Methods for applying the Neural Engineering Framework to neuromorphic hardware

Aaron R. Voelker and Chris Eliasmith
Centre for Theoretical Neuroscience, University of Waterloo

August 29, 2017

Abstract

We review our current software tools and theoretical methods for applying the Neural Engineering Framework to state-of-the-art neuromorphic hardware. These methods can be used to implement linear and nonlinear dynamical systems that exploit axonal transmission time-delays, and to fully account for nonideal mixed-analog-digital synapses that exhibit higher-order dynamics with heterogeneous time-constants. This summarizes earlier versions of these methods that have been discussed in a more biological context (Voelker & Eliasmith, 2017) or regarding a specific neuromorphic architecture (Voelker et al., 2017).

1 Introduction

This report is motivated by the recent influx of neuromorphic computing architectures including SpiNNaker [1], Neurogrid [2], Brainstorm [3], IBM's TrueNorth [4], BrainScaleS [5], and ROLLS [6], among others. These are massively-parallel, low-power, analog, and/or digital systems (often a mix of the two) that are designed to simulate large-scale artificial neural networks rivalling the scale and complexity of real biological systems. Despite growing excitement, we believe that methods to fully unlock the computational power of neuromorphic hardware are lacking. This is primarily due to a theoretical gap between our traditional, discrete, *von Neumann*-like understanding of conventional algorithms and the continuous spike-based signal processing of real brains that is often emulated in silicon [7].

We use the term "neural compiler" to refer loosely to any systematic method of converting an algorithm, expressed in some high-level mathematical language, into synaptic connection weights between populations of spiking neurons [8]. To fully leverage neuromorphic hardware for real-world applications, we require neural compilers that can account for the effects of spiking neuron models and mixed-analog-digital synapse models, and, perhaps more importantly, exploit these details in useful ways when possible. There exist various approaches to neural engineering, including those by Denève et al. [9, 10] and Memmesheimer et al. [11]. However, the Neural Engineering Framework (NEF; [12, 13]) stands apart in terms of software implementation (Nengo; [8, 14, 15, 16]), large-scale cognitive modeling [17, 18], and neuromorphic applications [1, 2, 3, 19, 20, 21, 22, 23]. Competing methods consistently exclude important details, such as refractory periods and membrane voltage leaks, from their networks [9, 11]. The NEF, on the other hand, embraces biological complexity whenever it proves computationally useful [24] and/or improves contact

with neuroscience literature [25]. This approach to biological modeling directly mirrors a similar need to account for the details in neuromorphic hardware when building neural networks [19, 20, 26].

Nevertheless, there are many open problems in optimizing the NEF for state-of-the-art neuromorphics. In particular, we have been working to account for more detailed dynamics in heterogeneous models of the post-synaptic current (PSC) induced by each spike, as well as delays in spike propagation [24, 26, 27, 28]. The purpose of this report is to summarize our methods, both theoretical and practical, that have progressed in this direction. There are similar challenges in extending the NEF to account for multi-compartment neuron models [25, 29], conductance-based synapses [30], and to minimize the total number of spikes [7] – but these topics will not be addressed in this report.

The remainder of this report assumes that the reader is already comfortable with Nengo and the NEF in the context of engineering dynamical systems (i.e., "Principle III"). A technical yet accessible overview of the NEF can be found in [31], and a tutorial for Nengo (version 2) can be found in [15].

2 Accounting for Synaptic Dynamics

This section provides a theoretical account of the effect of higher-order linear synapse models on the dynamics of the network, by summarizing the extensions from [24] and [26]. This yields two novel proofs of Principle III from the NEF, and generalizes the principle to include more detailed synapses, including those modeling axonal transmission delays.

2.1 Linear systems

Here we focus our attention on linear time-invariant (LTI) systems:

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t)$$
$$\mathbf{y}(t) = C\mathbf{x}(t) + D\mathbf{u}(t) \tag{1}$$

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 "state-space" matrices (A, B, C, D) fully determine the system's dynamics. We will omit the variable t when not needed.

Principle III from the NEF states that in order to train the recurrent connection weights to implement (1)—using a continuous-time lowpass filter $h(t) = (1/\tau)e^{-t/\tau}$ to model the PSC—we use Principle II to train the decoders for the recurrent transformation $(\tau A + I)\mathbf{x}$, input transformation $\tau B\mathbf{u}$, output transformation $C\mathbf{x}$, and passthrough transformation $D\mathbf{u}$ [13, pp. 221–225]. This drives the recurrent synapses with the signal $\tau\dot{\mathbf{x}} + \mathbf{x}$ so that their output is the signal $\mathbf{x}$, in effect transforming the synapses into perfect integrators with respect to $\dot{\mathbf{x}}$. The vector $\mathbf{x}$ is then represented by the population of neurons via Principle I. Thus, this provides a systematic approach for training a recurrent neural network to implement any linear dynamical system. Now 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 (1). The transfer function is defined as the ratio of $Y(s)$ to $U(s)$, given by the Laplace transforms of $\mathbf{y}(t)$ and $\mathbf{u}(t)$ 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 (1) by the following:

$$F(s) = \frac{Y(s)}{U(s)} = C(sI - A)^{-1}B + D. \tag{2}$$

The transfer function $F(s)$ can be converted into the state-space model (A, B, C, D) 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.

Continuous Identity   In order to account for the introduction of a synaptic filter $h(t)$, we replace the integrator s^{-1} in (2) with $H(s)$, where $H(s) = \mathcal{L}\{h(t)\}$. 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* $F^H(s)$ such that $F^H(H(s)^{-1})$ is equal to the *desired* transfer function, $F(s)$. We highlight this as the following identity:

$$\boxed{F^H\left(\frac{1}{H(s)}\right) = F(s).} \tag{3}$$

Then the state-space model (A^H, B^H, C^H, D^H) satisfying (2) with respect to $F^H(s)$ will implement the desired dynamics (1) given $h(t)$.

Discrete Identity   For the discrete (i.e., digital synapse) case, we begin with $F(z)$ and $H(z)$ expressed as digital systems. The form of $H(z)$ is usually determined by the hardware, and $F(z)$ is usually found by a zero-order hold (ZOH) discretization of $F(s)$ using the simulation time-step (dt), resulting in the discrete LTI system:

$$\begin{aligned} \mathbf{x}[t + dt] &= \bar{A}\mathbf{x}[t] + \bar{B}\mathbf{u}[t] \\ \mathbf{y}[t] &= \bar{C}\mathbf{x}[t] + \bar{D}\mathbf{u}[t]. \end{aligned} \tag{4}$$

Here, we have the same relationship as (2),

$$F(z) = \frac{Y(z)}{U(z)} = \bar{C}(zI - \bar{A})^{-1}\bar{B} + \bar{D}. \tag{5}$$

Therefore, the previous discussion applies, and we must find an $F^H(z)$ that satisfies:

$$\boxed{F^H\left(\frac{1}{H(z)}\right) = F(z).} \tag{6}$$

Then the state-space model $(\bar{A}^H, \bar{B}^H, \bar{C}^H, \bar{D}^H)$ satisfying (5) with respect to $F^H(z)$ will implement the desired dynamics (4) given $h[t]$. In either case, the general problem reduces to solving this change-of-variables problem for various synaptic models. We now provide a number of results. More detailed derivations are available in [24].

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

$$F^H(\tau s + 1) = F(s) \iff F^H(s) = C(sI - (\tau A + I))^{-1}(\tau B) + D \tag{7}$$

which rederives the standard form of Principle III from the NEF [13].

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 $a = e^{-\frac{dt}{\tau}}$:

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

(8)

Therefore, $\bar{A}^H = \frac{1}{1-a}(\bar{A} - aI)$, $\bar{B}^H = \frac{1}{1-a}\bar{B}$, $\bar{C}^H = \bar{C}$, and $\bar{D}^H = \bar{D}$. This mapping can dramatically improve the accuracy of Principle III in digital simulations (e.g., when using a desktop computer) [24].

Delayed Continuous Lowpass Synapse Replacing the integrator with a continuous lowpass filter containing a pure time-delay of length λ, so that $H(s) = \frac{e^{-\lambda s}}{\tau s + 1}$:

$$F^H \left(\frac{\tau s + 1}{e^{-\lambda s}} \right) = F(s) \iff F^H(s) = F \left(\frac{1}{\lambda} W_0(ds) - \frac{1}{\tau} \right),$$

(9)

where $d = \frac{\lambda}{\tau} e^{\frac{\lambda}{\tau}}$ and $W_0(xe^x) = x$ is the principal branch of the Lambert-W function [32].[1] This synapse model can be used to model axonal transmission time-delays due to the finite-velocity propagation of action potentials, or to model feedback delays within a broader control-theoretic context. To demonstrate the case where a pure time-delay of length θ is the desired transfer function ($F(s) = e^{-\theta s}$), we let $c = e^{\frac{\theta}{\tau}}$ and $r = \frac{\theta}{\lambda}$ to obtain the required transfer function:

$$F^H(s) = c \left(\frac{W_0(ds)}{ds} \right)^r = cr \sum_{i=0}^{\infty} \frac{(i+r)^{i-1}}{i!}(-ds)^i.$$

(10)

We then numerically find the Padé approximants of the latter Taylor series. More details and validation may be found in [24] and [27].

General Case Finally, we consider any linear synapse model of the form:

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

(11)

for some polynomial coefficients (c_i) of arbitrary degree k. To the best of our knowledge, this class includes the majority of linear synapse models used in the literature. For synapses containing a polynomial numerator with degree $q > 1$ (e.g., considering the Taylor series expansion of the box filter $\epsilon^{-1}(1 - e^{-\epsilon s})s^{-1}$), we take its $[0, q]$-Padé approximants to transform the synapse into this form within some radius of convergence.[2] To map $F(s)$ onto (11), we begin by defining our solution to $F^H(s)$ in the form of its state-space model:

$$A^H = \sum_{i=0}^{k} c_i A^i, \qquad\qquad\qquad C^H = C,$$

$$B^H = \left(\sum_{j=0}^{k-1} s^j \sum_{i=j+1}^{k} c_i A^{i-j-1} \right) B, \qquad D^H = D,$$

(12)

---

[1]This assumes that $|\eta| < \pi$ and $\lambda \mathrm{Re}\,[s] + \frac{\lambda}{\tau} > -\eta \cot \eta$, where $\eta := \lambda \mathrm{Im}\,[s]$ [24].
[2]This is equivalent to the approach taken in [26, equations (9)-(11)].

which we claim satisfies (3). To validate this claim, we assert the following algebraic relationship (proven in [24]):

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

We now verify that (12) satisfies (3):

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

Since s^j is the jth-order differential operator, this form of B^H states that we must supply the jth-order input derivatives $\mathbf{u}^{(j)}$, for all $j = 1 \ldots k-1$. To be more precise, let us first define $B_j^H := \left(\sum_{i=j+1}^{k} c_i A^{i-j-1} \right) B$. Then (12) states that the ideal state-space model must implement the input transformation as a linear combination of input derivatives, $\sum_{j=0}^{k-1} B_j^H \mathbf{u}^{(j)}$. However, if the required derivatives are not included in the neural representation, then it is natural to use a ZOH method by assuming $\mathbf{u}^{(j)} = 0$, for all $j = 1 \ldots k-1$:

$$B^H = \left(\sum_{i=1}^{k} c_i A^{i-1} \right) B, \tag{13}$$

with A^H, C^H, and D^H as in (12). This is now an equivalent model to (12) assuming ZOH, and in the form of the standard state-space model (1).

The same derivation also applies to the discrete-time domain, with respect to the discrete synapse (corresponding to some implementation in digital hardware):

$$H(z) = \frac{1}{\sum_{i=0}^{k} \bar{c}_i z^i}. \tag{14}$$

Here, the only real difference (apart from notation) is the discrete version of (13):

$$\bar{B}^H = \left(\sum_{j=0}^{k-1} \sum_{i=j+1}^{k} \bar{c}_i \bar{A}^{i-j-1} \right) \bar{B}. \tag{15}$$

These mappings are made available by `ss2sim` and `LinearNetwork` in nengolib 0.4.0 [28]. Additional details may again be found in [24].

2.2 Nonlinear systems

Here we derive two theorems for nonlinear systems, by taking a different perspective that is consistent with §2.1. This generalizes the approach taken in [26], which considered the special case of a pulse-extended (i.e., time-delayed) double-exponential.

We wish to implement some desired nonlinear dynamical system,

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}, \mathbf{u}), \tag{16}$$

using (11) as the synaptic filter $h(t)$. Letting $\mathbf{w}(t) = f^h(\mathbf{x}, \mathbf{u})$ for some recurrent function f^h and observing that $\mathbf{x}(t) = (\mathbf{w} * h)(t)$, we may express these dynamics in the Laplace domain:

$$\frac{\mathbf{X}(s)}{\mathbf{W}(s)} = \frac{1}{\sum_{i=0}^{k} c_i s^i}$$

$$\Longleftrightarrow \qquad \mathbf{W}(s) = \mathbf{X}(s) \sum_{i=0}^{k} c_i s^i = \sum_{i=0}^{k} c_i \left[s^i \mathbf{X}(s) \right]$$

$$\Longleftrightarrow \qquad \mathbf{w}(t) = \sum_{i=0}^{k} c_i \mathbf{x}^{(i)}$$

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

Theorem 1. *Let the function computed along the recurrent connection be:*

$$\boxed{f^h(\mathbf{x}, \mathbf{u}) = \sum_{i=0}^{k} c_i \mathbf{x}^{(i)}} \qquad (17)$$

where $\mathbf{x}^{(i)}$ denotes the ith time-derivative of $\mathbf{x}(t)$, and c_i are given by (11). Then the resulting dynamical system is precisely (16).

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

$$\mathbf{x}[t + dt] = \bar{f}(\mathbf{x}, \mathbf{u}), \qquad (18)$$

using (14) as the synaptic filter $h[t]$, followed by an analogous theorem:

Theorem 2. *Let the function computed along the recurrent connection be:*

$$\boxed{\bar{f}^h(\mathbf{x}, \mathbf{u}) = \sum_{i=0}^{k} \bar{c}_i \mathbf{x}^{[i]}} \qquad (19)$$

where $\mathbf{x}^{[i]}$ denotes the ith discrete forwards time-shift of $\mathbf{x}$, and $\bar{c}_i$ are given by (14). Then the resulting dynamical system is precisely (18).

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

$$\frac{\mathbf{X}(z)}{\mathbf{W}(z)} = \frac{1}{\sum_{i=0}^{k} \bar{c}_i z^i}$$

$$\Longleftrightarrow \qquad \mathbf{W}(z) = \mathbf{X}(z) \sum_{i=0}^{k} \bar{c}_i z^i = \sum_{i=0}^{k} \bar{c}_i \left[z^i \mathbf{X}(z) \right]$$

$$\Longleftrightarrow \qquad \mathbf{w}[t] = \sum_{i=0}^{k} \bar{c}_i \mathbf{x}^{[i]}$$

since z is the forwards time-shift operator. $\qquad\qquad$ $\square$

Continuous Lowpass Synapse   For standard Principle III, we have $H(s) = \frac{1}{\tau s + 1}$ $\implies k = 1$, $c_0 = 1$ and $c_1 = \tau$,

$$\implies f^h(\mathbf{x}, \mathbf{u}) = c_0 \mathbf{x}^{(0)} + c_1 \mathbf{x}^{(1)} = \mathbf{x} + \tau \dot{\mathbf{x}} = \tau f(\mathbf{x}, \mathbf{u}) + \mathbf{x}. \tag{20}$$

Note that (20) is consistent with (7) and with Principle III from the NEF.

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

$$\implies \bar{f}^h(\mathbf{x}, \mathbf{u}) = \bar{c}_0 \mathbf{x}^{[0]} + \bar{c}_1 \mathbf{x}^{[1]} = (1-a)^{-1}(\bar{f}(\mathbf{x}, \mathbf{u}) - a\mathbf{x}). \tag{21}$$

Note that (21) is consistent with (8).

Continuous Double Exponential Synapse   For the double exponential synapse:

$$H(s) = \frac{1}{(\tau_1 s + 1)(\tau_2 s + 1)} = \frac{1}{\tau_1 \tau_2 s^2 + (\tau_1 + \tau_2)s + 1} \tag{22}$$

$$\implies f^h(\mathbf{x}, \mathbf{u}) = \mathbf{x} + (\tau_1 + \tau_2)\dot{\mathbf{x}} + \tau_1 \tau_2 \ddot{\mathbf{x}}$$
$$= \mathbf{x} + (\tau_1 + \tau_2)f(\mathbf{x}, \mathbf{u}) + \tau_1 \tau_2 \left(\frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}} \cdot f(\mathbf{x}, \mathbf{u}) + \frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}} \cdot \dot{\mathbf{u}} \right). \tag{23}$$

In the linear case, this simplifies to:

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

Linear Systems   As in §2.1, Theorems 1 and 2 require that we differentiate the desired dynamical system. For the case of nonlinear systems, this means determining the (possibly higher-order) Jacobian(s) of f, as shown in (23). For the special case of LTI systems, we can determine this analytically to obtain a closed-form expression. By induction it can be shown that:

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

Then by expanding and rewriting the summations:

$$f^h(\mathbf{x}, \mathbf{u}) = \sum_{i=0}^{k} c_i \mathbf{x}^{(i)}$$
$$= \sum_{i=0}^{k} c_i \left[A^i \mathbf{x} + \sum_{j=0}^{i-1} A^{i-j-1} B\mathbf{u}^{(j)} \right]$$
$$= \underbrace{\left(\sum_{i=0}^{k} c_i A^i \right)}_{\substack{\text{Recurrent} \\ \text{Matrix}}} \mathbf{x} + \sum_{j=0}^{k-1} \underbrace{\left(\sum_{i=j+1}^{k} c_i A^{i-j-1} \right)}_{\text{Input Matrices}} B\mathbf{u}^{(j)}. \tag{24}$$

The discrete case is identical:

$$\bar{f}^h(\mathbf{x}, \mathbf{u}) = \underbrace{\left(\sum_{i=0}^{k} \bar{c}_i \bar{A}^i \right)}_{\substack{\text{Recurrent} \\ \text{Matrix}}} \mathbf{x} + \sum_{j=0}^{k-1} \underbrace{\left(\sum_{i=j+1}^{k} \bar{c}_i \bar{A}^{i-j-1} \right) \bar{B}}_{\text{Input Matrices}} \mathbf{u}^{[j]}. \tag{25}$$

This gives a matrix form for any LTI system with a kth order synapse, provided we can determine $\mathbf{u}^{(j)}$ or $\mathbf{u}^{[j]}$ for $0 \le j \le k-1$. Again, (24) is consistent with (12) and (13), as is (25) with (15).

3   Accounting for Synaptic Heterogeneity

We now show how §2 can be applied to train efficient networks where the ith neuron has a distinct synaptic filter $h_i(t)$, given by:

$$H_i(s) = \frac{1}{\sum_{j=0}^{k_i} c_{ij} s^j}. \tag{26}$$

This network architecture can be modeled in Nengo using nengolib 0.4.0 [28]. This is particularly useful for applications to neuromorphic hardware, where transistor mismatch can change the effective time-constant(s) of each synapse. To this end, we abstract the approach taken in [26]. We show this specifically for Theorem 1, but this naturally applies to all methods in this report.

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

We can solve both issues simultaneously by taking advantage of the linear structure within f^{h_i} that is shared between all h_i. Considering Theorem 1, we need to drive the ith synapse with the function:

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

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

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

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

$$\hat{\mathbf{x}}^{(j)} = \left\langle \mathbf{a}, \mathbf{d}^j \right\rangle.$$

Now, putting this all together, we obtain:

$$\left\langle \mathbf{a}, \mathbf{d}^{f^{h_i}} \right\rangle = \sum_{j=0}^{k_i} c_{ij} \left\langle \mathbf{a}, \mathbf{d}^j \right\rangle = \sum_{j=0}^{k_i} c_{ij} \hat{\mathbf{x}}^{(j)} \approx f^{h_i}(\mathbf{x}, \mathbf{u}). \tag{27}$$

Therefore, we only need to solve $k+1$ optimization problems, decode the "matrix representation" $\left[\hat{\mathbf{x}}^{(0)}, \hat{\mathbf{x}}^{(1)}, \ldots, \hat{\mathbf{x}}^{(k)} \right]$, and then linearly combine these $k+1$ different decodings as shown in (27)—using the matrix of coefficients c_{ij}—to determine the input to each synapse. This approach reclaims the advantages of using factorized connection weight matrices, at the expense of a factor $\mathcal{O}(k)$ increase in space and time efficiency.

4   Discussion

We have reviewed three major extensions to the NEF that appear in recent publications. These methods can be used to implement linear and nonlinear systems in spiking neurons recurrently coupled with heterogeneous higher-order mixed-analog-digital synapses. This provides us with the ability to implement NEF networks in state-of-the-art neuromorphics while accounting for, and sometimes even exploiting, their nonideal nature.

While the linear and nonlinear methods can both be used to harness pure spike time-delays (due to axonal transmission) by modeling them in the synapse, the linear approach provides greater flexibility. Both extensions can first transform the time-delay into the standard form of (11) via Padé approximants, which maintains the same internal representation $\mathbf{x}$ as the desired dynamics (within some radius of convergence). But the linear extension also allows the representation to change, since it is only concerned with maintaining the overall input-output transfer function relation. In particular, we derived an analytic solution using the Lambert-W function, which allows the neural representation $\mathbf{x}$, and even its dimensionality, to change according to the expansion of some Taylor series. The linear case is also much simpler to analyze in terms of the network-level transfer function that results from substituting one synapse model for another. For all other results that we have shown, the linear extension is consistent with the nonlinear extension, as they both maintain the desired representation by fully accounting for the dynamics in the synapse.

Acknowledgements

References

[1] A. Mundy, J. Knight, T. C. Stewart, and S. Furber, "An efficient SpiNNaker implementation of the Neural Engineering Framework," in *The 2015 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2015.

[2] S. Choudhary, S. Sloan, S. Fok, A. Neckar, E. Trautmann, P. Gao, T. Stewart, C. Eliasmith, and K. Boahen, "Silicon neurons that compute," in *International Conference on Artificial Neural Networks (ICANN)*, pp. 121–128, Springer, 2012.

[3] "Projects - The neuromorphics project - Stanford University." `http://brainstorm.stanford.edu/projects/`. Accessed: 2017-08-12.

[4] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[5] J. Schemmel, D. Briiderle, A. Griibl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *IEEE International Symposium on Circuits and systems (ISCAS)*, pp. 1947–1950, IEEE, 2010.

[6] N. Qiao, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, and G. Indiveri, "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses," *Frontiers in neuroscience*, vol. 9, 2015.

[7] K. Boahen, "A neuromorph's prospectus," *Computing in Science & Engineering*, vol. 19, no. 2, pp. 14–28, 2017.

[8] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, D. Rasmussen, X. Choo, A. R. Voelker, and C. Eliasmith, "Nengo: A Python tool for building large-scale functional brain models," *Frontiers in Neuroinformatics*, vol. 7, no. 48, 2014.

[9] M. Boerlin, C. K. Machens, and S. Denève, "Predictive coding of dynamical variables in balanced spiking networks," *PLoS Comput Biol*, vol. 9, no. 11, p. e1003258, 2013.

[10] M. A. Schwemmer, A. L. Fairhall, S. Denève, and E. T. Shea-Brown, "Constructing precisely computing networks with biophysical spiking neurons," *The Journal of Neuroscience*, vol. 35, no. 28, pp. 10112–10134, 2015.

[11] D. Thalmeier, M. Uhlmann, H. J. Kappen, and R.-M. Memmesheimer, "Learning universal computations with spikes," *PLoS Comput Biol*, vol. 12, no. 6, p. e1004895, 2016.

[12] C. Eliasmith and C. H. Anderson, "Developing and applying a toolkit from a general neurocomputational framework," *Neurocomputing*, vol. 26, pp. 1013–1018, 1999.

[13] C. Eliasmith and C. H. Anderson, *Neural engineering: Computation, representation, and dynamics in neurobiological systems.* MIT press, 2003.

[14] T. C. Stewart, B. Tripp, and C. Eliasmith, "Python scripting in the Nengo simulator," *Frontiers in Neuroinformatics*, vol. 3, 2009.

[15] S. Sharma, S. Aubin, and C. Eliasmith, "Large-scale cognitive model design using the Nengo neural simulator," *Biologically Inspired Cognitive Architectures*, 2016.

[16] J. Gosmann and C. Eliasmith, "Automatic optimization of the computation graph in the Nengo neural network simulator," *Frontiers in Neuroinformatics*, vol. 11, p. 33, 2017.

[17] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen, "A large-scale model of the functioning brain," *science*, vol. 338, no. 6111, pp. 1202–1205, 2012.

[18] C. Eliasmith, *How to build a brain: A neural architecture for biological cognition.* Oxford University Press, 2013.

[19] J. Dethier, P. Nuyujukian, C. Eliasmith, T. C. Stewart, S. A. Elasaad, K. V. Shenoy, and K. A. Boahen, "A brain-machine interface operating with a real-time spiking neural network control algorithm," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 2213–2221, 2011.

[20] F. Corradi, C. Eliasmith, and G. 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.

[21] J. Knight, A. R. Voelker, A. Mundy, C. Eliasmith, and S. Furber, "Efficient SpiNNaker simulation of a heteroassociative memory using the Neural Engineering Framework," in *The 2016 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 07 2016.

[22] M. Berzish, C. Eliasmith, and B. Tripp, "Real-time FPGA simulation of surrogate models of large spiking networks," in *International Conference on Artificial Neural Networks (ICANN)*, 2016.

[23] A. Mundy, *Real time Spaun on SpiNNaker*. PhD thesis, University of Manchester, 2016.

[24] A. R. Voelker and C. Eliasmith, "Improving spiking dynamical networks: Accurate delays, higher-order synapses, and time cells," *(under review)*, 2017.

[25] C. Eliasmith, J. Gosmann, and X.-F. Choo, "BioSpaun: A large-scale behaving brain model with complex neurons," *ArXiv*, 2016.

[26] A. R. Voelker, B. V. Benjamin, T. C. Stewart, K. Boahen, and C. Eliasmith, "Extending the Neural Engineering Framework for nonideal silicon synapses," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, (Baltimore, MD), IEEE, 05 2017.

[27] A. R. Voelker and C. Eliasmith, "Methods and systems for implementing dynamic neural networks," *(patent pending)*, 07 2016.

[28] "Nengolib – Additional extensions and tools for modelling dynamical systems in Nengo." `https://github.com/arvoelke/nengolib/`. Accessed: 2017-08-12.

[29] P. Duggins, "Incorporating biologically realistic neuron models into the NEF," Master's thesis, University of Waterloo, Waterloo, ON, 2017.

[30] A. Stöckel, "Point neurons with conductance-based synapses in the Neural Engineering Framework," tech. rep., Centre for Theoretical Neuroscience, Waterloo, ON, 2017.

[31] T. C. Stewart, "A technical overview of the Neural Engineering Framework," tech. rep., Centre for Theoretical Neuroscience, Waterloo, ON, 2012.

[32] R. M. Corless, G. H. Gonnet, D. E. Hare, D. J. Jeffrey, and D. E. Knuth, "On the Lambert W function," *Advances in Computational mathematics*, vol. 5, no. 1, pp. 329–359, 1996.