
Learning efficient dynamical systems in recurrent spiking neural networks

Anonymous Author(s)

Affiliation

Address

email

Abstract

1 Recurrent neural networks (RNNs) are notoriously difficult to train and interpret.
 2 The paradigm of Reservoir Computing (RC) has managed to overcome some of
 3 these limitations by using randomly connected dynamical “reservoirs” of neurons.
 4 Despite the success of this approach, it is still difficult to interpret the representa-
 5 tions employed by the reservoir, or to incorporate prior knowledge into the reservoir
 6 to improve task performance. This ultimately leads to an inefficient use of neural
 7 resources. Here we show that the Neural Engineering Framework (NEF) provides
 8 a principled way of effectively optimizing the reservoir, by mapping desired dy-
 9 namics onto a latent representation within a factorable connection weight matrix.
 10 This enables us to train networks that outperform RC by over 4x on a delay line
 11 task, while reducing the cost of simulation by a factor of $O(n)$. This new approach
 12 readily incorporates synaptic dynamics, and supports various rate-based and spik-
 13 ing neuron models on analog and digital neuromorphic hardware. We demonstrate
 14 the potential of this approach with comparisons to Echo State Networks (ESNs)
 15 and Liquid State Machines (LSMs), and provide practical examples relevant to
 16 dynamic neural information processing.

17 1 Introduction

18 One of the central challenges in neural information processing is understanding how dynamic stimuli
 19 can be processed by neural mechanisms to drive behaviour. Recurrent connections, adaptation, and
 20 synaptic responses are ubiquitous sources of dynamics throughout the mammalian brain that work in
 21 concert to support dynamic information processing.

22 Recurrent neural networks (RNNs) employ nonlinear units that are interconnected to compute func-
 23 tions that combine their current input with the input *history* transmitted by the recurrent connections.
 24 While it is well known that RNNs can approximate any dynamical system arbitrarily well [10], they
 25 have encountered several limitations in practice. Chief among these is the difficulty of training RNNs
 26 at larger scales, due to the inherent challenge of globally optimizing the non-convex error function [2].

27 To bypass this problem, Reservoir Computing (RC) has taken a specialized approach to training
 28 RNNs. The two major variants of RC networks are Echo State Networks (ESNs; 12) and Liquid
 29 State Machines (LSMs; 15), which differ mainly by their use of rate-based and spiking neurons,
 30 respectively. RC networks make a conceptual separation between a randomly connected recurrent
 31 *reservoir* and a supervised feed-forward *readout*. The reservoir essentially extracts nonlinear temporal
 32 features of its input signal (referred to as *echoes*), and the readout learns how to combine these features
 33 to approximate the desired target. Since the readout is typically a linear combination of the reservoir
 34 units, the optimal solution can be learned via least-squares approximation. In contrast, the reservoir
 35 remains untrained with dynamical properties that are fixed independently of the desired task [14].

While RC solves the issue of training RNNs, the problem of efficient scaling remains. Indeed, as noted by RC theorists, “just simply creating a reservoir at random is unsatisfactory” and “setting up the reservoir such that a good state expansion emerges is an ill-understood challenge in many respects” [14]. Nevertheless, the computational power of RC generally relies on having a sufficiently large reservoir in order to represent a superset of the required features, ultimately resulting in an inefficient use of the neural substrate for certain functions. This is problematic considering the simulation cost of conventional RC networks scale as $O(n^2)$, where n is the number of neurons, thus making large-scale simulations of the brain computationally prohibitive.

The Neural Engineering Framework (NEF; 7, 8) offers a principled way forward by providing methods for mapping dynamical systems onto the *latent representations* of recurrently connected populations of neurons. This method of training the recurrent weights results in low-dimensional dynamics, which are commonly observed in biological systems [5], that can be readily understood using the mathematics of signal processing and control theory. The NEF has been used to implement various algorithms using rate-based and spiking neuron models on both analog [6, 4] and digital [11, 16] neuromorphic hardware.

We begin with a review of the NEF, and generalize the theory to characterize the effect of the synaptic model on the dynamical state of the network. We then proceed by further extending this theory within the context of RC. This is used to build a network that computes delayed versions of its input as accurately as ESNs [13], with a factor $O(n)$ speed-up in simulation time. The same network can then be made spiking, which significantly outperforms LSMs (over 4x in our experiments). We attribute these resource savings to having engineered prior knowledge of the desired low-dimensional function into the recurrent weight matrix.

We argue that this novel approach retains the benefits of RC, while enabling us to understand the representations being used by the network, the computations they afford, and therefore the dynamical systems that can be accurately realized. We demonstrate the generation of a nonlinear trajectory and the computation of a nonlinear filter (i.e., the power spectrum) as applications of these methods.

2 Structured recurrent neural networks

2.1 Neural Engineering Framework

The Neural Engineering Framework (NEF) is a set of three mathematical principles for describing neural computation [8], each of which is described below. The NEF is most commonly applied to building dynamic spiking neural networks, but also applies to non-spiking and feed-forward networks. Recently, the NEF has been used to build what remains the world’s largest functioning brain model [9], capable of performing perceptual, motor, and cognitive tasks. Here we give a brief overview of these methods applied to training both feed-forward and recurrent connection weights, and then extend this framework to account for synaptic dynamics and to incorporate ideas from RC.

2.1.1 Principle 1 – Representation

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

Specifically, we choose encoders $E \in \mathbb{R}^{n,k}$, gains $\alpha_i > 0$, and biases β_i as parameters for the encoding, which map $\mathbf{x}(t)$ to neural activities. These parameters can be chosen based on prior knowledge (e.g., tuning curves, firing rates, sparsity, etc.), or randomly. Most often, we select distributions from which to sample these parameters, incorporating known constraints on the system under consideration. For a spiking neuron, the encoding is:

$$\delta_i^{\mathbf{x}}(t) = G_i [\alpha_i \langle E_i, \mathbf{x}(t') \rangle + \beta_i : 0 \leq t' \leq t] \quad (1)$$

where $\delta_i^{\mathbf{x}}$ is the neural spike-train generated by the i^{th} neuron and $G_i[\cdot]$ is the model for a single neuron (e.g., a leaky integrate-and-fire (LIF) neuron, a conductance neuron, etc.). We then analytically determine the firing rates of each neuron under constant inputs:

$$r_i(\mathbf{v}) = \mathbb{E}_{t \geq 0} [\delta_i^{\mathbf{x}}; \mathbf{x}(t) = \mathbf{v}] \quad (2)$$

In the case of non-spiking neurons, we use its rate model in place of (2).

84 In biological systems, any transfer of information between layers of a network are affected by a
 85 post-synaptic response. Consequently we introduce a post-synaptic filter $h(t)$, referred to as the
 86 synapse model, that accounts for the temporal dynamics of a receiving neuron's synapse.
 87 For a purely representational relationship, we can then define the desired decoding as:

$$(\hat{\mathbf{x}} * h)(t) = \sum_{i=1}^n (\delta_i^{\mathbf{x}} * h)(t) D_i \quad (3)$$

88 To complete our characterization of a representation, we need to determine the linear decoders D_i .
 89 This determination is the same for the first and second principles, so we consider it next.

90 2.1.2 Principle 2 – Transformation

91 The second principle of the NEF addresses the issue of computing nonlinear functions of the vector
 92 space. The encoding remains as defined in (1). However, we may decode nonlinear functions of $\mathbf{x}(t)$
 93 by the same linear combination of temporal basis functions:

$$(\hat{f}(\mathbf{x}) * h)(t) = \sum_{i=1}^n (\delta_i^{\mathbf{x}} * h)(t) D_i^f \quad (4)$$

94 In general, we solve for the decoders $D^f \in \mathbb{R}^{n,l}$ to compute a desired nonlinear function $f : S \rightarrow \mathbb{R}^l$,
 95 where $S = \{\mathbf{x}(t) : t \geq 0\}$ is the domain of the signal – typically the unit k -ball $\{\mathbf{v} \in \mathbb{R}^k : \|\mathbf{v}\|_2 = 1\}$
 96 or the k -cube $[-1, 1]^k$. To account for fluctuations introduced by neural spiking, or other sources of
 97 uncertainty, we introduce a noise term $\eta \sim \mathcal{N}(0, \sigma^2)$ to regularize the solution:

$$D^f = \operatorname{argmin}_{D \in \mathbb{R}^{n,l}} \int_S \left[f(\mathbf{v}) - \sum_{i=1}^n (r_i(\mathbf{v}) + \eta) D_i \right]^2 d\mathbf{v} \quad (5)$$

98 Note this training only depends on $r_i(\mathbf{v})$ for $\mathbf{v} \in S$. It does not depend on $\mathbf{x}(t)$. Performing this
 99 regularized least-squares optimization provides the decoders needed by equations (3) and (4) to repre-
 100 sent the vector or compute transformations, respectively. We can use singular value decomposition
 101 (SVD) to characterize the class of functions that can be accurately approximated by the chosen basis
 102 functions [8: pp. 185–217].

103 The accuracy of this approach for spiking neurons relies on $r_i(\mathbf{v})$ being a suitable proxy for $\delta_i^{\mathbf{x}}$
 104 whenever $\mathbf{x}(t) = \mathbf{v}$. This is clearly true for constant \mathbf{x} , and is also a suitable model in practice for
 105 low-frequency $\mathbf{x}(t)$.

106 In general, we can use (1) and (4) to derive the connection weight matrix between layers that computes
 107 the desired function f . Specifically, the weight matrix $\mathcal{W} \in \mathbb{R}^{m,n}$ which decodes from neuron i and
 108 encodes into neuron j is given by:

$$\mathcal{W}_{ji} = \langle E_j, D_i^f \rangle = (E(D^f)^T)_{ji} \quad (6)$$

109 That is, the matrices D and E are simply a low-dimensional factorization of \mathcal{W} . This observation
 110 implies that the process of decoding and then encoding is equivalent to the more familiar expression
 111 for mapping activities between layers of neurons in a conventional neural network:

$$G[\alpha \mathcal{W} \delta^{\mathbf{x}}(t) + \beta] \quad (7)$$

112 The crucial difference is that a constant dimensionality k means $O(n)$ operations per time-step to
 113 implement (7) as compared to $O(n^2)$ operations for typical weight matrices, including those in RC.

114 In essence, structure is imposed by the choice of representing $\mathbf{x}(t)$ in neural activities, which in
 115 turn becomes latent within the factored weight matrix \mathcal{W} . Notably, since we are free to choose the
 116 representation, we do not lose computational power with this framework. Rather, this additional
 117 structure gives us a way to constrain the model by describing its state, and therefore understand in
 118 detail what the network is computing.

2.1.3 Principle 3 – Dynamics

Given our ability to compute nonlinear functions with Principle 2, the methods described here apply to both linear and nonlinear dynamical systems. However, for simplicity of exposition, we focus our discussion on linear time-invariant (LTI) systems:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= A\mathbf{x}(t) + B\mathbf{u}(t) \\ \mathbf{y}(t) &= C\mathbf{x}(t) + D\mathbf{u}(t)\end{aligned}\tag{8}$$

where the time-varying vector $\mathbf{x}(t)$ represents the system state, $\mathbf{y}(t)$ the output, $\mathbf{u}(t)$ the input, and the time-invariant matrices (A, B, C, D) fully define the system dynamics.

Principle 3 from the NEF states that in order to train the recurrent connection weights to implement (8) using a continuous-time lowpass filter $h(t) = \frac{1}{\tau}e^{-\frac{t}{\tau}}$, we use Principle 2 to train the decoders for the recurrent transformation $f(\mathbf{x}(t)) = (\tau A + I)\mathbf{x}(t)$, input transformation $f(\mathbf{x}(t)) = \tau B\mathbf{x}(t)$, output transformation $f(\mathbf{x}(t)) = C\mathbf{x}(t)$, and passthrough transformation $f(\mathbf{x}(t)) = D\mathbf{x}(t)$ [8; pp. 221–225]. This provides a method for training the network to implement any dynamical system. Here we show that this approach generalizes to other synaptic models.

For these purposes, the transfer function is a more useful description of the LTI system than (8). The transfer function is defined as the ratio of $Y(s)$ and $U(s)$, given by the Laplace transforms of $y(t)$ and $u(t)$ respectively. The variable s denotes a complex value in the frequency domain, while t is non-negative in the time domain. The transfer function is related to the state-space representation by the following:

$$F(s) = \frac{Y(s)}{U(s)} = C(sI - A)^{-1}B + D\tag{9}$$

A transfer function can be converted into a state-space representation using (9) if and only if it can be written as a proper ratio of finite polynomials in s . The ratio is proper when the degree of the numerator does not exceed that of the denominator. In this case, the output will not depend on future input, and so the system is *causal*. The order of the denominator corresponds to the dimensionality of \mathbf{x} , and therefore must be finite. Both of these conditions can be interpreted as physically realistic constraints where time may only progress forward, and neural resources are finite.

In order to account for the introduction of a synaptic filter $h(t)$, we replace the integrator s^{-1} with $H(s)$. This new system has the transfer function $C(H(s)^{-1}I - A)^{-1}B + D = F(H(s)^{-1})$. To compensate for this change in dynamics, we must invert the change of variables $s \leftrightarrow H(s)^{-1}$. This means finding the required $F'(s)$ such that $F'(H(s)^{-1})$ is equal to the desired transfer function, $F(s)$. Then the state-space representation of $F'(s)$ provides (A', B', C', D') which implement the desired dynamics. Thus the general problem reduces to solving this change of variables problem for various synaptic models. We now provide results for three common synaptic models. Complete derivations may be found in the supplementary material.

Continuous Lowpass Synapse Replacing the integrator with the standard continuous-time lowpass filter so that $H(s) = \frac{1}{\tau s + 1}$, gives:

$$F'(\tau s + 1) = F(s) \iff F'(s) = C(sI - (\tau A + I))^{-1}(\tau B) + D\tag{10}$$

which rederives the standard form of Principle 3 from the NEF [8].

Discrete Lowpass Synapse Replacing the integrator with a discrete-time lowpass filter $H(z) = \frac{1-a}{z-a}$ in the z -domain with time-step dt , where $a = e^{-\frac{dt}{\tau}}$, gives:

$$F'\left(\frac{z-a}{1-a}\right) = F(z) \iff F'(z) = \bar{C}\left(zI - \frac{1}{1-a}(\bar{A} - aI)\right)^{-1}\left(\frac{1}{1-a}\bar{B}\right) + \bar{D}\tag{11}$$

where $(\bar{A}, \bar{B}, \bar{C}, \bar{D})$ is the result of discretizing (A, B, C, D) with the same time-step. This permits an exact implementation of the desired system on digital hardware, even for large time-steps. Consequently, we use this method in the simulations reported in this paper unless otherwise noted.

158 **Delayed Continuous Lowpass Synapse** Replacing the integrator with a continuous lowpass filter
 159 with a pure delay of β seconds so that $H(s) = \frac{e^{-\beta s}}{\tau s + 1}$, gives:

$$F'(\frac{\tau s + 1}{e^{-\beta s}}) = F(s) \iff F'(s) = F(\frac{1}{\beta} W_0(ds) - \frac{1}{\tau}) \quad (12)$$

160 where $d = \frac{\beta}{\tau} e^{\frac{\beta}{\tau}}$ and $W_0(xe^x) = x$ is the principal branch of the Lambert W function. Such a
 161 synapse model can be used to account for possible transmission time-delays along presynaptic axons,
 162 or feedback delays within a broader context of control. To demonstrate the case when a pure delay
 163 of $F(s) = e^{-\alpha s}$ is the desired transfer function, we let $c = e^{\frac{\alpha}{\tau}}$ and $r = \frac{\alpha}{\beta}$ to obtain the required
 164 transfer function:

$$F'(s) = c(\frac{W_0(ds)}{ds})^r = c \sum_{i=0}^{\infty} \frac{r(i+r)^{i-1}}{i!} (-ds)^i \quad (13)$$

165 2.2 The NEF and Reservoir Computing

166 Reservoir Computing (RC) takes a different approach to training RNNs. Here we show that this
 167 alternative strategy may be leveraged by NEF networks (and vice-versa), by considering the two
 168 most prominent varieties of RC networks: Echo State Networks (ESNs; 12) and Liquid State
 169 Machines (LSMs; 15).

170 ESNs connect together sigmoidal activation units (most often \tanh) using a randomly generated
 171 weight matrix \mathcal{W} normalized by its spectral radius $\rho(\mathcal{W})$. This so-called “reservoir” of rate neurons
 172 essentially forms a discrete temporal feature representation of the input stimulus $\mathbf{u}(t)$. The activities
 173 are filtered using the same lowpass synapse $h(t)$ discussed earlier, where the rate of decay τ is a free
 174 parameter which alters the effective length of input history [14].

175 To compute a desired function $\mathbf{y}(t) = f[\mathbf{u}(t') : 0 \leq t' \leq t]$, a “readout” is trained from the activity
 176 of the reservoir using a feed-forward network. This readout is typically a linear matrix D , to facilitate
 177 efficient optimal training via regularized least-squares approximation. We refer to this method of
 178 explicit simulation followed by least-squares optimization as the “RC training method”, since LSMs
 179 also take the same fundamental approach, but employ models of continuous-time spiking neurons.

180 Within the NEF, the important distinction between ESNs and LSMs is captured simply by the choice
 181 of $G_i[\cdot]$ in equation (1). In either case, we can identify the neural activity of the output units in
 182 response to the input stimulus with $\delta_i^{\mathbf{x}}$, where $\mathbf{x}(t)$ is some *unknown* state that depends on both $\mathbf{u}(t)$
 183 and the dynamical properties of the reservoir. Note that characterizing $\mathbf{x}(t)$ is precisely the problem
 184 faced when trying to understand what RC networks are representing. Now the linear readout amounts
 185 to the following approximate nonlinear function of $\mathbf{x}(t)$:

$$\hat{\mathbf{y}}(t) = \sum_{i=1}^n (\delta_i^{\mathbf{x}} * h)(t) D_i \quad (14)$$

186 This has the same form as equation (3) from Principle 2. The important difference lies in how the
 187 decoders D are trained. Rather than solving for D using $r_i(\mathbf{v})$ in (5), the network is explicitly
 188 simulated to obtain exact values for the temporal basis functions. In general the RC training method
 189 relies on an explicit simulation since the form of $\mathbf{x}(t)$ is unknown.

190 From the perspective of the NEF, this method of training is more costly (especially as the time-step
 191 dt becomes small), but has the benefit of automatically *refining* the distortion error in the decoding
 192 induced by the rate-mode approximation (in the spiking case). In the rate-mode case, this is mainly
 193 beneficial as a conceptual tool, and for applications where the mapping between the state $\mathbf{x}(t)$
 194 and the desired target $\mathbf{y}(t)$ is either unknown or approximate (see Section 4.2 for an example).
 195 Similarly, as shown in Section 4.1, the readout can learn to compensate for approximation error or
 196 mischaracterization of the state variable.

197 From the perspective of RC, the NEF provides a way to solve for the recurrent connection weights
 198 by imposing a low-dimensional dynamical state $\mathbf{x}(t)$ on the reservoir. By using Principle 3 we can
 199 incorporate prior knowledge of the required function within this latent space, by “placing” temporal
 200 basis functions throughout the state-space. The NEF provides a way to understand these dynamics,

while still permitting the readout to capture nonlinear interactions between the state and the input (as characterized by Principle 2). Importantly, this reduces the cost of simulation by a factor of $O(n)$ for constant dimensionality k , since the number of connection weights for RC scale as $O(n^2)$.

3 Methods

3.1 Software

All networks were built and simulated using Nengo 2.1.0 [1], a Python tool for simulating large-scale neural networks including those built using the NEF. Benchmarks were run on Intel® Core™ i7-4770 CPU @ 3.40 GHz, using Numpy 1.10.4 and SciPy 0.17.0rc2 linked with the Intel Math Kernel Library (MKL).

Weights were trained and validated using randomly sampled band-limited white-noise. 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; 14). As well, 95% confidence intervals were bootstrapped and plotted using Seaborn [17]. The hyperparameters for all ESNs and LSMs (readout τ , recurrent τ , input gain, recurrent gain, and regularization σ^2) were found using Hyperopt [3] by optimizing the validation error across 200 random configurations containing 5 trials each. LSMs were implemented by replacing the tanh units with LIF spiking neurons, in the same manner as for NEF networks. Simulations used a 1 ms time-step (except Fig. 1) with Nengo’s default configurations for tuning curves and encoders.

3.2 Approximating a pure delay

To implement a pure delay of α seconds we must approximate the irrational transfer function $F(s) = e^{-\alpha s}$ as a ratio of finite-order polynomials. We do so using its *Padé approximants*,

$$[p/q]e^{-\alpha s} = \frac{Q_p(-\alpha s)}{Q_q(\alpha s)}, \quad Q_k(s) = \sum_{i=0}^k \binom{k}{i} \frac{(p+q-i)!}{(p+q)!} s^i \quad (15)$$

This gives the optimal approximation for a transfer function with order p in the numerator and order q in the denominator. After choosing suitable values for $p \leq q$, we numerically find a normalized state-space representation (A, B, C, D) that satisfies (9) using standard methods, and then implement this system given the synapse model using Principle 3. An expression for a normalized state-space representation when $p = q$ can be found in the supplementary material.

Fig. 1 demonstrates that we can implement the delay using spiking LIF neurons while analytically accounting for a continuous lowpass synapse ($\tau = 10$ ms) or a delayed lowpass synapse ($\tau = \beta = 10$ ms) using (10) or (13) respectively (to perform this comparison we do not use (11) here). This simulation uses $p = 5$ and $q = 6$. This shows that a synaptic delay can intuitively be “amplified” 10-fold while improving the NRMSE by a factor of 3 compared to the continuous lowpass. Remarkably, both networks use the same readout; only the recurrent connection weights are trained differently using the NEF.

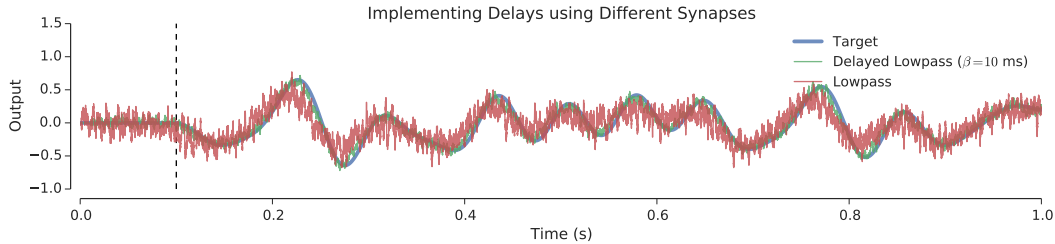


Figure 1: Computing a 100 ms delay of 15 Hz band-limited white-noise using a time-step of 0.01 ms ($10 \mu\text{s}$) in 2,000 spiking LIF neurons. Recurrent weights are trained for two different synapse models.

234 4 Results

235 4.1 Delay line benchmark

236 We now explore how the performance of NEF networks (using the RC training method) compare to
 237 traditional RC approaches. We consider the delay line task, in which we must compute a range of
 238 pure delays using the same reservoir. This is a natural way to measure the system’s ability to maintain
 239 history of its input signal, which is necessary when computing functions over past inputs. In fact,
 240 ESNs presented this task as one of its first examples within the RC paradigm [13].

241 We compare the short-term memory capacity of each approach (ESN, LSM, and NEF using various
 242 neuron models for $G_i[\cdot]$) by constructing a reservoir of 500 neurons, and then training separate linear
 243 readouts to compute delays ranging from 50–100 ms on 8 Hz band-limited white-noise.

244 For the NEF case, the prescribed dynamical system is a 100 ms delay using (15), yet the readouts are
 245 capable of computing many different delays without loss in accuracy (see Fig. 2). This demonstrates
 246 that the state variable is robust to changes in the target function (i.e., the prior simply biases which
 247 functions will be most accurate). The performance is comparable to that of ESNs for both LIF rate
 248 neurons and tanh rate neurons, and 4x (or more) better than LSMs for spiking LIF neurons.

249 These network were also simulated while varying the number of neurons from 5 to 2,500 in order to
 250 measure the real-time cost of simulation (see Fig. 3). We again note that traditional RC suffers from
 251 scaling issues since the recurrent weight matrices have $O(n^2)$ coefficients.

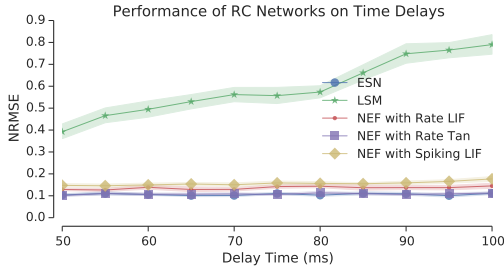


Figure 2: Performance from training each reservoir with 11 different linear readouts to compute delays between 50 – 100 ms. Performance is averaged across 20 trials.

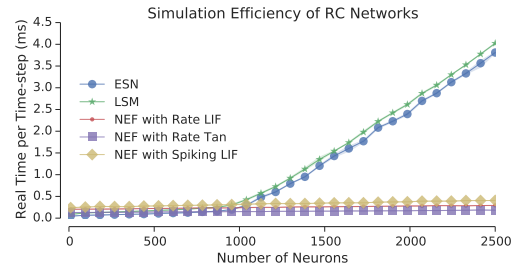


Figure 3: Cost of simulating each RC network for various numbers of neurons. Simulation time is averaged across 10 trials.

252 4.2 Autonomous trajectory generation

253 We now consider the task of generating a fixed trajectory over time by training an RC readout. Here
 254 we evenly space 5 delays apart by 100 ms with $p = 3$, $q = 4$, and $100 \cdot i$ spiking LIF neurons for
 255 the i^{th} delay. Connections use a lowpass synapse with time-constant $\tau = 50$ ms. The combined
 256 population of neurons then forms the reservoir that the readout uses, as in (14), to compute the desired
 257 trajectory (see Fig. 4). The input is a unit impulse that triggers the trajectory. The network is therefore
 258 *autonomous* in the sense that its responses are driven solely by internal state following each impulse.

259 4.3 Computing a spectrogram

260 The examples that we have seen thus far have been for relatively low-frequency inputs. Recall that in
 261 the spiking case the accuracy of the NEF relies on δ_i^x being well-approximated by a rate-model $r_i(\mathbf{v})$
 262 when decoding (3), which is a poor assumption in general for higher frequencies and lower numbers
 263 of neurons. For rate-based neuron models this is not an issue.

264 The transformations resulting from applying Principle 3 to the state-space representation of (15) are
 265 linear. Therefore, we can leverage the theory from Section 2.1 to compute continuous-time delays
 266 arbitrarily well, provided the methods of Section 3.2 are numerically stable with respect to the input
 267 frequency and the desired level of accuracy. We demonstrate this by sampling a sliding window of 20
 268 points spaced evenly apart by 20 ms (i.e., a sampling rate of 50 Hz). The sampling is implemented

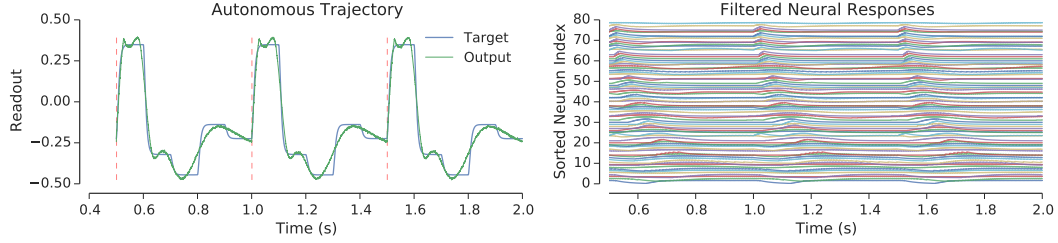


Figure 4: Autonomous activity of a trained network. (Left) The network receives a unit impulse every 500 ms and responds by autonomously generating a trajectory that connects a set of 5 discrete points spaced evenly apart by 100 ms. (Right) Filtered activity of 80 uniformly sampled neurons, sorted by time to peak activation.

269 using 20 independent delays with $p = q = 40$, $\tau = 1$ ms, and q linear neurons per delay with
 270 orthogonal encoders. This sliding window is then mapped by a linear transformation to compute
 271 the discrete Fourier transform (DFT) of the input history, which is then encoded by a population of
 272 22,000 LIF rate neurons to compute the nonlinear power by squaring each complex coefficient.

273 We test this on a randomly sampled 25 Hz band-limited white-noise signal (see Fig. 5). Notably,
 274 both the recurrent connection weights and the linear readout of this network have been trained quite
 275 accurately without explicitly simulating the network on any training signal.

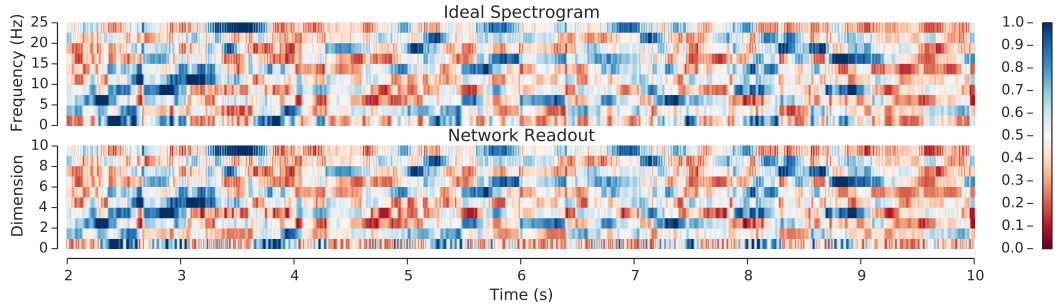


Figure 5: Simulation of a network trained to compute the power of its input at various frequencies using the DFT. Each element of the decoded vector corresponds to the power at a particular frequency.

276 5 Conclusions and future work

277 We have demonstrated that NEF-based methods significantly outperform RC networks for dynamic
 278 information processing with neural networks. In the rate neuron case, the accuracy is similar but
 279 simulation times are reduced by a factor of $O(n)$. In the spiking neuron case, the accuracy is 4x or
 280 better with the same improvement in simulation time.

281 The proposed methods demonstrate how to account for synaptic dynamics while accurately imple-
 282 menting system-level dynamics across a wide variety of synaptic models. Specifically, we showed
 283 how to exploit synaptic model properties (i.e., a small delay) to improve system-level performance
 284 (i.e., a long delay), which is a general challenge for dynamic information processing in neural
 285 networks. We further demonstrated that the same principles can be used to generate autonomous
 286 dynamics (i.e., movement between discrete points over an extended period). In addition, the methods
 287 described are efficient and accurate for both digital and analog implementations. This allowed us
 288 to demonstrate a spiking neural network computing a challenging nonlinear filter (i.e., the spectral
 289 power).

290 Conceptually speaking, these methods also provide a more intuitive understanding of the state being
 291 represented by the recurrent network, and the dynamics it is implementing. This understanding is
 292 partly a result of being able to specify a desired structure for the system. Some structures, like delays,
 293 are powerful constructs for implementing a wide class of useful dynamics.

References

- [1] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C Stewart, Daniel Rasmussen, Xuan Choo, Aaron Russell Voelker, and Chris Eliasmith. Nengo: a python tool for building large-scale functional brain models. *Frontiers in neuroinformatics*, 7, 2013.
- [2] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994.
- [3] James Bergstra, Daniel Yamins, and David Daniel Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning. ICML*, 2013.
- [4] Federico Corradi, Chris Eliasmith, and Giacomo Indiveri. 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.
- [5] John P Cunningham and M Yu Byron. Dimensionality reduction for large-scale neural recordings. *Nature neuroscience*, 17(11):1500–1509, 2014.
- [6] Julie Dethier, Paul Nuyujukian, Chris Eliasmith, Terrence C Stewart, Shauki A Elasaad, Krishna V Shenoy, and Kwabena A Boahen. A brain-machine interface operating with a real-time spiking neural network control algorithm. In *Advances in neural information processing systems*, pages 2213–2221, 2011.
- [7] Chris Eliasmith and Charles H Anderson. Developing and applying a toolkit from a general neurocomputational framework. *Neurocomputing*, 26:1013–1018, 1999.
- [8] Chris Eliasmith and Charles H Anderson. *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT press, 2003.
- [9] Chris Eliasmith, Terrence C Stewart, Xuan Choo, Trevor Bekolay, Travis DeWolf, Yichuan Tang, and Daniel Rasmussen. A large-scale model of the functioning brain. *science*, 338(6111):1202–1205, 2012.
- [10] Ken-ichi Funahashi and Yuichi Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural networks*, 6(6):801–806, 1993.
- [11] Francesco Galluppi, Christian Denk, Matthias Meiner, Terrence C Stewart, Luis Plana, Chris Eliasmith, Steve Furber, and Jorg Conradt. Event-based neural computing on an autonomous mobile platform. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, 2014.
- [12] Herbert Jaeger. The “echo state” approach to analysing and training recurrent neural networks. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148:34, 2001.
- [13] Herbert Jaeger. Short term memory in echo state networks. 2002.
- [14] Mantas Lukoševicius. *Reservoir computing and self-organized neural hierarchies*. PhD thesis, Jacobs University Bremen, 2012.
- [15] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.
- [16] Andrew Mundy, James Knight, Terrence C Stewart, and Steve Furber. An efficient spinnaker implementation of the neural engineering framework. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pages 1–8. IEEE, 2015.
- [17] Michael Waskom, Olga Botvinnik, Paul Hobson, Jordi Warmenhoven, John B. Cole, Yaroslav Halchenko, Jake Vanderplas, Stephan Hoyer, Santi Villalba, Eric Quintero, and et al. seaborn: v0.6.0, June 2015.