Programming Neuromorphics Using the Neural Engineering Framework

Aaron R. Voelker and Chris Eliasmith

Contents

Abstract

As neuromorphic hardware begins to emerge as a viable target platform for artificial intelligence (AI) applications, there is a need for tools and software that can effectively compile a variety of AI models onto such hardware. Nengo (http://nengo.ai) is an ecosystem of software designed to fill this need with a suite of tools for creating, training, deploying, and visualizing neural networks for various hardware backends, including CPUs, GPUs, FPGAs, microcontrollers, and neuromorphic hardware. While backpropagation-based methods are pow-

A. R. Voelker (✉) · C. Eliasmith
Centre for Theoretical Neuroscience, University of Waterloo, Applied Brain Research Inc., Waterloo, ON, Canada
e-mail: arvoelke@uwaterloo.ca; arvoelke@gmail.com; celiasmith@uwaterloo.ca

N. V. Thakor (ed.), *Handbook of Neuroengineering*,
https://doi.org/10.1007/978-981-15-2848-4_115-1

erful and fully supported in Nengo, there is also a need for frameworks that are capable of efficiently mapping dynamical systems onto such hardware while best utilizing its computational resources. The neural engineering framework (NEF) is one such method that is supported by Nengo. Most prominently, Nengo and the NEF have been used to engineer the world's largest functional model of the human brain. In addition, as a particularly efficient approach to training neural networks for neuromorphics, the NEF has been ported to several neuromorphic platforms. In this chapter, we discuss the mathematical foundations of the NEF and a number of its extensions and review several recent applications that use Nengo to build models for neuromorphic hardware. We focus in-depth on a particular class of dynamic neural networks, Legendre Memory Units (LMUs), which have demonstrated advantages over state-of-the-art approaches in deep learning with respect to energy efficiency, training time, and accuracy.

Keywords

Neural engineering framework · Nengo · Spiking neural networks · Dynamical systems · Legendre Memory Units · Deep learning · Neuromorphic hardware · Loihi · Braindrop

1 Introduction

The neural engineering framework (NEF) was proposed around the turn of the century as a way to understand how spiking neural networks (SNNs) compute [29, 31]. After about a decade of use for building biologically focused models of various brain functions, it began to be adopted as a way to program neuromorphic hardware for real-time control [24] and to implement dynamical systems [39]. Recently, it has seen widespread use on a variety of neuromorphic platforms, including Neurogrid [13], Braindrop [67], SpiNNaker [65], TrueNorth [35], FPGAs [63, 99, 100], Loihi [9, 25], and prototype VLSI circuits [14]. The common denominator across all of these deployments is that of processing time-varying data using spiking neurons, with the overarching goal of scalable, real-time, low-power computation.

In this chapter, we provide an introduction to the NEF and a recently extended example of its use to develop a novel recurrent neural network (RNN), the Legendre Memory Unit (LMU).We discuss why the NEF is suitable for neuromorphic hardware and show examples of circuits, including the LMU, running on two different neuromorphic platforms. We then compare the NEF to other methods of building RNNs and provide quantitative results showing it is far more accurate and space-efficient than its competitors. We also address some standard criticisms of the NEF.

Overall, the NEF has been focused on building functional brain models or "neural applications," be they in simulation or in hardware. As a result, throughout our discussion, we mention how the neural simulator Nengo can be used to construct such networks and deploy them onto neuromorphic hardware. Nengo [5] is a

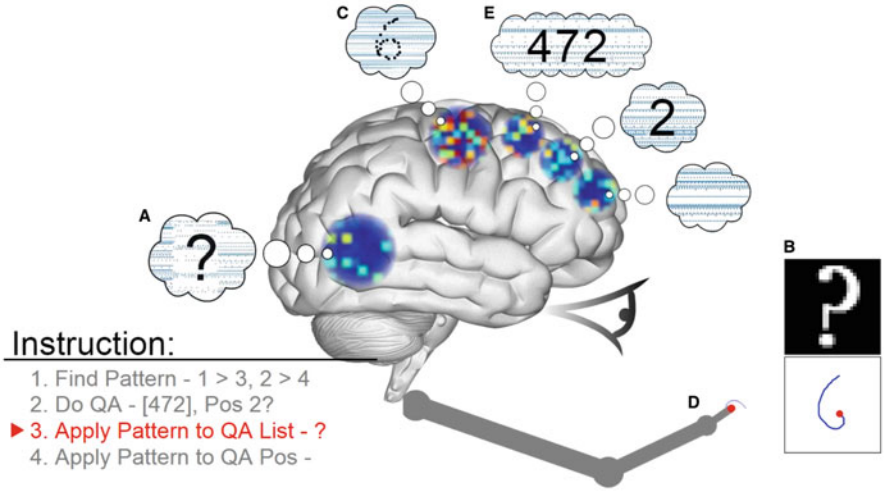


Fig. 1 Spaun 2.0, the world's largest functional brain model, performing an instruction following task. The model is using (**a**) a spiking deep network to process (**b**) visual inputs, (**c**) a spiking adaptive controller to control (**d**) its arm to write responses, (**e**) spiking high-dimensional attractor networks to remember responses and stimuli, along with spiking action selection, learning, and syntactic pattern inference components to follow dynamic instructions on the fly [12]

general-purpose neural development environment that supports many varieties of networks, spiking and non-spiking, many varieties of optimization, backpropagation and others, and many scales of systems, from single cells to millions of neurons. In fact, Nengo and the NEF were used to build and run the world's largest functional brain model, Spaun [33], the latest version of which (see Fig. 1) has over six million neurons and 20 billion connections [12]. Nengo is unique in supporting the NEF natively, which is the main reason we refer to it throughout.

We refer the reader to Sharma et al. [76] and the Nengo documentation (http://nengo.ai) for extensive and up-to-date tutorials that illustrate the use of Nengo for cognitive modelling, deep learning, and neuromorphic hardware. This chapter draws extensively from Voelker [88], which we regularly refer to for additional proofs, formalization, and discussions surrounding the NEF and its use as a framework for programming neuromorphics.

2 The Neural Engineering Framework (NEF)

Fundamentally, the NEF is a means of building, or 'training' recurrent neural networks (RNNs). This is of interest because the RNN is the most computationally powerful kind of neural network that we know how to physically implement. Standard artificial RNNs serve as a universal approximator to any finite-dimensional,

causal, discrete-time, dynamical system [75].[1] 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 neural 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, and, more generally, carry out arbitrary computations over time.

2.1 Principles of the NEF

In this section, we provide a detailed overview of the original formulation of the NEF that has been adapted from Voelker and Eliasmith [92].

The NEF proposes a general approach to model dynamical systems in artificial networks of spiking (and non-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," (see Fig. 1) that is capable of performing perceptual, motor, and cognitive tasks [12, 33]. This model incorporates many kinds of observed neural dynamics, including self-generated oscillations, sustained activity, and point attractor dynamics.

As mentioned in the introduction, the flexibility of the NEF has led to it being deployed on mixed-analog-digital neuromorphic chips as well as digital architectures (see Sect. 4.1). Consequently, the NEF provides a practical method for programming neuromorphic hardware, helping the field deliver on its promise of an energy-efficient computing platform that emulates core principles of nervous systems, namely, analog computation via neurons and synapses, and digital communication via overlapping dendritic trees and hierarchical axonal arbors [10].

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. This is made efficient through three important steps:

1. The 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 that determines each weight matrix is recast as a batched least-squares problem that may be solved offline without needing to explicitly simulate the network over time.
3. The dynamics of the synapses are leveraged as temporal bases for the overall system.

[1] There is a similar theorem for continuous time [38].

In the context of software implementation, these three steps lead to procedures for training and simulating that are efficiently implemented – scaling as $\mathcal{O}(n)$ in both time and space – by Nengo [5]. In the context of deep learning, backpropagation is a useful tool for fine-tuning the performance of a network built using the NEF and to learn functions that span multiple layers in such networks. Thus, NengoDL [71] bidirectionally integrates Nengo with TensorFlow [1] – Google's popular toolkit for deep learning. With this extension, Nengo models can be used within TensorFlow and vice-versa. In particular, NengoDL can be used to apply TensorFlow's backpropagation algorithm to any existing Nengo model, to process batches of inputs in parallel, and to deploy models onto any hardware that is supported by TensorFlow, including CPUs, GPUs, tensor processing units (TPUs), and microcontrollers. Likewise, TensorFlow models can be converted using NengoDL to run on neuromorphic hardware [e.g., 25].

The top-down approach taken by the NEF 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 [31, pp. 185–217]. Optimized architectures can therefore be analytically derived for the case of specific functions, such as multiplication [44]. 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 [92]. 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 [27, 28, 32].

Conceptually, the NEF consists of three mathematical principles used to describe neural computation:

1. Representation – The use of heterogeneous populations of neurons to encode distributed representations of some time-varying state.
2. Transformation – The use of connection weights to decode nonlinear functions of this state.
3. Dynamics – The use of synapse models to solve differential equations.

These three principles are summarized in Fig. 2 and explicated throughout the next three sections.

The NEF's primary strengths reside in providing a well-understood and efficient means of engineering spiking neural models, and programming neuromorphic hardware, to approximate computations with precision that scales as $\mathcal{O}\left(\sqrt{n}\right)$ in the spiking case and $\mathcal{O}(n)$ in the non-spiking case [10, 31]. 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 [89, 91].

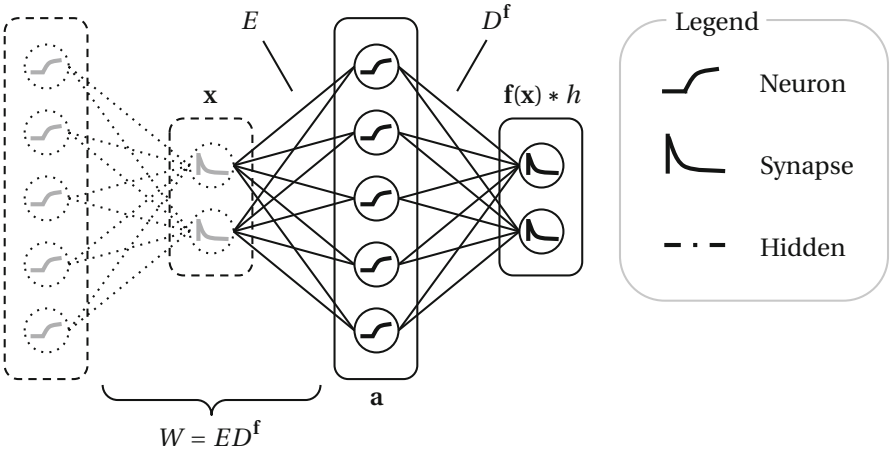


Fig. 2 Three principles of the neural engineering framework (NEF), illustrated with $n = 5$ neurons and $q = 2$ dimensions. (1) Representation: Heterogeneous populations of neurons are used to encode (E) distributed representations of some time-varying state ($\mathbf{x}$). (2) Transformation: Connection weights (W) are used to decode ($D^{\mathbf{f}}$) nonlinear functions ($\mathbf{f}$) of this state from the neural activity ($\mathbf{a}$). (3) Dynamics: Synapse models (h) are used to solve differential equations, specifically, $\mathbf{f}(\mathbf{x}) * h = \mathbf{x} \iff \tau\dot{\mathbf{x}} + \mathbf{x} = \mathbf{f}(\mathbf{x})$. Hidden units correspond to those "hidden" from the optimization problem of solving for decoders

2.1.1 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:

$$\begin{aligned} cs_i(t) &= \alpha_i \langle \mathbf{e}_i, \mathbf{x}(t) \rangle + \beta_i \\ a_i(t) &= G_i\left[s_i(t)\right] \\ &= \sum_m \delta\left(t - t_{i,m}\right), \end{aligned} \tag{1}$$

where $s_i(t)$ is the input current to the ith neuron, $\langle \cdot, \cdot \rangle$ denotes a dot product, $a_i(t)$ is the neural activity generated by the ith 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 [53]. This model is considered to strike a convenient balance between simplicity and biological realism [31, 88, 89], although the NEF can support more biologically detailed neuron models [26, 78]. Likewise, neurons driven by nonlinear conductances and dendritic interactions, rather than pure current sources, can be leveraged by the NEF [79, 82].

The heterogeneous parameters that define the encoding, $\{(\mathbf{e}_i, \alpha_i, \beta_i) : i = 1 \ldots n\}$, 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 tuning curves, preferred directions, firing rates, sparsity, etc.; see, for instance, Friedl et al. [37]), predetermined using prior knowledge of the desired transformation [44] or trained via backpropagation through time [46–48, 71]. In essence, Eq. 1 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. 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:

$$h(t) = \frac{1}{\tau} e^{-\frac{t}{\tau}},\ t \geq 0. \tag{2}$$

This low-pass synapse is also known as a "leaky integrator" and is the conventional choice of synapse in the NEF. As discussed in Sect. 2.1, we can 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 $D = [\mathbf{d}_1, \ldots, \mathbf{d}_n] \in \mathbb{R}^{q \times n}$ 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:

$$\begin{aligned} c\,(\mathbf{x} * h)\,(t) &\approx \sum_{i=1}^{n} (a_i * h)\,(t)\mathbf{d}_i \\ &= \sum_{i=1}^{n} \sum_{m} h\left(t - t_{i,m}\right) \mathbf{d}_i\,, \end{aligned} \tag{3}$$

where * is the convolution operator used to apply the synaptic filter.[2] Eq. 3 takes a linear combination of the filtered activities, in order to recover a filtered version of the encoded signal. 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.

2.1.2 Principle 2: Transformation

The second principle of the NEF addresses the issue of computing transformations of the represented signal. The encoding remains defined by Eq. 1. However, we now decode a filtered version of some function, $\mathbf{f}$: $\mathbb{R}^q \to \mathbb{R}^q$, by applying an alternate matrix of decoders to the same filtered spike trains:

$$\begin{aligned} D^{\mathbf{f}} &= \left[\mathbf{d}_1^{\mathbf{f}}, \ldots, \mathbf{d}_n^{\mathbf{f}}\right] \in \mathbb{R}^{q \times n} \\ (\mathbf{f}(\mathbf{x}) * h)(t) &\approx \sum_{i=1}^{n} (a_i * h)(t) \mathbf{d}_i^{\mathbf{f}} \\ &= \sum_{i=1}^{n} \sum_{m} h\left(t - t_{i,m}\right) \mathbf{d}_i^{\mathbf{f}}. \end{aligned} \tag{4}$$

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, $S = \{\mathbf{x}(t)\colon t \geq 0\}$, 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 ith neuron under the constant input $\mathbf{v} \in S$:

$$r_i(\mathbf{v}) = \lim_{t \to \infty} \frac{1}{t} \int_0^t a_i\left(t'\right) dt'. \tag{5}$$

For nonadaptive neuron models, Eq. 5 reduces to encoding $\mathbf{v}$ using a rate model.

For adaptive neuron models, other definitions for $r_i(\mathbf{v})$ may be considered, but we limit our discussion here to the (nonadaptive) 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 Eq. 4 becomes the following:

[2]By linearity of convolution, it does not matter whether the filter is applied before or after the decoding or any subsequent encoding. For efficiency reasons, it is often applied in the lower-dimensional (i.e., decoded) space. What is most efficient depends on the hardware and the sparsity of neural activity relative to the integration timescale.

$$D^{\mathbf{f}} = \underset{D \in \mathbb{R}^{q \times n}}{\arg\min} \int_S \left\| \mathbf{f}(\mathbf{v}) - \sum_{i=1}^{n} (r_i(\mathbf{v}) + \eta)\, \mathbf{d}_i \right\|_2^2 d^q \mathbf{v}. \tag{6}$$

Note that this optimization only depends on $r_i(\mathbf{v})$ for $\mathbf{v} \in S$, as opposed to depending on the signal $\mathbf{x}(t)$. Since we have replaced the optimization problem involving spikes (Eq. 4) 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 [94] and applying a standard regularized least-squares solver to the sampled data [5]. Nengo also supports alternative decoder solvers that optimize variants of Eq. 6 [e.g., 37, 51] and randomized singular-value- decomposition (SVD) solvers that take $\mathcal{O}(mn)$ time and space, where m is the number of samples from S – but we do not use them here.

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

$$\mathbf{x}(t) = \mathbf{v} \Rightarrow \sum_{i=1}^{n} (a_i * h)(t) \mathbf{d}_i^{\mathbf{f}} \approx \sum_{i=1}^{n} ((r_i(\mathbf{v}) + \eta) * h)(t) \mathbf{d}_i^{\mathbf{f}}. \tag{7}$$

In Sect. 5.3, we discuss how our adoption of Eq. 7 has led many to mischaracterize the NEF as using a "rate code," which comes with significant baggage. 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 [88, Sect. 3.2.1].

It remains to map these encoding and decoding parameters onto a neural architecture. Equations 1 and 4 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 jth presynaptic neuron and the ith postsynaptic neuron. Specifically, the weight matrix $W = [\omega_{ij}] \in \mathbb{R}^{n \times n}$, which maps activities from the jth presynaptic neuron to the ith postsynaptic neuron (disregarding the gain and bias, for notational simplicity), is given by

$$W = E D^{\mathbf{f}}. \tag{8}$$

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 (Eq. 4) and then encoding (Eq. 1) 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. 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 [79, 82].

The crucial difference between the factorized and non-factorized forms is that it takes $\mathcal{O}(qn)$ operations per simulation time-step to implement this dot product in the factored form of Eq. 8, as opposed to $\mathcal{O}(n^2)$ operations for a full weight matrix. Since q is typically held constant, this yields a factor $\mathcal{O}(n)$ improvement in simulation time. Similarly, this factorization yields an $\mathcal{O}(n)$ reduction in memory, which significantly improves the scaling of neuromorphics [65].

2.1.3 Principle 3: Dynamics

Principle 3 is a method of harnessing the dynamics of the synapse model for network-level information processing, in effect by solving differential equations over time. For ease of explanation, we begin by focusing our discussion of NEF dynamics on the neural implementation of continuous, linear time-invariant (LTI) systems (see Fig. 3):

$$\begin{aligned} \dot{\mathbf{x}}(t) &= A\mathbf{x}(t) + B\mathbf{u}(t) \\ \mathbf{y}(t) &= C\mathbf{x}(t) + D\mathbf{u}(t), \end{aligned} \tag{9}$$

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 [11]. This form of an LTI system is commonly referred to as the *state-space model*.

For LTI systems, the *dynamical primitive* – that is, the source of the dynamics – is the integrator. However, in the model that we consider, the dynamical primitive at our disposal is the *leaky* integrator, given by the synapse (repeating Eq. 2, for clarity):

$$h(t) = \frac{1}{\tau} e^{-\frac{t}{\tau}} = \mathcal{L}^{-1}\left\{\frac{1}{\tau s + 1}\right\}, \tag{10}$$

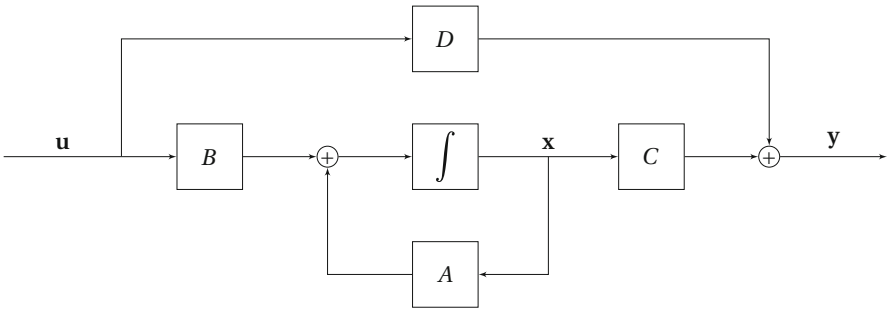


Fig. 3 Block diagram for an LTI system. The integrator is driven by the signal $\dot{\mathbf{x}}(t)$. (Reproduced from Voelker and Eliasmith [92, Fig. 1])

where $\mathscr{L}^{-1}\{\cdot\}$ denotes the inverse Laplace transform and the latter representation is referred to as a transfer function. Our approach is to represent the state vector $\mathbf{x}(t)$ in a population of spiking neurons (Principle 1; Eq. 1), such that *this same* vector is obtained by filtering an appropriate transformation with a leaky integrator (Principle 2; Eq. 4). Thus, the goal of Principle 3 is to determine the transformations required to implement Eq. 9, given that the required $\mathbf{x}(t)$ is obtained by some convolution with a leaky integrator, rather than the perfect integrator assumed by standard LTI formulations.

Principle 3 states that in order to implement Eq. 9 in a population of neurons that represents $\mathbf{x}(t)$, we must compensate for the effect of "replacing" the integrator with a leaky integrator 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 pass-through transformation $D\ \mathbf{u}(t)$ [31, pp. 221–225]. The resulting model is summarized in Fig. 4.

The correctness of this "mapping" procedure relies on three assumptions:

1. The synapse model is Eq. 10.
2. The network is simulated in continuous time (or the discrete time-step is sufficiently small).
3. The necessary representations and transformations are sufficiently accurate, such that the approximation error from Eq. 4 is negligible.

In the next section, we discuss recent work that extends Principle 3 to relax the first and second assumptions, and in Voelker [88, Sect. 3.2.1], it is shown how the third assumption holds.

Given the ability of Principle 2 to compute nonlinear functions (i.e., Eq. 4), Principle 3 also naturally generalizes nonlinear dynamical systems.

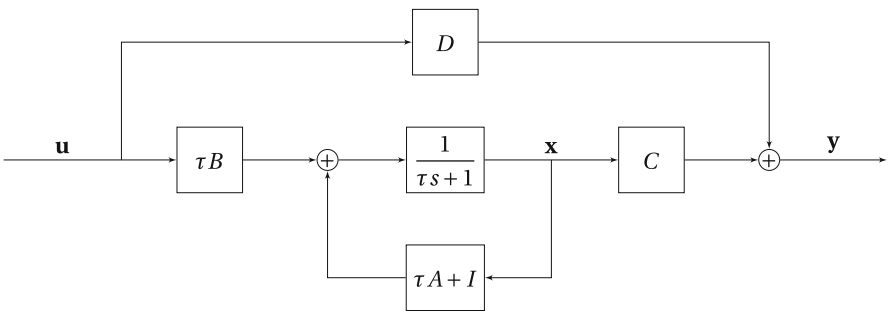


Fig. 4 Block diagram for an LTI system, equivalent to Fig. 3, with the integrator replaced by a first-order low-pass filter. The low pass is driven by the signal $\tau\dot{\mathbf{x}}(t) + \mathbf{x}(t)$ to ensure that it implements the same system as in Fig. 3. The state $\mathbf{x}(t)$ may be represented and transformed within a recurrently connected population of neurons by using Principles 1 and 2 to implement the operations of each block. (Reproduced from Voelker and Eliasmith [92, Fig. 2])

2.1.4 An Extension to the NEF: Generalizing Principle 3

Rapid growth in the field of neuromorphic engineering over the past 30 years has accelerated the production of a large variety of neuromorphic architectures, each with distinct constraints and trade-offs. Among these considerations are issues involving discretization (in time), quantization (limited bit-precision for 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. Recently, new extensions to the NEF have bolstered it with a variety of additional methods and techniques for analyzing and leveraging spiking dynamical computations [89–93]. In particular, the assumption of a specific synapse model, $h(t)$, has been lifted [88], Theorems 5.1.3 and 5.1.4]. This addresses many of the differences that emerge between real brains, the idealizations made by the NEF, and the realities imposed by neuromorphic hardware.

Specifically, Principle 3 now covers the general class of all linear continuous-time synapse models of the form:

$$H(s) = \frac{1 + \sum_{i=1}^{p} b_i s^i}{\sum_{i=0}^{q} c_i s^i}, \tag{11}$$

as well as the analogous class of discrete-time synapse models, as used in digital hardware. This encompasses the class of nearly all linear synapses, including all that we are aware of in the literature [e.g., 4, 20, 70, 101], as well as the higher-order synapses in neuromorphic hardware [89]. Nonlinear synapse models, such as conductance-based synapses [e.g., 23, Eq. 6], can augment the computational power of neural networks and are the active subject of study in the NEF [79, 81, 82].

3 Extended Example: The Legendre Memory Unit

A particularly important dynamical system that has not been discussed before in the NEF literature is the pure *continuous-time delay* line. However, the NEF provides perhaps the most natural set of tools to explicitly address the challenges of this dynamical system. In order to implement such a system, one must represent a *sliding 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 section, we discuss a novel, optimal, low-dimensional, linear approximation to a continuous-time delay and then realize this *Legendre Memory Unit* [LMU; 95] using the NEF in a recurrent spiking network. We then investigate its computational properties, showing that the resulting network implements a clearly 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 repre-

sentation, with a level of accuracy that depends on the input frequency, chosen dimensionality (i.e., the order of the approximation), and particular synapse model. The activity of the neurons in the LMU also bears a striking similarity to time cells in the neuroscience literature [19, 92]. To our knowledge, this is the first network to demonstrate that such a temporal code may be accurately realized by a spiking dynamical network.

Reservoir computing approaches, such as liquid state machines [LSMs; 61] and echo state networks [ESNs; 49], may be used to approximate a delay line. However, since these networks use randomly chosen feedback weights, we show in Sect. 5.2 that they do so with relatively poor accuracy despite extensive hyper-optimization. Such networks instead represent a random variety of nonlinear memory traces [59]. We also find that the delay line is a difficult function for FORCE [83] networks to learn (Sect. 5.2); re-encoding the delayed output as a teaching signal ends up "confusing" the network. The method that we discuss 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 can use our extensions discussed in Sect. 2.1, which improves our understanding of the relationship between synapse models and network-level computations (see [88] for details).

In other work, it has been shown that our network, when expressed as an RNN cell (without spikes), outperforms equivalently sized stacked long short-term memories [LSTMs; 45] on computing long delays, predicting the Mackey-Glass dataset – a difficult chaotic time-series benchmark – in training time, inference time, and test accuracy [88], and sets a new state-of-the-art result for RNNs on the permuted sequential MNIST benchmark [95].

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

$$y(t) = (u * \delta_\theta)(t) = u(t - \theta), \ \theta > 0, \tag{12}$$

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 nontrivial to implement using continuous-time spiking dynamical primitives. Specifically, Eq. 12 requires that we maintain a *sliding 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 $0 \leq \theta' \leq \theta$. 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 [74]. Fourth, a delay line can be coupled with a single nonlinearity to construct a network displaying many of the same benefits as reservoir computing [3].

3.1 Defining the LMU

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 [15]. To resolve this problem, we approximate $u(t)$ as a low-frequency signal or, equivalently, approximate Eq. 12 as a low-dimensional system expanded about the zeroth frequency in the *Laplace domain*. Our choice of a zero-frequency approximation is informed by an analysis [88], Sect. 3.1.6], which suggests that neural systems require energy that grows linearly in the representational frequency.

Here, we do not repeat the full derivation of the LMU [88, 92], but rather present the two main results. First, we show that the following LTI is the optimal realization of a delay of dimensionality q. Second, we relate the dynamics of this system to the Legendre polynomials up to degree $q-1$. The dynamical system is described by the following equations:

$$\begin{aligned}
\theta\dot{\mathbf{x}}(t) &= A\mathbf{x}(t) + Bu(t) \\
y(t) &= C\mathbf{x}(t) \\
A = [a]_{i\ j} \in \mathbb{R}^{q\times q}, \quad a_{i\ j} &= (2i+1)\begin{cases} -1 & i < j \\ (-1)^{i-j+1} & i \geq j \end{cases} \\
B = [b]_i \in \mathbb{R}^{q\times 1},\ b_i &= (2i+1)\,(-1)^i,\ i, j \in [0, q-1] \\
C = [c]_i \in \mathbb{R}^{1\times q},\ c_i &= 1
\end{aligned} \tag{13}$$

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 Eq. 13 onto a spiking dynamical network to accurately implement an optimal low-frequency approximation of the delay. Since this system depends only on θ by the rate at which $\mathbf{x}(t)$ is integrated, we may control the length of the delay by adjusting the gain on the integration time constant [19]. The NEF can be used to build such controlled dynamical systems, without introducing multiplicative dendritic interactions or implausible on-the-fly connection weight scaling [30].

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 six-dimensional vector space (see Fig. 5). The connections between neurons are determined by applying Principle 3 (Sect. 2.1) to the state-space model derived above (Eq. 13,

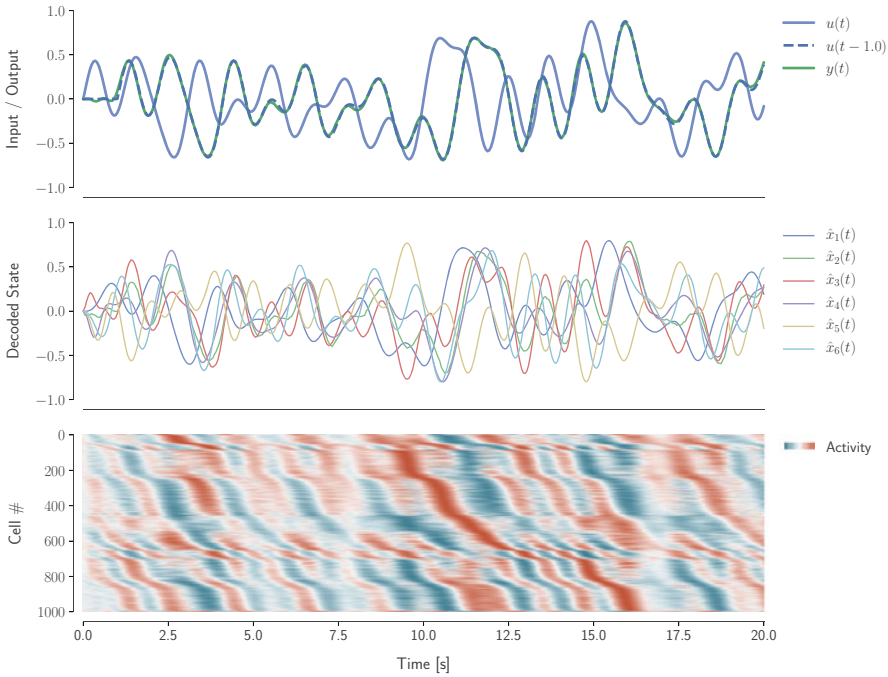


Fig. 5 Delay of 1 s implemented by applying standard Principle 3 to Eq. 13 using $q = 6$, dt = 1 ms, 1,000 spiking LIF neurons, and a low-pass synapse with $\tau = 0.1$ s. The input signal is white noise with a cutoff frequency of 1 Hz. The plotted spikes are filtered with the same $\tau = 0.1$ s and encoded with respect to 1,000 encoders sampled uniformly from the surface of the hypersphere (sorted by time to peak activation, colored from lowest activity [blue] to highest activity [red]). (Reproduced from Voelker and Eliasmith [92, Fig. 3], reprinted courtesy of The MIT Press)

$q = 6$) via the Padé approximants of the delay line. The normalized root-mean-squared error (NRMSE; normalized so that 100% 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, the network accuracy improves as dt approaches zero due to the continuous-time assumption mentioned in Sect. 2.1.

When dt is nonzero (as is the case in digital hardware) or sufficiently large relative to θ, the system of Eq. 13 can be discretized (e.g., using zero-order hold or Euler's method). To map the discretized LTI system onto a neural network, we account for the discretization using the NEF extensions discussed in Sect. 2.1.

3.1.1 Temporal Representation

Although the LMU has its dynamics optimized for a single delay $\theta > 0$, we may still accurately decode any delay $0 \leq \theta' \leq \theta$ 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 sliding window (i.e., history) of length θ. To compute these other delays, we must approximate $e^{-\theta' s}$ with a transfer function:

$$F_{\theta\to\theta'}(s) := \frac{\mathcal{C}\left(s;\theta,\theta'\right)}{\mathcal{D}\left(s;\theta\right)}, \tag{14}$$

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.

We have shown elsewhere that different decodings require different linear output transformations (C) for each θ', as summarized by the following formulas [88]:

$$\begin{aligned} u\left(t-\theta'\right) &\approx \sum_{i=0}^{q-1} \mathcal{P}_i\left(\tfrac{\theta'}{\theta}\right) x_i(t),\ 0 \le \theta' \le \theta \\ \mathcal{P}_i(r) &= (-1)^i \sum_{j=0}^{i} \binom{i}{j}\binom{i+j}{j}(-r)^j, \end{aligned} \tag{15}$$

where $\mathcal{P}_i\ (r)$ is the ith *shifted Legendre polynomial* [56, 73].

In other words, the q-dimensional state vector of the LMU represents a sliding window of length θ. That is, a single LMU with some fixed $\theta > 0$ may be used to accurately decode any delay of length θ' ($0 \le \theta' \le \theta$) by taking a linear transformation of its state vector according to the coefficients of Eq. 15. Functions of $\mathbf{x}(t)$ then correspond to computations across the input window, orthogonally projected onto the Legendre polynomials – hence the name Legendre Memory Unit.

The significance of the Legendre polynomials lies in them being the unique set of orthogonal polynomials (up to a normalization factor) across any finite interval. In this case, the finite interval corresponds to the sliding window of time, and the state of the system corresponds to the linear combination of polynomials that approximate the said window. Hence, the polynomial basis forms a compact representation of time that is determined online as the input signal is streamed into the network. We are not aware of any other class of networks that have this unique property. This is understood as being what has enabled the LMU to set a new state-of-the-art result on the permuted sequential MNIST benchmark [95] – a challenging stress test of an RNN's ability to efficiently compute across a time-varying signal representing the scrambled pixels of a handwritten digit.

In Fig. 6, we take different linear transformations of the same state vector by evaluating Eq. 15 at various delays between 0 and θ, to decode the window of input from the state of the system.[3] This demonstrates that the LMU compresses the input's history (lasting θ seconds) into a low-dimensional state.

If we sweep Eq. 15 across $\theta'\ \theta^{-1}$, we obtain the temporal "Legendre basis functions" of the LMU (see Fig. 7 for an example). This provides a means of

[3]The optimization problem from Eq. 6 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).

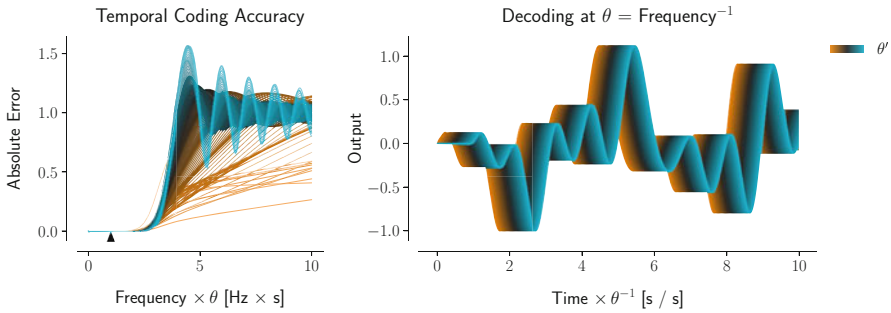


Fig. 6 Decoding a sliding window of length θ. Each line corresponds to a different delay, ranging from 0 to θ, decoded from a single LMU ($q = 12$). (Left) Error of each delay, as the input frequency, f, is increased relative to θ. Shorter delays are decoded more accurately than longer delays at higher frequencies. A triangle marks $\theta = f^{-1}$. (Right) Example simulation decoding a sliding window of white noise with a cutoff frequency of θ^{-1} Hz. (Reproduced from Voelker and Eliasmith [92, Fig. 4], reprinted courtesy of The MIT Press)

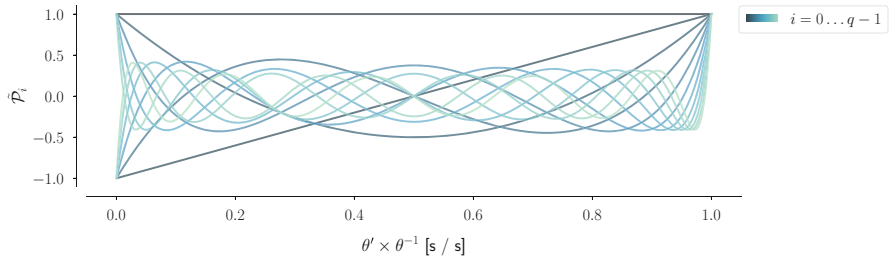


Fig. 7 Temporal basis functions for the Legendre state-space realization of the LMU ($q = 12$). Each line corresponds to the basis function of a single dimension (i) ranging from 0 (darkest) to $q - 1$ (lightest). The ith basis function is the shifted Legendre polynomial over $\theta' \, \theta^{-1}$ with degree i. The state vector of the LMU takes a linear combination of these q basis functions in order to represent a sliding window of length θ

understanding the relationship between the chosen state-space representation (i.e., the 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 LMU. The instantaneous value of each dimension acts as a coefficient on its Legendre 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 as it slides over time.

Mathematically, any linear transformation of $\mathbf{x}(t)$ is equivalent to a linear transformation of the window $[u(t - \theta), u(t)]$ projected onto the Legendre polynomials. Thus, any integral transform, such as the Fourier transform or the Taylor series expansion of $u(t - \theta')$ up to degree $q - 1$, can be approximated by taking a linear transformation of the LMU's state vector (see Voelker [88], Sect. 6.1.4 for details).

The encoder of each neuron can also be understood directly in these terms as taking a linear combination of the basis functions (via Eq. 1). Each neuron nonlinearly encodes a projection of the sliding 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 LMU 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.

3.2 LMU Application Examples

3.2.1 Deep LMU Networks

We now consider an architecture that stacks multiple LMUs on top of one another, to form a *deep LMU* (DLMU) chaining multiple delays together. 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 ith 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). Since each layer implements the transfer function $[q - 1/q]e^{-\gamma s}$, by the convolution theorem, the overall filter is the product of each transfer function. Therefore, the error is characterized by the filter:

$$E_{q,k}\,(\theta s) = \left([q - 1/q]\,e^{-\gamma s}\right)^k - e^{-\theta s}. \tag{16}$$

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).[4] We evaluate $E_{q,k}(\theta s)$ while varying (q,k) in such a way that keeps the resource cost fixed (see Fig. 8).

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. 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,

[4] Also assuming the use of a dense state-space realization such as from zero-order hold discretization of the LMU dynamics.

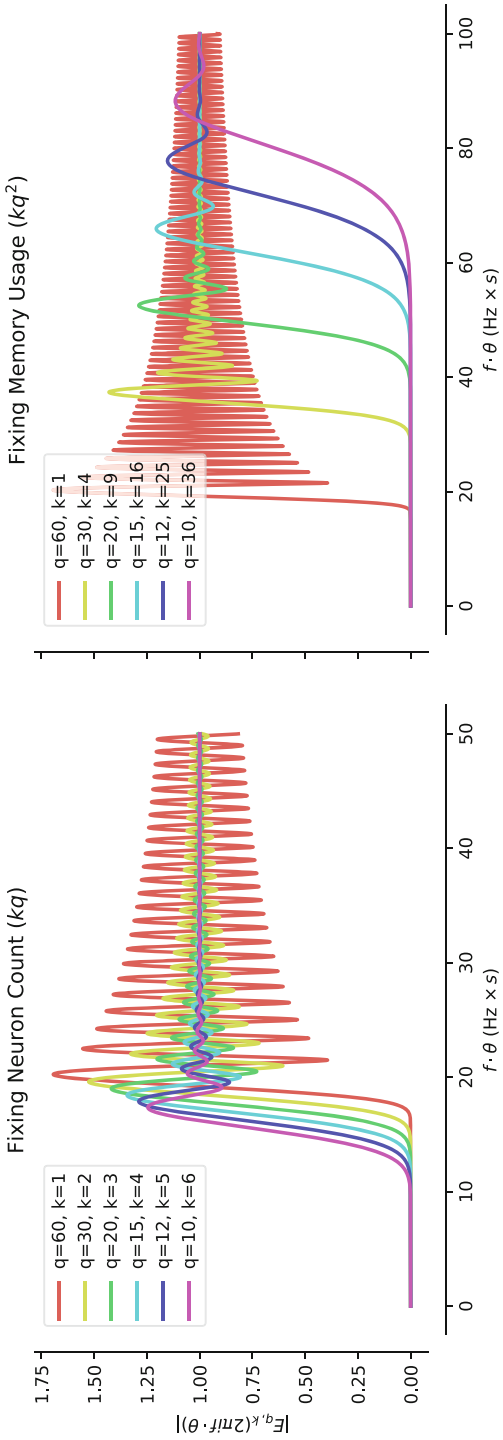


Fig. 8 Visualizing the error in the deep LMU of width q and depth k (see Eq. 16) for two different resource-cost functions. (Left) Fixing the number of dimensions, kq, to be constant. Smaller k scales further in this case. (Right) Fixing the density of connections, kq^2, to be constant. Smaller q is more accurate in this case

as the LMU 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 connection density or the number of 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 Fig. 8, the error oscillates with dampened amplitude beyond the radius of convergence for larger k. This can be seen from Eq. 16 by noticing that $[q - 1/q]e^{-\gamma s}$ is a complex number with a magnitude less than 1, which is then exponentiated to the power of k. By a triangle-inequality argument, this can be shown to effectively regularize the error k times towards 1 outside the radius of convergence [88].

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 corresponding Legendre basis functions via Principles 1 and 2. Deeper networks effectively partition the basis functions into individually orthogonalized 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.

3.2.2 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 ith layer will represent $\mathscr{L}^{-1}\{H(s)^k\ U(s)\}(t)$, that is, the input $u(t)$ convolved k times with $h(t)$. This is demonstrated in Fig. 9 (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 low-pass-filtered in time. This has the often[5] undesirable effect of losing the information within the frequency content of the input. This phenomenon contributes to the misconception that the NEF does not support the high-speed transmission of information through networks, as discussed in Sect. 5.3.

To solve this problem, we are free to scale the synaptic time constant, τ, arbitrarily small, so long as n is scaled as $\mathcal{O}(\tau^{-2})$ to maintain the same level of feed-forward precision [88, Table 3.1]. Alternatively, if τ cannot be changed, then one can use Principle 3 to implement a low-pass filter $(\theta s + 1)^{-1}$ with arbitrarily small θ, which likewise requires $\mathcal{O}(\theta^{-2})$ neurons for some fixed accuracy. Our solution can be viewed as a generalization of the latter.

[5]Goldman [43] has shown that repeated low-pass filtering can be usefully exploited to implement an integrator, by summing across all of the filters.

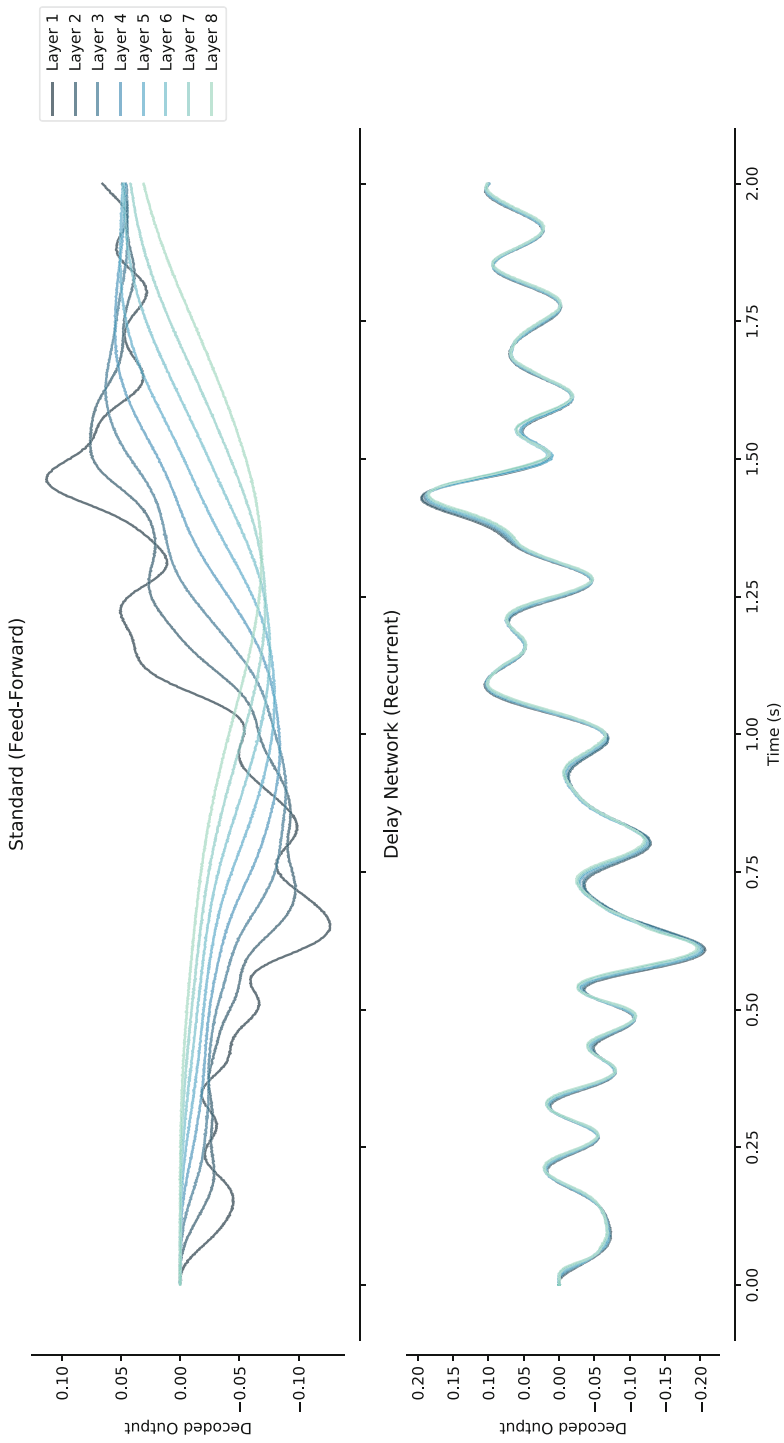


Fig. 9 Demonstrating the ability of the LMU to approximate a synaptic deconvolution operation, realizing a pure communication channel. The test input $u(t)$ is a white noise signal band-limited at 10 Hz. (Top) A standard feed-forward communication channel, implemented by decoding the identity function at each layer. (Bottom) A recurrent communication channel, implemented by a deep LMU (Sect. 3.2) approximating $u(t)$ at each layer. Each layer is an LMU 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 low-pass synapse with time constant$\tau = 100$ ms.

A natural solution falls out of the LMU: 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 low-pass filter) to produce a filtered version (e.g., phase-shifted) of this signal. Nevertheless, the state vector is reconstructing an unfiltered 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 LMU essentially models the statistics of the input and provides a robust estimate of the current $u(t)$ from the spiking activity of the population. The transformation to do so is simply a linear readout of the state (Eq. 15 with $\theta' = 0$):

$$u(t) \approx \sum_{i=0}^{q-1} \mathcal{P}_i(0) x_i(t). \tag{17}$$

We use this fact in Fig. 9 (bottom) to instantaneously propagate the input through eight layers, using the same neurons and low-pass synapses as in (top). The difference between these two simulations is that the recurrence, local to each layer, effectively undoes the synaptic filtering by using its internal model of the input's history. This demonstrates the utility of including recurrence at each layer, not only to support dynamical computations but also to maintain the frequency content of the input signal while facilitating high-speed computation through deep neural structures.

4 A Neuromorphic Hardware Example

Among current neuromorphic hardware systems, a number support the compilation of NEF networks, including a VLSI prototype [14], several FPGA implementations [8, 63, 99, 100], several GPU implementations [5, 71], and, most recently, TrueNorth [35].

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 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. In the case of the NEF, these criteria have been validated by many of the extensions and analyses described in Voelker [88], as well as by past work compiling networks onto Neurogrid [24],

SpiNNaker [52, 64, 65], TrueNorth [35], and many other architectures [8, 14, 71, 99].

However, the current Nengo backends for Braindrop [67] and Loihi [42] are brand new – as are the chips themselves [18, 66] – and thus currently fall short of their promise to realize *large-scale* functional SNNs in neuromorphic hardware, for the time being. 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.[6] And with current software workarounds, the feed-forward precision stops scaling in NengoLoihi after about 400 neurons. Nevertheless, this is the first and only software abstraction to currently make use of Loihi [9] outside of Intel [57], and it is under active development. We expect this to get significantly better with time.

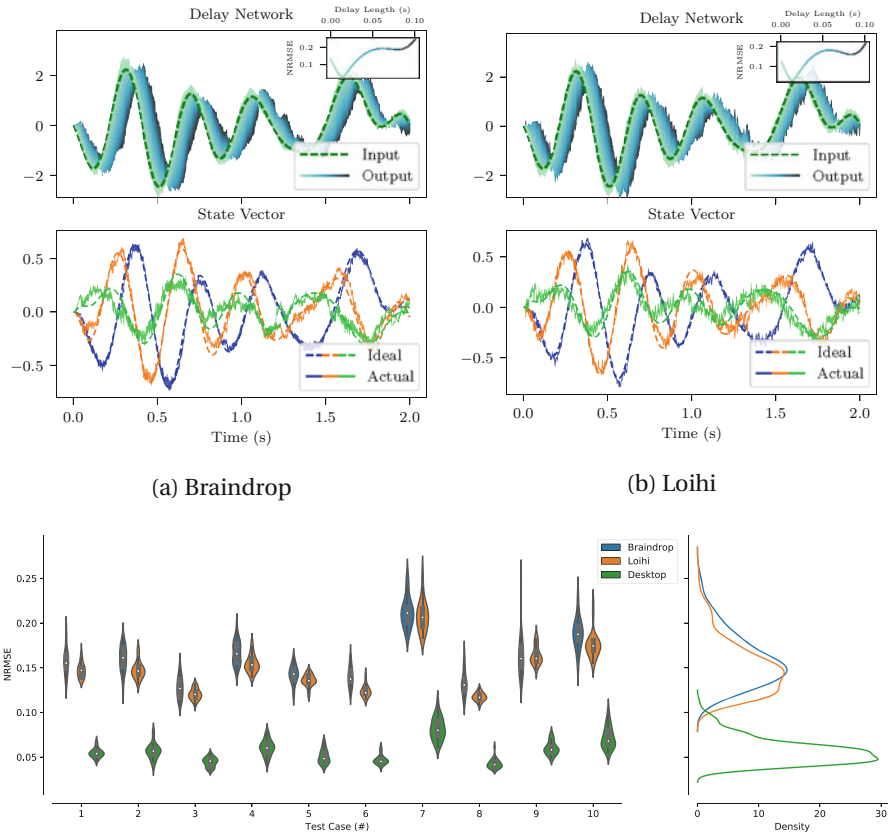
Nevertheless, these two architectures – Braindrop and Loihi – represent significant milestones in the evolution of neuromorphic hardware. The first, Braindrop, consumes energy roughly equivalent to 381 fJ per synaptic operation, for typical network configurations, or about 10–20 times more than the human brain [10]. 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 section is to demonstrate that the fundamentals for systematically programming neuromorphic hardware are in place. We focus here on demonstrating the principles of the NEF and a few of its extensions applied to the LMU. This dynamical system is described in the high-level language of Nengo but mapped onto two vastly different state-of-the-art neuromorphic architectures: Braindrop and Loihi.

4.1 The LMU

We now instantiate the LMU on state-of-the-art neuromorphic hardware (see Fig. 10). To implement the LMU, three pools, each containing 128 spiking LIF neurons, are recurrently coupled 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 low-pass synapse with $\tau = 20$ ms and weighted to decode both the state vector and the window of history via Eq. 15.

[6] Determined empirically using the NengoLoihi = 0.5.0 emulator



(a) Braindrop

(b) Loihi

(c) Overall Error

Fig. 10 Legendre Memory Unit (LMU; $q = 3$, $\theta = 100$ ms) running on state-of-the-art neuromorphic hardware. (**a**) Nengo Braindrop implementation (Reproduced from Neckar et al. [67, Fig. 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

For this experiment, we use identical Nengo model code for both neuromorphic backends. On Braindrop (see Fig. 10(a), reproduced from Neckar et al. [67, Fig. 16]), the chip is configured to use the default distribution of synaptic time constants (mean $\tau \approx 18$ ms). For Loihi (see Fig. 10b), 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 window $\theta' \in [0, \theta]$.

Table 1 Performance of the Legendre Memory Unit (LMU) 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

Method	95% Confidence interval
Braindrop	[0.156, 0.163]
Loihi	[0.146, 0.153]
NengoLoihi emulator	[0.145, 0.151]
Reference CPU	[0.055, 0.059]

Table 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 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 (see [88]).

5 The NEF Compared to Other Methods

5.1 Theoretical Comparison

Conventional RNNs are notoriously difficult to train and interpret in practice [6]. The paradigm of reservoir computing [RC; 49, 61] 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; 83] 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. 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 [22] 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. Many have argued that dynamics in biological systems are unions of low-dimensional attractor manifolds [16, 84, 97]. 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, optimization in the time domain using a teacher signal [26], and backpropagation through time [71] may all be applied to learn the dynamics from raw data.

We now consider the architectural relationships, focusing on the one-dimensional input-output case for notational convenience. In the case of RC networks (see Fig. 11a), a conceptual separation between a fixed *reservoir* of randomly connected neural units and a learned *readout* is made. 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. 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.

The first-order reduced and controlled error [FORCE; 83] 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 Fig. 11b). Specifically, the recurrent weights include an additional outer-product term, $\mathbf{e}\mathbf{d}^T \in \mathbb{R}^{n \times n}$, where $\mathbf{e} \in \mathbb{R}^n$ is a fixed random

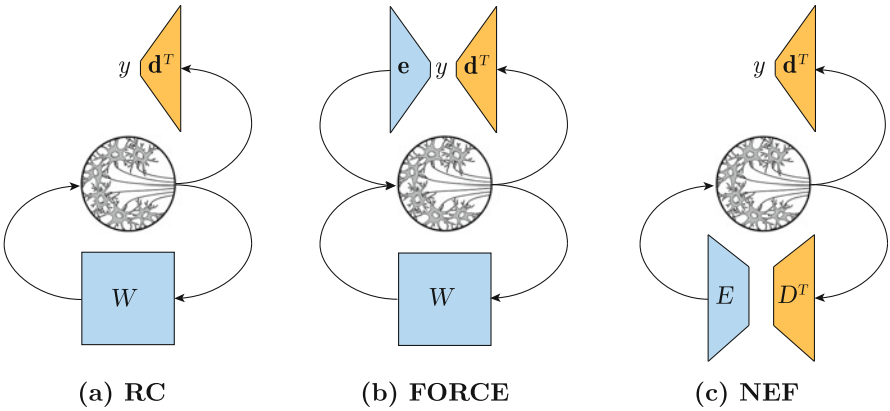


Fig. 11 Comparing the recurrent architectures of (**a**) reservoir computing (RC), (**b**) first-order reduced and controlled error (FORCE), and (**c**) the neural engineering framework (NEF). Blue weights are fixed and randomly chosen; orange weights are learned (online or offline). Each ensemble is a pool that nonlinearly encodes the weighted activities. The input is omitted for simplicity

vector and $\mathbf{d} \in \mathbb{R}^n$ is the same decoding vector from before. 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 of 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 (see Sect. 5.2). This discussion extends to the full-FORCE method [22], 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 [87].

The NEF provides an alternative method for building dynamical neural networks (see Fig. 11c). 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 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 selection of neuron models, the models of postsynaptic currents (PSCs), and whether the simulation is analog or digital [92]. Like 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; 77]. If all relevant state variables are identified and all target models are properly leveraged, then the high-dimensional dynamics introduced by W serve absolutely no purpose.

5.2 Practical Comparison

We have developed a rigorous theory to understand the LMU in terms of its linear dynamics and its nonlinear encoding of time and used the NEF to present a spiking example in Fig. 5. Here, we show that the NEF outperforms reservoir computing (RC) methods – using either spiking [LSM; 61] or rate-based [ESN; 49] neurons – in memory capacity and nonlinear computation while reducing the simulation time and space requirements by a factor of $\mathcal{O}(n)$.

5.2.1 Reservoir Computing: 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 [50]. Theoretical and experimental results in the past have pointed to the limited capability of random feedback to maintain memory [17, 98], in particular finding

Table 2 Hyperopt parameters for the linear benchmark in Sect. 5.2

	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

that linear feedback *maximizes* memory capacity [62] – consistent with Sect. 3, 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. This is consistent with our findings below. Moreover, the LMU provides 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.

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; 59]. As well, 95% confidence intervals were bootstrapped and plotted. We ported a Python implementation of the ESN from [60] to Nengo and implemented it analogously to the LMU. 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 with three trials per iteration, to maximize the validation error for 200 ms delays (see Table 2). 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 LMU (q).[7]

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 (dt = 1 ms). For the NEF case, the prescribed dynamical system is a $\theta = 200$ ms delay, implemented by mapping Eq. 13 onto a discretized low pass. Voelker [88] shows

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

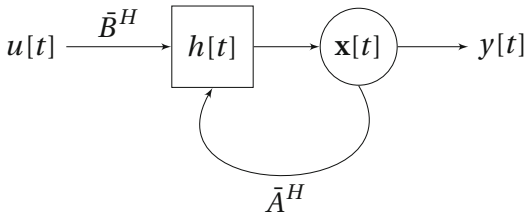


Fig. 12 LMU model for digital architecture. The synapse $h[t]$ is driven by $\overline{A}^H \mathbf{x}[t] + \overline{B}^H u[t]$ to yield the state $\mathbf{x}[t]$. This state is nonlinearly encoded by a heterogeneous population of neurons and subsequently decoded to estimate the desired $y[t]$

that the delay length can be trained from raw data. Importantly, the same networks are used to compute all of the different delays reported (Fig. 12).

As shown in Fig. 13, the NEF's performance is slightly better than ESNs for both LIF rate (i.e., non-spiking) neurons and tanh rate neurons and significantly better than LSMs for spiking LIF neurons. This demonstrates that, in terms of memory capacity, the LMU as a low-dimensional reservoir not only outperforms RC in the rate-based case but also 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.[8] The success of the LMU should not be surprising given that we have mapped the ideal delay 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.

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 Fig. 14). We again note that traditional RC suffers from scaling issues since the recurrent weight matrices have $\mathcal{O}(n^2)$ coefficients. Consequently, the NEF is more resource-efficient than these RC methods by a factor of $\mathcal{O}(n)$. RC networks often include a sparsity constraint of 20% connectivity, which is still $\mathcal{O}(n^2)$, in order to improve performance [58, 59]. 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 Eq. 13, as q linear-state variables are in exact correspondence with the ideal linear system (after the appropriate discretization) (see Sect. 5.2).

[8] As additional validation, lower input frequencies or shorter delay lengths were possible with the LSM.

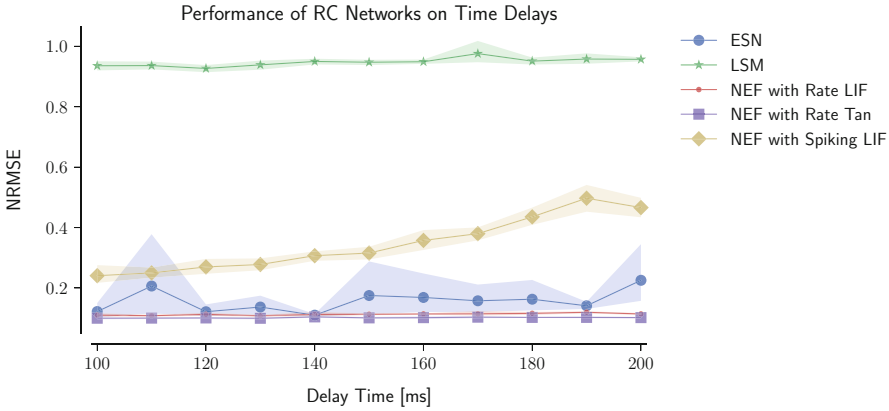


Fig. 13 Performance from training various linear readouts to compute delays of 25 Hz band-limited white noise (bootstrapped across 10 trials). Each line corresponds to a single reservoir of 1,500 neurons, either randomly connected (in the case of ESNs and LSMs) or specifically engineered (in the case of the NEF)

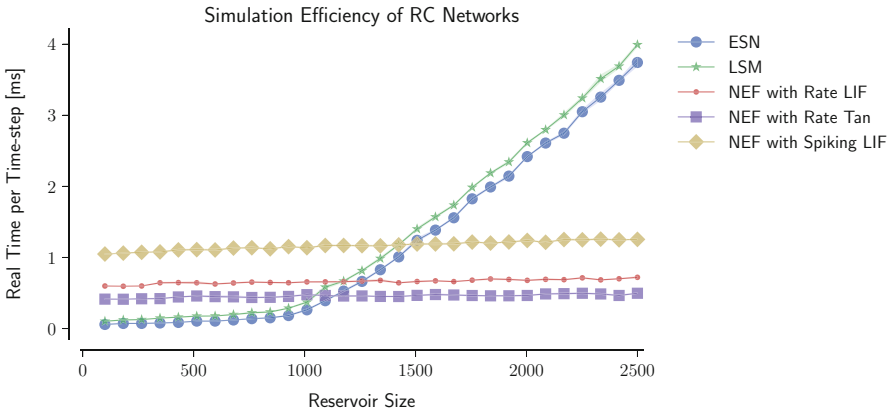


Fig. 14 Cost of simulating each RC network as the reservoir size is increased (bootstrapped across 10 trials). Conventional RC approaches require $\mathcal{O}(n^2)$ space and time, while the NEF improves this to $\mathcal{O}(n)$ for constant dimensionality

Another key finding is that, as shown in Table 2, 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. [17]. Crucially, this limits the ability of the ESN to perform nonlinear computations across the delay interval, as we now show.

5.2.2 Reservoir Computing: Nonlinear Benchmark

To demonstrate our ability to compute nonlinear window functions, we consider the function $y(t) = u(t - \theta)\, u(t)$. 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) [44]. However, the particular choice of function is not of importance, as the training can be data-driven or analyzed using theory from Sect. 3.1.

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 and apply Nengo's regularized least-squares solver, which bypasses the need to explicitly simulate the network on any input signals.

The model architecture of this LMU is, again, depicted in Fig. 12, 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 used Hyperopt [7] 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 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 LMU, 51.8% for the spiking LMU, 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 LMU, 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 LMU. In addition, trials took on average 5.10 s for the sigmoid LMU, 6.44 s for the spiking LMU, and 17.7 s for the ESN. This difference is a consequence of not simulating the LMU for training, and from using factorized weight matrices (i.e., encoders and decoders) to simulate the LMU. These results are consistent with that of the linear benchmark, except for the additional observation that here the spiking LMU 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.

5.2.3 FORCE Learning

We also compared our NEF solution to both FORCE [83] and full-FORCE [22] networks (also see Sect. 5.1). For this experiment, we first ported the same implementation described in DePasquale et al. [22] to Nengo and verified that it 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. [21], Thalmeier et al. [85], and Nicola and Clopath [68].

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 dt = 5 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 LMU (Eq. 13), consisting of just $n = 6$ linear units, coupled to one another by a discretized mapping ($q = 6$). In this scenario, the only error that remains is that 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 $\mathcal{O}(n^2)$ 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 Fig. 15 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.

5.3 Criticisms of the NEF

A long-standing debate in neuroscience has traditionally revolved around the question of whether biological neurons transmit information using a "rate code"

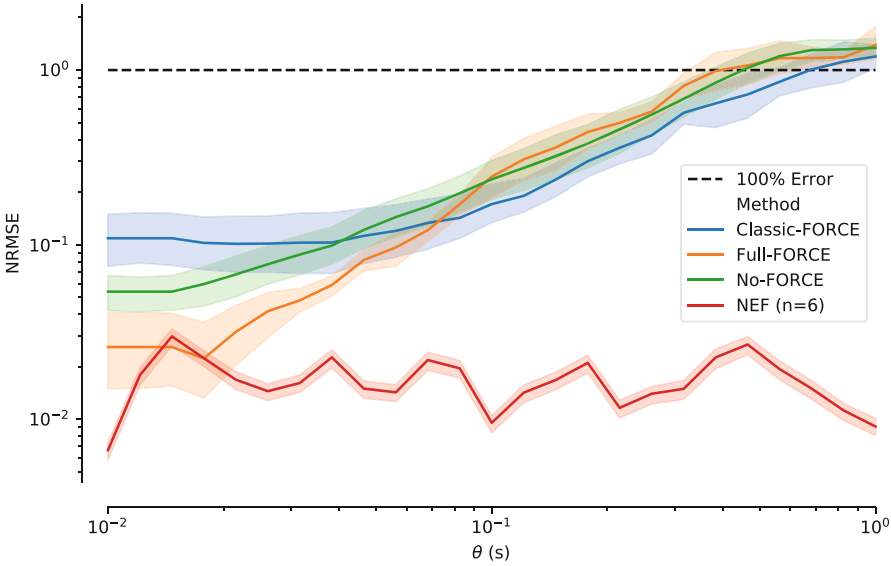


Fig. 15 Comparison of several FORCE learning methods versus the NEF on a delay task Networks are trained to delay a time-varying input by θ seconds. Each FORCE network consists of 500 tanh neurons, while the NEF network is six linear neurons

in which the information is encoded by the firing rates of individual neurons [2] or a "spike-timing code" in which information is encoded by the precise temporal patterns of spike trains [72] or likewise their temporal order in relation to one another [86].

However, there is historically little consensus between neuroscientists as to what exactly constitutes a rate code [31, pp. 89–91]. Gerstner [41] 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. [34] models the adaptive dynamics of neurons in the fly visual system and concludes that principles of its code depend on the timescale of interest. Eliasmith and Anderson [31] propose 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, including Lagorce and Benosman [55] and Frady and Sommer [36] for two recent examples. Specifically, this mischaracterization has led to the misapplication of many criticisms [40] that stem from the original proposal of Adrian [2], 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 [55] and Frady and Sommer [36]:

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.

We now challenge each claim in turn. For the first three, we refer to the same simulation depicted in Fig. 16. This simulation applies the NEF, as described in Sect. 2.1, to the case of an autonomous two-dimensional oscillator ($n = 5{,}000$, $\tau = 0.1$ s, 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 s of simulation to avoid initial transients.

1. **The NEF can exhibit repeatedly precise sequences of action potentials.**

See Fig. 16 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 [86], but also the spike timing is precise (± a couple of 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.[9] The precision of a feed-forward network scales as $\mathcal{O}\left(\tau\sqrt{n}\right)$ [88, Table 3.1]. Thus, one is free to set the synaptic time constant arbitrarily small, so long as the number of neurons is increased in proportion. This is both theoretically and numerically verified in Voelker [88]. This enables arbitrarily fast transmission of information throughout the network when appropriate criteria are met (detailed in [88]). Yet, even when constrained to longer time constants, Section 3.2 demonstrates a novel deep NEF network that is capable of *instantaneously* propagating low-frequency stimuli through eight 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 [29] such as the central pattern generator driving lamprey locomotion [30]. Indeed, Fig. 16 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

[9]The postsynaptic filters are leveraged to participate in the required computation (see Principle 3; Section 0.2.1). There is no unwanted phase shift.

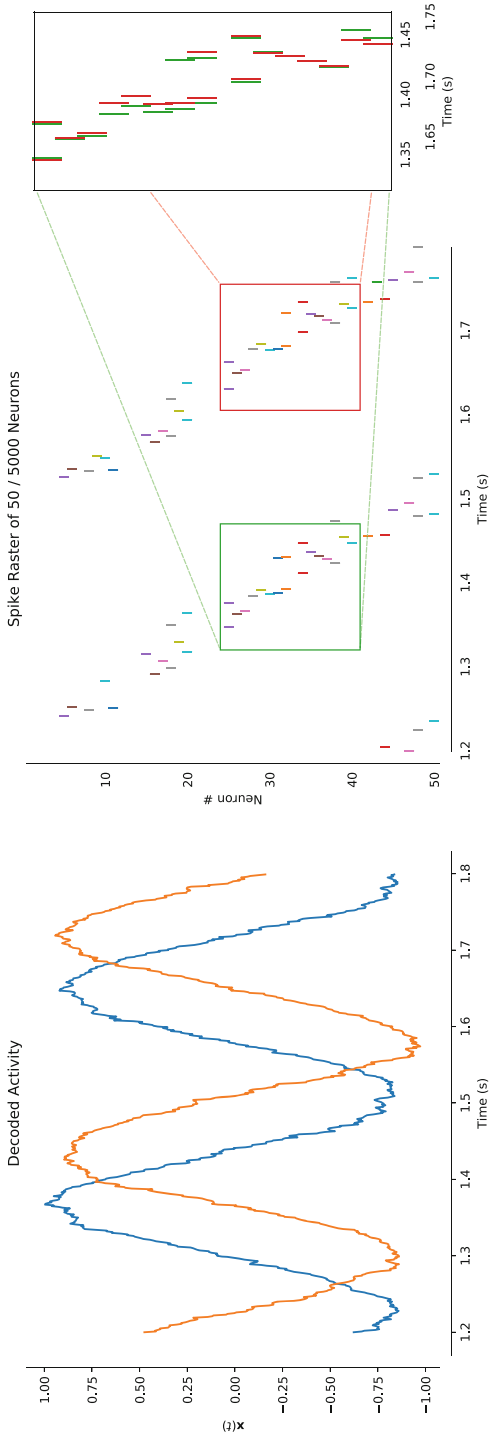


Fig. 16 Demonstrating the irrelevance of identifying a "spike-time code" versus a "rate code" in the context of a standard NEF-optimized network. We define this as a "postsynaptic current code" instead. (Left) An ensemble of LIF neurons are trained to oscillate at approximately 3.5 Hz. (Right) Spike 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

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**.

Voelker [88] provides an example of ten spiking LIF neurons implementing a line attractor ($\tau = 5$ ms), as well as 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 (e.g., for Spaun). 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). But one cannot consider the questions of resource usage and functional precision in 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.

The confusion surrounding rate coding in the NEF has essentially risen from the adoption of Eq. 7. However, this merely reformulates the optimization procedure to be more efficient, without sacrificing correctness, as proven in Voelker [88]. We remark that, in Fig. 16, 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 Eq. 7 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: "postsynaptic 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; 87]. This is summarized by taking the encoding Eq. 1 and decoding Eq. 4, which fold into the weight Eq. 8 – and combining them in a similar manner to Stöckel et al. [82]:

$$\alpha_i \langle \mathbf{e}_i, \mathbf{x}(t) \rangle + \beta_i \approx \sum_{j=1}^{n} \sum_{m} \omega_{i\ j} h\left(t - t_{j,m}\right). \tag{18}$$

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 Eq. 7 requires careful proof [88, Theorem 3.2.1], and

the subtleties surrounding why this can matter are challenging. 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.

If one is still unconvinced, then, as mentioned in Sect. 5.1 when comparing the NEF to RC and FORCE, one may forgo Eq. 7 and perform the optimization directly in the spiking time domain [26, 37] or even apply back-propagation [71]. 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.

Conclusion While we have addressed core criticisms of the NEF that are often misconstrued, we do not want to suggest that the NEF is a panacea for neural modelers. That is, the significant theoretical challenges posed by the many biological nonlinearities found in neural systems remain when using the NEF. While recent extensions to the framework have allowed more sophisticated NEF models (see, e.g., Sect. 2.1), the NEF does not completely solve the general problem of using an arbitrarily complex neuron model to perform any specifiable high-level computation or dynamical system. Nevertheless, recent work has allowed conductance-based synapses to be systematically introduced [79], has demonstrated the introduction of various challenging synaptic and connectivity constraints [80], and has shown how multicompartment, multi-channel neurons can be used in place of simpler LIF neurons [26, 32]. Consequently, we consider the NEF a useful tool for continuing to introduce additional biological complexity as needed while being focused on neural function. In the context of neuromorphics, biological complexity is often replaced with constraints from hardware implementations due to ease of realization, fabrication variability, or intrinsic nonlinearity. Interestingly, the mathematical and theoretical tools for handling these challenges, be they biological or neuromorphic, are often similar.

6 Future Directions

As a general-purpose method for developing neural networks, both biologically constrained and those focused purely on machine learning applications, the NEF is currently supporting a wide variety of research directions. Here, we focus on those more directly relevant to the neuromorphic community.

A significant challenge for this community is developing scalable, widely available hardware, and clearly demonstrating its advantages. This is especially true in the current context of a rapidly increasing number of targeted neural network accelerators. One way that the NEF and specific networks like the LMU can help in this increasingly complex landscape of neural hardware is to provide clear points of comparison, many of which are likely to demonstrate the advantages of neuromorphic chips. For instance, the LMU is a noise-robust approach that works in a spiking network. As a result, it can be run on neuromorphic and non-neuromorphic hardware. The same cannot be said for the standard LSTM. As a result, it is not possible to run the exact same LSTM algorithm on both neuromorphic and non-

neuromorphic hardware. Since the LMU is beating the LSTM (and other leading RNN architectures) on standard benchmarks in the non-neuromorphic case [95], it can serve as a state-of-the-art approach to time-series problems that allows direct comparison of neuromorphic and non-neuromorphic hardware. This is further made possible by Nengo's existing and future support of multiple hardware targets, including neuromorphic and non-neuromorphic hardware. Such clean, fair, head-to-head comparisons should make a clear case for the benefits of different hardware approaches.

This same kind of direct transition between spiking and non-spiking networks is supported in general by the NEF and directly in Nengo for the NEF and many other approaches, including standard deep learning. As such, a core future direction is to continue developing these and related methods to allow a smooth transition across spiking and non-spiking algorithms and hardware. This means not only developing algorithms like the LMU but also new kinds of representations (e.g., spatial semantic pointers; Komer et al. [54]) and new techniques for interpolating between spiking and non-spiking models [96, 69, submitted].

As mentioned, a core challenge for neuromorphics is scalability. Addressing this requires large-scale, functional models that run on such hardware. As such, we are pursing variants of the LMU, with a particular view to scalability, as well as co-designing hardware and neural network architectures hand in hand. More generally, the NEF supports building very large functional models (e.g., Spaun), so many such NEF models should be able to provide challenging tests of hardware scalability. However, the clean comparison of the LMU with other standard machine learning RNNs makes it a natural choice for addressing the scaling challenge as well.

Finally, the ability of the NEF to allow the construction of large functional models suggests an interesting connection point to common deep learning methods like gradient descent. In particular, it is possible to backpropagate through standard NEF models. This opens up the possibility of building large functional models, like Spaun, that would be extremely difficult to learn "from scratch" and then fine-tune them using backpropagation. This is currently supported with Nengo [71], although it remains to be seen when and how such a technique can be fully exploited. Perhaps this will prove to be a useful kind of "seeding" for large-scale architectures in deep learning.

7 Conclusions

In this brief overview, we have provided an introduction to the NEF and some examples of its application to building novel RNNs and mapping those onto different kinds of neuromorphic hardware. While we have only scratched the surface of example applications of the NEF, we have attempted to provide the core theoretical foundations both formally and through examples. Our comparisons to other approaches help to demonstrate that the NEF offers certain efficiency and accuracy advantages, which is critical for neuromorphic applications that are focused on low-power implementation. Coupled with implementation in the Nengo

software, the NEF provides both theoretical and practical tools for building a wide variety of spiking networks on a wide variety of hardware platforms. We hope that the NEF, as it continues to evolve, finds application in ways we have not yet anticipated.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: TensorFlow: A system for large-scale machine learning. OSDI. **16**, 265–283 (2016)
2. Adrian, E.D.: The Basis of Sensation. Christophers, London (1928)
3. Appeltant, L., Soriano, M.C., Van der Sande, G., Danckaert, J., Massar, S., Dambre, J., Schrauwen, B., Mirasso, C.R., Fischer, I.: Information processing using a single dynamical node as complex system. Nat. Commun. **2**, 468 (2011)
4. Armstrong-Gold, C.E., Rieke, F.: Bandpass filtering at the rod to second-order cell synapse in salamander (Ambystoma tigrinum) retina. J. Neurosci. **23**(9), 3796–3806 (2003)
5. Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T.C., Rasmussen, D., Choo, X., Voelker, A.R., Eliasmith, C.: Nengo: A Python tool for building large-scale functional brain models. Front. Neuroinform. **7**(48) (2014). https://doi.org/10.3389/fninf.2013.00048
6. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE Trans. Neural Netw. **5**(2), 157–166 (1994)
7. Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., Cox, D.D.: Hyperopt: A Python library for model selection and hyperparameter optimization. Comput. Sci. Discov. **8**(1), 014008 (2015)
8. Berzish, M., Eliasmith, C., Tripp, B.: Real-time FPGA simulation of surrogate models of large spiking networks. In: International Conference on Artificial Neural Networks (ICANN), Springer, Cham (2016)
9. Blouw, P., Choo, X., Hunsberger, E., Eliasmith, C.: Benchmarking keyword spotting efficiency on neuromorphic hardware. arXiv preprint arXiv:1812.01739 (2018)
10. Boahen, K.: A neuromorph's prospectus. Comput. Sci. Eng. **19**(2), 14–28 (2017)
11. Brogan, W.L.: Modern Control Theory, 3rd edn. Prentice-Hall, New Jersey (1991)
12. Choo, X.: Spaun 2.0: Extending the world's largest functional brain model. Ph.D. thesis, University of Waterloo (2018)
13. Choudhary, S., Sloan, S., Fok, S., Neckar, A., Trautmann, E., Gao, P., Stewart, T., Eliasmith, C., Boahen, K.: Silicon neurons that compute. In: International Conference on Artificial Neural Networks, vol. 7552, pp. 121–128. Springer (2012)
14. Corradi, F., Eliasmith, C., Indiveri, G.: 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 (2014)
15. Cover, T.M., Thomas, J.A.: Elements of Information Theory. Wiley, New York (2012)
16. Cunningham, J.P., Byron, M.Y.: Dimensionality reduction for large-scale neural recordings. Nat. Neurosci. **17**(11), 1500–1509 (2014)
17. Dambre, J., Verstraeten, D., Schrauwen, B., Massar, S.: Information processing capacity of dynamical systems. Sci. Rep. **2**, 514 (2012)
18. Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S.H., Dimou, G., Joshi, P., Imam, N., Jain, S., et al.: Loihi: A neuromorphic manycore processor with on-chip learning. IEEE Micro. **38**(1), 82–99 (2018)
19. de Jong, J., Voelker, A.R., van Rijn, H., Stewart, T.C., Eliasmith, C.: Flexible timing with delay networks – The scalar property and neural scaling. In: International Conference on Cognitive Modelling, Society for Mathematical Psychology (2019)
20. De Vries, B., Principe, J.C.: The gamma model – A new neural model for temporal processing. Neural Netw. **5**(4), 565–576 (1992)

21. DePasquale, B., Churchland, M.M., Abbott, L.: Using firing-rate dynamics to train recurrent networks of spiking model neurons. arXiv preprint arXiv:1601.07620 (2016)
22. DePasquale, B., Cueva, C.J., Rajan, K., Abbott, L., et al.: Full-FORCE: A target-based method for training recurrent networks. PLoS One. **13**(2), e0191527 (2018)
23. Destexhe, A., Mainen, Z.F., Sejnowski, T.J.: An efficient method for computing synaptic conductances based on a kinetic model of receptor binding. Neural Comput. **6**(1), 14–18 (1994)
24. Dethier, J., Nuyujukian, P., Eliasmith, C., Stewart, T.C., Elasaad, S.A., Shenoy, K.V., Boahen, K.A.: A brain-machine interface operating with a real-time spiking neural network control algorithm. In: Advances in Neural Information Processing Systems, pp. 2213–2221. (2011)
25. DeWolf, T., Jaworski, P., Eliasmith, C.: Nengo and low-power AI hardware for robust, embedded neurorobotics. Frontiers in Neurorobotics (2020)
26. Duggins, P.: Incorporating biologically realistic neuron models into the NEF. Master's thesis, University of Waterloo (2017)
27. Duggins, P., Stewart, T.C., Choo, X., Eliasmith, C.: Effects of guanfacine and phenylephrine on a spiking neuron model of working memory. Top. Cogn. Sci. **9**, 117–134 (2017)
28. Eliasmith, C.: How to Build a Brain: A Neural Architecture for Biological Cognition. Oxford University Press, New York (2013)
29. Eliasmith, C., Anderson, C.H.: Developing and applying a toolkit from a general neurocomputational framework. Neurocomputing. **26**, 1013–1018 (1999)
30. Eliasmith, C., Anderson, C.H.: Rethinking central pattern generators: A general approach. Neurocomputing. **32–33**, 735–740 (2000)
31. Eliasmith, C., Anderson, C.H.: Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems. MIT Press, Cambridge, MA (2003)
32. Eliasmith, C., Gosmann, J., Choo, X.: BioSpaun: A large-scale behaving brain model with complex neurons. arXiv preprint arXiv:1602.05220 (2016)
33. Eliasmith, C., Stewart, T.C., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., Rasmussen, D.: A large-scale model of the functioning brain. Science. **338**(6111), 1202–1205 (2012)
34. Fairhall, A.L., Lewen, G.D., Bialek, W., van Steveninck, R.R.d.R.: Efficiency and ambiguity in an adaptive neural code. Nature. **412**(6849), 787 (2001)
35. Fischl, K.D., Stewart, T.C., Fair, K.L., Andreou, A.G.: Implementation of the neural engineering framework on the TrueNorth neurosynaptic system. In: IEEE Biomedical Circuits and Systems Conference (BioCAS), pp. 587–590. IEEE (2018)
36. Frady, E.P., Sommer, F.T.: Robust computation with rhythmic spike patterns. arXiv preprint arXiv:1901.07718 (2019)
37. Friedl, K.E., Voelker, A.R., Peer, A., Eliasmith, C.: Human-inspired neurorobotic system for classifying surface textures by touch. Robot. Autom. Lett. **1**(1), 516–523 (2016)
38. Funahashi, K., Nakamura, Y.: Approximation of dynamical systems by continuous time recurrent neural networks. Neural Netw. **6**(6), 801–806 (1993)
39. Galluppi, F., Davies, S., Furber, S., Stewart, T., Eliasmith, C.: Real time on-chip implementation of dynamical systems with spiking neurons. In: International Joint Conference on Neural Networks (IJCNN), pp. 1–8. IEEE (2012)
40. Gautrais, J., Thorpe, S.: Rate coding versus temporal order coding: A theoretical approach. Biosystems. **48**(1–3), 57–65 (1998)
41. Gerstner, W.: Spiking neurons. Pulsed Neural Netw. **4**, 3–54 (1999)
42. GitHub.: nengo/nengo-loihi==0.5.0: Run nengo models on Intel's Loihi chip. https://github.com/nengo/nengo-loihi/ (2019). Accessed 20 Jan 2019
43. Goldman, M.S.: Memory without feedback in a neural network. Neuron. **61**(4), 621–634 (2009)
44. Gosmann, J.: Precise Multiplications with the NEF. Technical Report. Centre for Theoretical Neuroscience, Waterloo (2015)
45. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)

46. Hunsberger, E.: Spiking deep neural networks: engineered and biological approaches to object recognition. Ph.D. thesis, University of Waterloo (2018)
47. Hunsberger, E., Eliasmith, C.: Spiking deep networks with LIF neurons. arXiv preprint arXiv:1510.08829 (2015)
48. Hunsberger, E., Eliasmith, C.: Training spiking deep networks for neuromorphic hardware. arXiv preprint arXiv:1611.05141 (2016)
49. Jaeger, H.: The "Echo State" Approach to Analysing and Training Recurrent Neural Networks. German National Research Center for Information Technology, Bonn (2001)., GMD Technical Report, 148:34
50. Jaeger, H.: Short Term Memory in Echo State Networks. Technical report, Fraun-hofer Institute for Autonomous Intelligent Systems (2002)
51. Kauderer-Abrams, E., Gilbert, A., Voelker, A.R., Benjamin, B.V., Stewart, T.C., Boahen, K.: A population-level approach to temperature robustness in neuromorphic systems. In: IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, Baltimore (2017)
52. Knight, J., Voelker, A.R., Mundy, A., Eliasmith, C., Furber, S.: Efficient SpiNNaker simulation of a heteroassociative memory using the Neural Engineering Framework. In: International Joint Conference on Neural Networks (IJCNN). IEEE, Vancouver (2016)
53. Koch, C., Segev, I.: Methods in Neuronal Modeling: From Ions to Networks. MIT Press, Cambridge, MA (1998)
54. Komer, B., Stewart, T.C., Voelker, A.R., Eliasmith, C.: A neural representation of continuous space using fractional binding. In: 41st Annual Meeting of the Cognitive Science Society. Cognitive Science Society, Montreal (2019)
55. Lagorce, X., Benosman, R.: STICK: Spike time interval computational kernel, a framework for general purpose computation using neurons, precise timing, delays, and synchrony. Neural Comput. **27**(11), 2261–2317 (2015)
56. Legendre, A.-M.: Recherches sur l'attraction des sphéroïdes homogènes. Mémoires de Mathématiques et de Physique, présentés à l'Académie Royale des Sciences, pp. 411–435 (1782)
57. Lin, C.-K., Wild, A., Chinya, G.N., Lin, T.-H., Davies, M., Wang, H.: 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 (2018)
58. Lukoševičius, M.: A practical guide to applying echo state networks. In: Neural Networks: Tricks of the Trade, pp. 659–686. Springer, Berlin, Heidelberg (2012)
59. Lukoševičius, M.: Reservoir computing and self-organized neural hierarchies. Ph.D. thesis, Jacobs University Bremen (2012)
60. Lukoševičius, M., Jaeger, H.: Reservoir computing approaches to recurrent neural network training. Comput. Sci. Rev. **3**(3), 127–149 (2009)
61. Maass, W., Natschläger, T., Markram, H.: Real-time computing without stable states: A new framework for neural computation based on perturbations. Neural Comput. **14**(11), 2531–2560 (2002)
62. Mitra, P.P., Stark, J.B.: Nonlinear limits to the information capacity of optical fibre communications. Nature. **411**(6841), 1027 (2001)
63. Morcos, B., Stewart, T.C., Eliasmith, C., Kapre, N.: Implementing NEF neural networks on embedded FPGAs. In: 2018 International Conference on Field-Programmable Technology (FPT), pp. 22–29. IEEE (2018)
64. Mundy, A.: Real time Spaun on SpiNNaker. Ph.D. thesis, University of Manchester (2016)
65. Mundy, A., Knight, J., Stewart, T., Furber, S.: An efficient SpiNNaker implementation of the neural engineering framework. In: International Joint Conference on Neural Networks (IJCNN), pp. 1–8. IEEE (2015)
66. Neckar, A.: Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model. Ph.D. thesis, Stanford University (2018)
67. Neckar, A., Fok, S., Benjamin, B.V., Stewart, T.C., Oza, N.N., Voelker, A.R., Eliasmith, C., Manohar, R., Boahen, K.. Braindrop: A mixed-signal neuromorphic architecture with

a dynamical systems-based programming model. In: Proceedings of the IEEE (Accepted) (2019)
68. Nicola, W., Clopath, C.: Supervised learning in spiking neural networks with FORCE training. Nat. Commun. **8**(1), 2208 (2017)
69. Patel, K.P., Hunsberger, E., Batir, S., Eliasmith, C.: A spiking neural network for image segmentation. Neuromorphic Computing and Engineering (2020) (submitted)
70. Rall, W.: Distinguishing theoretical synaptic potentials computed for different soma-dendritic distributions of synaptic input. J. Neurophysiol. **30**(5), 1138–1168 (1967)
71. Rasmussen, D.: NengoDL: Combining deep learning and neuromorphic modelling methods. arXiv preprint arXiv:1805.11144 (2018)
72. Rieke, F., Warland, D.: Spikes: Exploring the Neural Code. MIT Press, Cambridge, MA (1997)
73. Rodrigues, O.: De l'attraction des sphéroïdes, Correspondence sur l'É-cole Impériale Polytechnique. Ph.D. thesis, Thesis for the Faculty of Science of the University of Paris (1816)
74. Roxin, A., Brunel, N., Hansel, D.: Role of delays in shaping spatiotemporal dynamics of neuronal activity in large networks. Phys. Rev. Lett. **94**(23), 238103 (2005)
75. Schäfer, A.M., Zimmermann, H.G.: Recurrent neural networks are universal approximators. In: International Conference on Artificial Neural Networks, pp. 632–640. Springer (2006)
76. Sharma, S., Aubin, S., Eliasmith, C.: Large-scale cognitive model design using the Nengo neural simulator. In: Biologically Inspired Cognitive Architectures, pp. 86–100. Elsevier B.V., Amsterdam (2016)
77. Singh, R., Eliasmith, C.: A Dynamic Model of Working Memory in the PFC During a Somatosensory Discrimination Task. In: Computational and Systems Neuroscience, Cold Spring Harbor Laboratory (2004)
78. Singh, R., Eliasmith, C.: Higher-dimensional neurons explain the tuning and dynamics of working memory cells. J. Neurosci. **26**, 3667–3678 (2006)
79. Stöckel, A., Eliasmith, C.: Passive nonlinear dendritic interactions as a computational resource in spiking neural networks. Neural Comput. **33**, 1–33 (2020)
80. Stöckel, A., Stewart, T.C., Eliasmith, C.: Connecting biological detail with neural computation: Application to the cerebellar granule-golgi microcircuit. In: 18th Annual Meeting of the International Conference on Cognitive Modelling. Society for Mathematical Psychology, Toronto (2020)
81. Stöckel, A., Voelker, A.R., Eliasmith, C.: Point Neurons with Conductance-Based Synapses in the Neural Engineering Framework. Technical Report. Centre for Theoretical Neuroscience, Waterloo (2017)
82. Stöckel, A., Voelker, A.R., Eliasmith, C.: Nonlinear synaptic interaction as a computational resource in the neural engineering framework. In: Cosyne Abstracts, Denver (2018)
83. Sussillo, D., Abbott, L.F.: Generating coherent patterns of activity from chaotic neural networks. Neuron. **63**(4), 544–557 (2009)
84. Sussillo, D., Barak, O.: Opening the black box: Low-dimensional dynamics in high-dimensional recurrent neural networks. Neural Comput. **25**(3), 626–649 (2013)
85. Thalmeier, D., Uhlmann, M., Kappen, H.J., Memmesheimer, R.-M.: Learning universal computations with spikes. PLoS Comput. Biol. **12**(6), e1004895 (2016)
86. Thorpe, S., Gautrais, J.: Rank order coding. In: Computational Neuroscience, pp. 113–118. Springer (1998)
87. Tripp, B., Eliasmith, C.: Neural populations can induce reliable postsynaptic currents without observable spike rate changes or precise spike timing. Cereb. Cortex. **17**(8), 1830–1840 (2006)
88. Voelker, A.R.: Dynamical systems in spiking neuromorphic hardware. Ph.D. thesis, University of Waterloo (2019)
89. Voelker, A.R., Benjamin, B.V., Stewart, T.C., Boahen, K., Eliasmith, C.: Extending the Neural Engineering Framework for nonideal silicon synapses. In: IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, Baltimore (2017)

90. Voelker, A.R., Eliasmith, C.: Methods and systems for implementing dynamic neural networks. US Patent App. 15/243,223 (patent pending) (2016)
91. Voelker, A.R., Eliasmith, C.: Methods for applying the neural engineering framework to neuromorphic hardware. arXiv preprint arXiv:1708.08133 (2017)
92. Voelker, A.R., Eliasmith, C.: Improving spiking dynamical networks: Accurate delays, higher-order synapses, and time cells. Neural Comput. **30**(3), 569–609 (2018)
93. Voelker, A.R., Eliasmith, C.: Legendre memory units in recurrent neural networks. PCT App. PCT/CA2020/00989 (patent pending) (2019)
94. Voelker, A.R., Gosmann, J., Stewart, T.C.: Efficiently Sampling Vectors and Coordinates from the n-Sphere and n-Ball. Technical Report. Centre for Theoretical Neuroscience, Waterloo (2017)
95. Voelker, A.R., Kajić, I., Eliasmith, C.: Legendre memory units: continuous-time representation in recurrent neural networks. In: Advances in Neural Information Processing Systems, pp. 15544–15553 (2019)
96. Voelker, A.R., Rasmussen, D., Eliasmith, C.. A spike in performance: Training hybrid-spiking neural networks with quantized activation functions. arXiv preprint arXiv:2002.03553 (2020)
97. Waernberg, E., Kumar, A.: Low dimensional activity in spiking neuronal networks. bioRxiv (2017)
98. Wallace, E., Maei, H.R., Latham, P.E.: Randomly connected networks have short temporal memory. Neural Comput. **25**(6), 1408–1439 (2013)
99. Wang, R., Hamilton, T.J., Tapson, J., van Schaik, A.: A compact neural core for digital implementation of the Neural Engineering Framework. In: Biomedical Circuits and Systems Conference (BioCAS), pp. 548–551. IEEE (2014)
100. Wang, R., Thakur, C.S., Cohen, G., Hamilton, T.J., Tapson, J., van Schaik, A.: A neuromorphic hardware architecture using the neural engineering framework for pattern recognition. IEEE Trans. Biomed. Circuits Syst. **11**(3), 574–584 (2017)
101. Wilson, M.A., Bower, J.M.: The simulation of large-scale neural networks. In: Methods in Neuronal Modeling, pp. 291–333. MIT Press, Cambridge, MA (1989)