurophysiology*, 104(3):1484–1496.

Kingma, D. P. and J. Ba
2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Knight, J., A. R. Voelker, A. Mundy, C. Eliasmith, and S. Furber
2016. Efficient SpiNNaker simulation of a heteroassociative memory using the Neural Engineering Framework. In *International Joint Conference on Neural Networks (IJCNN)*, Vancouver, British Columbia. IEEE.

Knight, J. C. and T. Nowotny
2018. GPUs outperform current HPC and neuromorphic solutions in terms of speed and energy when simulating a highly-connected cortical model. *Frontiers in Neuroscience*, 12:941.

Koch, C.
2004. *Biophysics of computation: Information processing in single neurons.* Oxford university press.

Koch, C. and I. Segev
1998. *Methods in neuronal modeling: From ions to networks.* MIT press.

Kohonen, T.
1972. Correlation matrix memories. *IEEE Transactions on Computers*, 21(4):353–359.

Komer, B.
2015. Biologically inspired adaptive control of quadcopter flight. Masters thesis, University of Waterloo, Waterloo, ON.

Krichmar, J. L., W. Severa, S. M. Khan, and J. L. Olds
2018. Making BREAD: Biomimetic strategies for artificial intelligence now and in the future. *arXiv preprint arXiv:1812.01184*.

Kudithipudi, D., Q. Saleh, C. Merkel, J. Thesing, and B. Wysocki
2016. Design and analysis of a neuromemristive reservoir computing architecture for biosignal processing. *Frontiers in Neuroscience*, 9:502.

Lagorce, X. and R. Benosman
2015. STICK: Spike time interval computational kernel, a framework for general purpose computation using neurons, precise timing, delays, and synchrony. *Neural Computation*, 27(11):2261–2317.

Larras, B., P. Chollet, C. Lahuec, F. Seguin, and M. Arzel
2018. A fully flexible circuit implementation of clique-based neural networks in 65-nm CMOS. *IEEE Transactions on Circuits and Systems I: Regular Papers.*

Laub, A. J., M. T. Heath, C. Paige, and R. Ward
1987. Computation of system balancing transformations and other applications of simultaneous diagonalization algorithms. *IEEE Transactions on Automatic Control*, 32(2):115–122.

LeCun, Y., Y. Bengio, and G. Hinton
2015. Deep learning. *nature*, 521(7553):436.

Lee, J. H., T. Delbruck, and M. Pfeiffer
2016. Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, 10:508.

Legendre, A.-M.
1782. Recherches sur l'attraction des sphéroïdes homogènes. *Mémoires de Mathématiques et de Physique, présentés à l'Académie Royale des Sciences*, Pp. 411–435.

Leng, G., A. K. Ray, T. McGinnity, S. Coleman, and L. Maguire
2013. Online sliding window based self-organising fuzzy neural network for cognitive reasoning. *COGN 2013*, Pp. 114–119.

Lillicrap, T. P. and A. Santoro
2019. Backpropagation through time and the brain. *Current Opinion in Neurobiology*, 55:82–89.

Lin, C.-K., A. Wild, G. N. Chinya, Y. Cao, M. Davies, D. M. Lavery, and H. Wang
2018a. Programming spiking neural networks on Intel's Loihi. *Computer*, 51(3):52–61.

Lin, C.-K., A. Wild, G. N. Chinya, T.-H. Lin, M. Davies, and H. Wang
2018b. Mapping spiking neural networks onto a manycore neuromorphic architecture. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, Pp. 78–89. ACM.

Lin, T.-H. and P. T. P. Tang
2018. Dictionary learning by dynamical neural networks. *arXiv preprint arXiv:1805.08952.*

Litt, A., C. Eliasmith, F. W. Kroon, S. Weinstein, and P. Thagard
2006. Is the brain a quantum computer? *Cognitive Science*, 30(3):593–603.

Liu, C., G. Bellec, B. Vogginger, D. Kappel, J. Partzsch, S. Höppner, W. Maass, S. B. Furber, R. Legenstein, C. G. Mayr, et al.
2018. Memory-efficient deep learning on a SpiNNaker 2 prototype. *Frontiers in Neuroscience*, 12:840.

Liu, S.-C., J. Kramer, G. Indiveri, T. Delbrück, and R. Douglas
2002. *Analog VLSI: circuits and principles.*

Luczak, A., B. L. McNaughton, and K. D. Harris
2015. Packet-based communication in the cortex. *Nature Reviews Neuroscience.*

Lukoševičius, M.
2012a. A practical guide to applying echo state networks. In *Neural networks: Tricks of the trade*, Pp. 659–686. Springer.

Lukoševičius, M.
2012b. *Reservoir computing and self-organized neural hierarchies*. PhD thesis, Jacobs University Bremen.

Lukoševičius, M. and H. Jaeger
2009. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149.

Lundstrom, B. N., M. H. Higgs, W. J. Spain, and A. L. Fairhall
2008. Fractional differentiation by neocortical pyramidal neurons. *Nature Neuroscience*, 11(11):1335–1342.

Maass, W., T. Natschläger, and H. Markram
2002. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560.

Mackey, M. C. and L. Glass
1977. Oscillation and chaos in physiological control systems. *Science*, 197(4300):287–289.

Mahowald, M. and R. Douglas
1991. A silicon neuron. *Nature*, 354(6354):515.

Mainen, Z. F. and T. J. Sejnowski
1995. Reliability of spike timing in neocortical neurons. *Science*, 268(5216):1503.

Manipatruni, S., D. E. Nikonov, and I. A. Young
2018. Beyond CMOS computing with spin and polarization. *Nature Physics*, 14(4):338.

Marblestone, A. H., G. Wayne, and K. P. Kording
2016. Toward an integration of deep learning and neuroscience. *Frontiers in Computational Neuroscience*, 10:94.

Massey Jr, F. J.
1951. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46(253):68–78.

Mayor, J. and W. Gerstner
2005. Signal buffering in random networks of spiking neurons: Microscopic versus macroscopic phenomena. *Physical Review E*, 72(5):051906.

McCulloch, W. S. and W. Pitts
1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

McKenna, T., T. McMullen, and M. Shlesinger
1994. The brain as a dynamic physical system. *Neuroscience*, 60(3):587–605.

Mead, C.
1989. Analog VLSI and neural systems.

Mead, C. A. and M. A. Mahowald
1988. A silicon model of early visual processing. *Neural Networks*, 1(1):91–97.

Mello, G. B., S. Soares, and J. J. Paton
2015. A scalable population code for time in the striatum. *Current Biology*, 25(9):1113–1122.

Menon, S., S. Fok, A. Neckar, O. Khatib, and K. Boahen
2014. Controlling articulated robots in task-space with spiking silicon neurons. In *5th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics*, Pp. 181–186. IEEE.

Merolla, P., J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha
2011. A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm. In *2011 IEEE custom integrated circuits conference (CICC)*, Pp. 1–4. IEEE.

Merolla, P. A., J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, et al.
2014. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673.

Minsky, M. and S. A. Papert
1969. Perceptrons: An introduction to computational geometry.

Mitchell, J. P., M. E. Dean, G. R. Bruer, J. S. Plank, and G. S. Rose
2018. DANNA 2: Dynamic adaptive neural network arrays. In *Proceedings of the International Conference on Neuromorphic Systems*, P. 10. ACM.

Mitra, P. P. and J. B. Stark
2001. Nonlinear limits to the information capacity of optical fibre communications. *Nature*, 411(6841):1027.

Moore, C.
1990. Unpredictability and undecidability in dynamical systems. *Physical Review Letters*, 64(20):2354.

Moore, C.
1991. Generalized shifts: Unpredictability and undecidability in dynamical systems. *Nonlinearity*, 4(2):199.

Moradi, S. and G. Indiveri
2011. A VLSI network of spiking neurons with an asynchronous static random access memory. In *2011 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, Pp. 277–280. IEEE.

Moradi, S. and G. Indiveri
2014. An event-based neural network architecture with an asynchronous programmable synaptic memory. *IEEE Transactions on Biomedical Circuits and Systems*, 8(1):98–107.

Moradi, S., N. Qiao, F. Stefanini, and G. Indiveri
2018. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs). *IEEE Transactions on Biomedical Circuits and Systems*, 12(1):106–122.

Mundy, A.
2016. *Real time Spaun on SpiNNaker*. PhD thesis, PhD thesis, University of Manchester.

Mundy, A., J. Knight, T. Stewart, and S. Furber
2015. An efficient SpiNNaker implementation of the Neural Engineering Framework. In *International Joint Conference on Neural Networks (IJCNN)*, Pp. 1–8.

Naylor, M., P. J. Fox, A. T. Markettos, and S. W. Moore
2013. Managing the FPGA memory wall: Custom computing or vector processing? In *2013 23rd International Conference on Field programmable Logic and Applications*, Pp. 1–6. IEEE.

Neckar, A.
2018. *Braindrop: A Mixed-Signal Neuromorphic Architecture with a Dynamical Systems-Based Programming Model*. PhD thesis, Stanford University.

Neckar, A., S. Fok, B. V. Benjamin, T. C. Stewart, N. N. Oza, A. R. Voelker, C. Eliasmith, R. Manohar, and K. Boahen
2019. Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model. *Proceedings of the IEEE (Accepted)*.

Neckar, A., T. C. Stewart, B. V. Benjamin, and K. Boahen
2018. Optimizing an analog neuron circuit design for nonlinear function approximation. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, Pp. 1–5. IEEE.

Neftci, E. O., C. Augustine, S. Paul, and G. Detorakis
2017. Event-driven random back-propagation: Enabling neuromorphic deep learning machines. *Frontiers in Neuroscience*, 11:324.

Neftci, E. O., H. Mostafa, and F. Zenke
2019. Surrogate gradient learning in spiking neural networks. *arXiv preprint arXiv:1901.09948*.

Neil, D., M. Pfeiffer, and S.-C. Liu
2016. Phased LSTM: Accelerating recurrent network training for long or event-based sequences. In *Advances in Neural Information Processing Systems*, Pp. 3882–3890.

Nelles, O.
2013. *Nonlinear system identification: From classical approaches to neural networks and fuzzy models*. Springer Science & Business Media.

Newell, A., H. A. Simon, et al.
1972. *Human problem solving*, volume 104. Prentice-Hall Englewood Cliffs, NJ.

Nicola, W. and C. Clopath
2017. Supervised learning in spiking neural networks with FORCE training. *Nature communications*, 8(1):2208.

Noakes, L.
1991. The Takens embedding theorem. *International Journal of Bifurcation and Chaos*, 1(04):867–872.

Oja, E.
1989. Neural networks, principal components, and subspaces. *International Journal of Neural Systems*, 1(01):61–68.

O'Keefe, J. and M. L. Recce
1993. Phase relationship between hippocampal place units and the EEG theta rhythm. *Hippocampus*, 3(3):317–330.

Olazaran, M.
1996. A sociological study of the official history of the perceptrons controversy. *Social Studies of Science*, 26(3):611–659.

Orchard, J., H. Yang, and X. Ji
2013. Does the entorhinal cortex use the fourier transform? *Frontiers in Computational Neuroscience*, 7:179.

Padé, H.
1892. Sur la représentation approchée d'une fonction par des fractions rationnelles. *Annales scientifiques de l'École Normale Supérieure*, 9:3–93.

Painkras, E., L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber
2013. SpiNNaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation. *IEEE Journal of Solid-State Circuits*, 48(8):1943–1953.

Park, J., S. Ha, T. Yu, E. Neftci, and G. Cauwenberghs
2014. A 65k-neuron 73-Mevents/s 22-pJ/event asynchronous micro-pipelined integrate-and-fire array transceiver. In *2014 IEEE Biomedical Circuits and Systems Conference (BioCAS) Proceedings*, Pp. 675–678. IEEE.

Partzsch, J., S. Höppner, M. Eberlein, R. Schüffny, C. Mayr, D. R. Lester, and S. Furber
2017. A fixed point exponential function accelerator for a neuromorphic many-core system. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, Pp. 1–4. IEEE.

Pascanu, R., C. Gulcehre, K. Cho, and Y. Bengio
2013a. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*.

Pascanu, R., T. Mikolov, and Y. Bengio
2013b. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, Pp. 1310–1318.

Payvand, M., M. V. Nair, L. K. Müller, and G. Indiveri
2018. A neuromorphic systems approach to in-memory computing with non-ideal memristive devices: From mitigation to exploitation. *Faraday Discussions*.

Pehlevan, C.
2019. A spiking neural network with local learning rules derived from nonnegative similarity matching. *arXiv preprint arXiv:1902.01429*.

Perev, K.
2011. Approximation of pure time delay elements by using hankel norm and balanced realizations. *Problems of Engineering Cybernetics and Robotics*, 64:24–37.

Pfeiffer, M. and T. Pfeil
2018. Deep learning with spiking neurons: Opportunities and challenges. *Frontiers in Neuroscience*, 12.

Pfeil, T., A. Grübl, S. Jeltsch, E. Müller, P. Müller, M. A. Petrovici, M. Schmuker, D. Brüderle, J. Schemmel, and K. Meier
2013. Six networks on a universal neuromorphic computing substrate. *Frontiers in Neuroscience*, 7:11.

Pollack, J. B.
1988. Recursive auto-associative memory: Devising compositional distributed representations. In *Proceedings of the 10th Annual Conference of the Cognitive Science Society*.

Port, R. F. and T. Van Gelder
1995. *Mind as motion: Explorations in the dynamics of cognition.* MIT Press.

Pozzi, I., R. Nusselder, D. Zambrano, and S. Bohté
2018. Gating sensory noise in a spiking subtractive LSTM. In *International Conference on Artificial Neural Networks*, Pp. 284–293. Springer.

Pulvermüller, F., N. Birbaumer, W. Lutzenberger, and B. Mohr
1997. High-frequency brain activity: Its possible role in attention, perception and language processing. *Progress in neurobiology*, 52(5):427–445.

Qiao, N., H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, and G. Indiveri
2015. A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses. *Frontiers in Neuroscience*, 9:141.

QY Research
2018. Global neuromorphic computing systems market size, status and forecast 2018-2025. Technical report, QY Research.

Rajendran, B., A. Sebastian, M. Schmuker, N. Srinivasa, and E. Eleftheriou
2019. Low-power neuromorphic hardware for signal processing applications. *arXiv preprint arXiv:1901.03690*.

Rall, W.
1967. Distinguishing theoretical synaptic potentials computed for different soma-dendritic distributions of synaptic input. *Journal of Neurophysiology*, 30(5):1138–1168.

Rasmussen, D.
2018. NengoDL: Combining deep learning and neuromorphic modelling methods. *arXiv preprint arXiv:1805.11144*.

Rasmussen, D., A. R. Voelker, and C. Eliasmith
2017. A neural model of hierarchical reinforcement learning. *PLOS ONE*, 12(7):1–39.

Reid, S., A. Montoya, and K. Boahen
2019. PinT: Polynomial in temperature decode weights in a neuromorphic architecture. In *Proceedings of the 2019 Design, Automation & Test in Europe (DATE)*.

Rhodes, O., P. A. Bogdan, C. Brenninkmeijer, S. Davidson, D. Fellows, A. Gait, D. R. Lester, M. Mikaitis, L. A. Plana, A. G. Rowley, et al.
2018. sPyNNaker: A software package for running PyNN simulations on SpiNNaker. *Frontiers in Neuroscience*, 12:816.

Rieke, F. and D. Warland
1997. *Spikes: Exploring the neural code*. MIT press.

Roberts, M.
2018. The unreasonable effectiveness of quasirandom sequences. http://extremelearning.com.au/unreasonable-effectiveness-of-quasirandom-sequences/. Accessed: 2019-01-22.

Rodrigues, O.
1816. *De l'attraction des sphéroïdes, Correspondence sur l'É-cole Impériale Polytechnique*. PhD thesis, Thesis for the Faculty of Science of the University of Paris.

Rosenblatt, F.
1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Roth, A. and M. C. van Rossum
2009. Modeling synapses. *Computational modeling methods for neuroscientists*, 6:139–160.

Roxin, A., N. Brunel, and D. Hansel
2005. Role of delays in shaping spatiotemporal dynamics of neuronal activity in large networks. *Physical Review Letters*, 94(23):238103.

Rueckauer, B., I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu
2017. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, 11:682.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams
1986. Learning representations by back-propagating errors. *nature*, 323(6088):533.

Sak, H., A. Senior, and F. Beaufays
2014. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth annual conference of the international speech communication association*.

Salehinejad, H., J. Baarbe, S. Sankar, J. Barfett, E. Colak, and S. Valaee
2017. Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078*.

Schäfer, A. M. and H. G. Zimmermann
2006. Recurrent neural networks are universal approximators. In *International Conference on Artificial Neural Networks*, Pp. 632–640. Springer.

Schemmel, J., D. Briiderle, A. Griibl, M. Hock, K. Meier, and S. Millner
2010. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, Pp. 1947–1950. IEEE.

Schuecker, J., S. Goedeke, and M. Helias
2018. Optimal sequence memory in driven random networks. *Physical Review X*, 8(4):041029.

Schuman, C. D., T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank
2017. A survey of neuromorphic computing and neural networks in hardware. *arXiv preprint arXiv:1705.06963*.

Schwemmer, M. A., A. L. Fairhall, S. Denéve, and E. T. Shea-Brown
2015. Constructing precisely computing networks with biophysical spiking neurons. *The Journal of Neuroscience*, 35(28):10112–10134.

Sejnowski, T. J.
2018. *The deep learning revolution*. MIT Press.

Severa, W., C. M. Vineyard, R. Dellana, S. J. Verzi, and J. B. Aimone
2018. Whetstone: A method for training deep artificial neural networks for binary communication. *arXiv preprint arXiv:1810.11521*.

Shainline, J. M.
2018. The largest cognitive systems will be optoelectronic. *arXiv preprint arXiv:1809.02572*.

Shankar, K. H. and M. W. Howard
2012. A scale-invariant internal representation of time. *Neural Computation*, 24(1):134–193.

Shapero, S., M. Zhu, J. Hasler, and C. Rozell
2014. Optimal sparse approximation with integrate and fire neurons. *International Journal of Neural Systems*, 24(05):1440001.

Sharma, S., A. R. Voelker, and C. Eliasmith
2017. A spiking neural Bayesian model of life span inference. In *Proceedings of the 39th Annual Conference of the Cognitive Science Society*, London, UK. Cognitive Science Society.

Shrestha, S. B. and G. Orchard
2018. SLAYER: Spike layer error reassignment in time. In *Advances in Neural Information Processing Systems*, Pp. 1419–1428.

Sidi, A.
2003. *Practical extrapolation methods: Theory and applications*, volume 10. Cambridge University Press.

Singer, W.
1999. Neuronal synchrony: A versatile code for the definition of relations? *Neuron*, 24(1):49–65.

Singh, R. and C. Eliasmith
2004. A dynamic model of working memory in the PFC during a somatosensory discrimination task. Cold Spring Harbour, NY.

Singh, R. and C. Eliasmith
2006. Higher-dimensional neurons explain the tuning and dynamics of working memory cells. *The Journal of Neuroscience*, 26:3667–3678.

Sivilotti, M. A., M. Emerling, and C. Mead
1985. A novel associative memory implemented using collective computation. In *Chapel Hill Conference on Very Large Scale Integration*, Pp. 329–339.

Smith, M. R., A. J. Hill, K. D. Carlson, C. M. Vineyard, J. Donaldson, D. R. Follett, P. L. Follett, J. H. Naegle, C. D. James, and J. B. Aimone
2017. A novel digital neuromorphic architecture efficiently facilitating complex synaptic response functions applied to liquid state machines. In *International Joint Conference on Neural Networks (IJCNN)*, Pp. 2421–2428. IEEE.

Sobol, I. M.
1967. On the distribution of points in a cube and the approximate evaluation of integrals. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 7(4):784–802.

Stewart, T. C., T. DeWolf, A. Kleinhans, and C. Eliasmith
2015. Closed-loop neuromorphic benchmarks. *Frontiers in Neuroscience*, 9:464.

Stewart, T. C. and C. Eliasmith
2014. Large-scale synthesis of functional spiking neural circuits. *Proceedings of the IEEE*, 102(5):881–898.

Stewart, T. C., Y. Tang, and C. Eliasmith
2011. A biologically realistic cleanup memory: Autoassociation in spiking neurons. *Cognitive Systems Research*, 12(2):84–92.

Stöckel, A., C. Jenzen, M. Thies, and U. Rückert
2017a. Binary associative memories as a benchmark for spiking neuromorphic hardware. *Frontiers in Computational Neuroscience*, 11:71.

Stöckel, A., A. R. Voelker, and C. Eliasmith
2017b. Point neurons with conductance-based synapses in the Neural Engineering Framework. Technical report, Centre for Theoretical Neuroscience, Waterloo, ON.

Stöckel, A., A. R. Voelker, and C. Eliasmith
2018. Nonlinear synaptic interaction as a computational resource in the Neural Engineering Framework. In *Cosyne Abstracts*, Denver USA.

Strogatz, S. H.
2015. *Nonlinear dynamics and chaos: With applications to physics, biology, chemistry, and engineering (2nd Edition).* Westview Press.

Stromatias, E., F. Galluppi, C. Patterson, and S. Furber
2013. Power analysis of large-scale, real-time neural networks on SpiNNaker. In *International Joint Conference on Neural Networks (IJCNN)*, Pp. 1–8. IEEE.

Su, J., D. V. Vargas, and K. Sakurai
2019. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation.*

Sugiarto, I., G. Liu, S. Davidson, L. A. Plana, and S. B. Furber
2016. High performance computing on SpiNNaker neuromorphic platform: A case study for energy efficient image processing. In *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*, Pp. 1–8. IEEE.

Sugihara, G., R. May, H. Ye, C.-h. Hsieh, E. Deyle, M. Fogarty, and S. Munch
2012. Detecting causality in complex ecosystems. *Science*, 338(6106):496–500.

Sussillo, D. and L. F. Abbott
2009. Generating coherent patterns of activity from chaotic neural networks. *Neuron*, 63(4):544–557.

Sussillo, D. and O. Barak
2013. Opening the black box: Low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural Computation*, 25(3):626–649.

Sutskever, I., O. Vinyals, and Q. V. Le
2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, Pp. 3104–3112.

Szendro, P., G. Vincze, and A. Szasz
2001. Bio-response to white noise excitation. *Electro-and Magnetobiology*, 20(2):215–229.

Tait, A. N., T. F. de Lima, M. A. Nahmias, H. B. Miller, H.-T. Peng, B. J. Shastri, and P. R. Prucnal
2018. A silicon photonic modulator neuron. *arXiv preprint arXiv:1812.11898*.

Takens, F.
1981. Detecting strange attractors in turbulence. In *Dynamical Systems and Turbulence*, Pp. 366–381. Springer.

Tang, G., A. Shah, and K. P. Michmizos
2019. Spiking neural network on neuromorphic hardware for energy-efficient unidimensional SLAM. *arXiv preprint arXiv:1903.02504*.

Tang, P. T. P., T.-H. Lin, and M. Davies
2017. Sparse coding by spiking neural networks: Convergence theory and computational results. *arXiv preprint arXiv:1705.05475*.

Tavanaei, A., M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida
2018. Deep learning in spiking neural networks. *Neural Networks*.

Thakur, C. S., J. L. Molin, G. Cauwenberghs, G. Indiveri, K. Kumar, N. Qiao, J. Schemmel, R. Wang, E. Chicca, J. Olson Hasler, J.-s. Seo, S. Yu, Y. Cao, A. van Schaik, and R. Etienne-Cummings
2018. Large-scale neuromorphic spiking array processors: A quest to mimic the brain. *Frontiers in Neuroscience*, 12:891.

Thalmeier, D., M. Uhlmann, H. J. Kappen, and R.-M. Memmesheimer
2016. Learning universal computations with spikes. *PLOS Computational Biology*, 12(6):e1004895.

Thorpe, S. and J. Gautrais
1998. Rank order coding. In *Computational Neuroscience*, Pp. 113–118. Springer.

Tiganj, Z., N. Cruzado, and M. Howard
2018. Constructing neural-level models of behavior in working memory tasks.

Tiganj, Z., M. E. Hasselmo, and M. W. Howard
2015. A simple biophysically plausible model for long time constants in single neurons. *Hippocampus*, 25(1):27–37.

Tiganj, Z., M. W. Jung, J. Kim, and M. W. Howard
2016. Sequential firing codes for time in rodent medial prefrontal cortex. *Cerebral Cortex*.

Tiganj, Z., K. H. Shankar, and M. W. Howard
2017. Neural scale-invariant time-frequency decomposition for detection of spatiotemporal features. *Submitted*.

Torrejon, J., M. Riou, F. A. Araujo, S. Tsunegi, G. Khalsa, D. Querlioz, P. Bortolotti, V. Cros, K. Yakushiji, A. Fukushima, et al.
2017. Neuromorphic computing with nanoscale spintronic oscillators. *Nature*, 547(7664):428.

Tripp, B. and C. Eliasmith
2006. Neural populations can induce reliable postsynaptic currents without observable spike rate changes or precise spike timing. *Cerebral Cortex*, 17(8):1830–1840.

Tripp, B. and C. Eliasmith
2010. Population models of temporal differentiation. *Neural Computation*, 22:621–659.

Trujillo, O.
2014. A spiking neural model of episodic memory encoding and replay in hippocampus. Masters thesis, University of Waterloo, Waterloo, Ontario.

Trujillo, O. and C. Eliasmith
2014. A spiking-neuron model of memory encoding and replay in hippocampus. In *BMC Neuroscience*, P. 166. Organization for Computational Neurosciences.

Turing, A. M.
1938. On computable numbers, with an application to the Entscheidungsproblem. A correction. *Proceedings of the London Mathematical Society*, 2(1):544–546.

University of Manchester
2012. Research groups: APT - Advanced Processor Technologies - SpiNNaker project. http://apt.cs.manchester.ac.uk/projects/SpiNNaker/project/. Accessed: 2018-11-26.

Usher, M. and J. L. McClelland
2001. The time course of perceptual choice: The leaky, competing accumulator model. *Psychological review*, 108(3):550.

Vajta, M.
2000. Some remarks on Padé-approximations. In *3rd TEMPUS-INTCOM Symposium*, Veszprém, Hungary.

van Albada, S. J., A. G. Rowley, J. Senk, M. Hopkins, M. Schmidt, A. B. Stokes, D. R. Lester, M. Diesmann, and S. B. Furber
2018. Performance comparison of the digital neuromorphic hardware SpiNNaker and the neural network simulation software NEST for a full-scale cortical microcircuit model. *Frontiers in Neuroscience*, 12.

van De Burgt, Y., A. Melianas, S. T. Keene, G. Malliaras, and A. Salleo
2018. Organic electronics for neuromorphic computing. *Nature Electronics*, P. 1.

Vinyals, O., A. Toshev, S. Bengio, and D. Erhan
2015. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Pp. 3156–3164.

Voelker, A. R.
2014. Implementing hill climbing with a spiking neural network. Technical report, Centre for Theoretical Neuroscience, Waterloo, ON.

Voelker, A. R.
2015a. Computing with temporal representations using recurrently connected populations of spiking neurons. In *Connecting Network Architecture and Network Computation*. Banff International Research Station for Mathematical Innovation and Discovery.

Voelker, A. R.
2015b. A solution to the dynamics of the Prescribed Error Sensitivity learning rule. Technical report, Centre for Theoretical Neuroscience, Waterloo, ON.

Voelker, A. R.
2019. NengoLib – Additional extensions and tools for modelling dynamical systems in Nengo. https://github.com/arvoelke/nengolib/. Accessed: 2019-03-09.

Voelker, A. R., B. V. Benjamin, T. C. Stewart, K. Boahen, and C. Eliasmith
2017a. Extending the Neural Engineering Framework for nonideal silicon synapses. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, Baltimore, MD. IEEE.

Voelker, A. R., E. Crawford, and C. Eliasmith
2014. Learning large-scale heteroassociative memories in spiking neurons. In *Unconventional Computation and Natural Computation*, S. K. Oscar H. Ibarra, Lila Kari, ed., London, Ontario. Springer International Publishing.

Voelker, A. R. and C. Eliasmith
2014. Controlling the Semantic Pointer Architecture with deterministic automata and adaptive symbolic associations. Technical report, Centre for Theoretical Neuroscience, Waterloo, ON.

Voelker, A. R. and C. Eliasmith
2016. Methods and systems for implementing dynamic neural networks. *US Patent App. 15/243,223 (patent pending)*.

Voelker, A. R. and C. Eliasmith
2017a. Analysis of oscillatory weight changes from online learning with filtered spiking feedback. Technical report, Centre for Theoretical Neuroscience, Waterloo, ON.

Voelker, A. R. and C. Eliasmith
2017b. Methods for applying the Neural Engineering Framework to neuromorphic hardware. *arXiv preprint arXiv:1708.08133*.

Voelker, A. R. and C. Eliasmith
2018. Improving spiking dynamical networks: Accurate delays, higher-order synapses, and time cells. *Neural Computation*, 30(3):569–609.

Voelker, A. R. and C. Eliasmith
2019. Dynamical systems in spiking neuromorphic hardware. *US Patent App. 62/814,767 (patent pending)*.

Voelker, A. R., J. Gosmann, and T. C. Stewart
2017b. Efficiently sampling vectors and coordinates from the n-sphere and n-ball. Technical report, Centre for Theoretical Neuroscience, Waterloo, ON.

Von Neumann, J.
1958. The computer and the brain. *Yale Univ. in the Animal and the Machine.*

Waernberg, E. and A. Kumar
2017. Low dimensional activity in spiking neuronal networks. *bioRxiv*.

Wallace, E., H. R. Maei, and P. E. Latham
2013. Randomly connected networks have short temporal memory. *Neural Computation*, 25(6):1408–1439.

Wang, R.
2013. *Neuromorphic implementations of polychronous spiking neural networks*. PhD thesis, School of Western Sydney.

Wang, R., T. J. Hamilton, J. Tapson, and A. van Schaik
2014. A compact neural core for digital implementation of the Neural Engineering Framework. In *Biomedical Circuits and Systems Conference (BioCAS)*, Pp. 548–551. IEEE.

Wang, R., C. S. Thakur, G. Cohen, T. J. Hamilton, J. Tapson, and A. van Schaik
2017. A neuromorphic hardware architecture using the Neural Engineering Framework for pattern recognition. *IEEE Transactions on Biomedical Circuits and Systems*, 11(3):574–584.

Wang, R. and A. van Schaik
2018. Breaking Liebig's law: An advanced multipurpose neuromorphic engine. *Frontiers in Neuroscience*, 12.

Waskom, M., O. Botvinnik, P. Hobson, J. Warmenhoven, J. B. Cole, Y. Halchenko, J. Vanderplas, S. Hoyer, S. Villalba, E. Quintero, and et al.
2015. seaborn: v0.6.0 (june 2015).

Werbos, P.
1974. Beyond regression: New tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University.*

Werbos, P. J.
1990. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

Weston, J., S. Chopra, and A. Bordes
2014. Memory networks. *arXiv preprint arXiv:1410.3916.*

White, O. L., D. D. Lee, and H. Sompolinsky
2004. Short-term memory in orthogonal neural networks. *Physical Review Letters*, 92(14):148102.

Wijekoon, J. H. and P. Dudek
2012. VLSI circuits implementing computational models of neocortical circuits. *Journal of Neuroscience Methods*, 210(1):93–109.

Wiklicky, H.
1994. On the non-existence of a universal learning algorithm for recurrent neural networks. In *Advances in Neural Information Processing Systems*, Pp. 431–436.

Willshaw, D. J., O. P. Buneman, and L. H. C. Higgins
1969. Non-holographic associative memory. *Nature*, 222:960–962.

Wilson, M. A. and J. M. Bower
1989. The simulation of large-scale neural networks. In *Methods in Neuronal Modeling*, Pp. 291–333. MIT Press.

Wolpert, D. H.
1996. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390.

Woodhouse, I. H.
2001. The ratio of the arithmetic to the geometric mean: A cross-entropy interpretation. *IEEE Transactions on Geoscience and Remote Sensing*, 39(1):188–189.

Wu, Y., M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al.
2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Yan, Y., D. Kappel, F. Neumaerker, J. Partzsch, B. Vogginger, S. Hoeppner, S. Furber, W. Maass, R. Legenstein, and C. Mayr
2019. Efficient reward-based structural plasticity on a SpiNNaker 2 prototype. *arXiv preprint arXiv:1903.08500*.

Yoon, Y. C.
2017. LIF and simplified SRM neurons encode signals into spikes via a form of asynchronous pulse sigma–delta modulation. *IEEE Transactions on Neural Networks and Learning Systems*, 28(5):1192–1205.

Yousefzadeh, A., S. Hoseini, P. Holanda, S. Leroux, T. Werner, T. Serrano-Gotarredona, B. Linares Barranco, B. Dhoedt, and P. Simoens
2019. Conversion of synchronous artificial neural network to asynchronous spiking neural network using sigma-delta quantization.

Zhao, B. and J. Ravichandran
2019. Low-power microwave relaxation oscillators based on phase-change oxides for neuromorphic computing. *Physical Review Applied*, 11(1):014020.

Zheng, N. and P. Mazumder
2018. A low-power hardware architecture for on-line supervised learning in multi-layer spiking neural networks. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, Pp. 1–5. IEEE.

Zhu, X., P. Sobihani, and H. Guo
2015. Long short-term memory over recursive structures. In *International Conference on Machine Learning*, Pp. 1604–1612.

Zumbahlen, H. et al.
2011. *Linear circuit design handbook*. Newnes.