Large-Scale Synthesis of Functional Spiking Neural Circuits

This paper reviews a system capable of performing multiple cognitive functions using a combination of biologically plausible spiking neurons, and an architecture that mimics the organization, function, and representational resources used in the mammalian brain.

By Terrence C. Stewart and Chris Eliasmith

ABSTRACT | In this paper, we review the theoretical and software tools used to construct Spaun, the first (and so far only) brain model capable of performing cognitive tasks. This tool set allowed us to configure 2.5 million simple nonlinear components (neurons) with 60 billion connections between them (synapses) such that the resulting model can perform eight different perceptual, motor, and cognitive tasks. To reverse-engineer the brain in this way, a method is needed that shows how large numbers of simple components, each of which receives thousands of inputs from other components, can be organized to perform the desired computations. We achieve this through the neural engineering framework (NEF), a mathematical theory that provides methods for systematically generating biologically plausible spiking networks to implement nonlinear and linear dynamical systems. On top of this, we propose the semantic pointer architecture (SPA), a hypothesis regarding some aspects of the organization, function, and representational resources used in the mammalian brain. We conclude by discussing Spaun, which is an example model that uses the SPA and is implemented using the NEF. Throughout, we discuss the software tool Neural ENGineering Objects (Nengo), which allows for the synthesis and simulation of neural models efficiently on the scale of Spaun, and provides support for constructing models using the NEF and the SPA. The resulting NEF/SPA/Nengo combination is a general tool set for both evaluating hypotheses about how the brain works, and for building systems that compute particular functions using neuron-like components.

Manuscript received October 9, 2013; revised December 21, 2013 and February 10, 2014; accepted February 11, 2014. Date of publication March 27, 2014; date of current version April 28, 2014.

The authors are with the Centre for Theoretical Neuroscience, University of Waterloo, Waterloo, ON N2L 3G1 Canada (e-mail: celiasmith@uwaterloo.ca).

Digital Object Identifier: 10.1109/JPROC.2014.2306061

KEYWORDS | Neural computation; neural engineering framework (NEF); neural modeling; neuromorphic engineering; semantic pointer architecture (SPA); Spaun; spiking neural networks

I. INTRODUCTION

In this paper, we describe the methodology and tools we have developed for building large-scale systems from simulated spiking neurons. In particular, we describe two theoretical tools [the neural engineering framework (NEF) and the semantic pointer architecture (SPA)] and one software suite [Neural ENGineering Objects (Nengo)] that we used to construct what is currently the world's first *functional* brain model (i.e., one that is capable of performing a variety of cognitive tasks). This model, which we refer to as the semantic pointer architecture unified network (or Spaun), consists of 2.5 million simulated spiking neurons whose properties and interconnections are consistent with those found in the human brain. The model receives input in the form of digital images on a virtual retina and produces output that controls a simulated arm. With this framework, Spaun is able to perform eight different tasks, including digit recognition, serial working memory, pattern completion, mental arithmetic, and question answering. Furthermore, it is able to switch between these tasks based on its own visual input, meaning that there are no external modifications made to the network between tasks. This sort of cognitive flexibility is a hallmark of cognitive systems, but it is difficult to achieve with traditional neural modeling approaches.

The major goal of this research is to understand how the brain works by reverse-engineering it. We do this by trying to build biologically *plausible* models of cognitive processes. For us, these are models where the individual components (simulated neurons) can be made as similar to real neurons as desired, and where the large-scale anatomy and connectivity of the brain is respected. While the work described here uses the leaky-integrate-and-fire (LIF) model of a neuron (the simplest and most common neuron model that produces spikes), all of the techniques apply to more detailed neural models. Different neurons in different brain areas have different properties, and we use this neurological data to constrain our models. While we do not argue that this is the *only* way to build such models, we believe that the NEF is a general tool for implementing a very large set of algorithms in components like neurons, and that the SPA is one particular algorithm that can be implemented with the NEF and that, we believe, is quite promising for matching human performance.

As a side effect of this reverse engineering goal, the NEF provides a methodology for biomimetic computation. That is, it shows how to connect large numbers of very simple components (neurons) with potentially stochastic behaviors acting in parallel to compute functions of the form $\mathbf{y} = f(\mathbf{x})$ and $d\mathbf{x}/dt = f(\mathbf{x}, \mathbf{u})$ where $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{u}$ are vectors. The individual components need not be exactly like neurons. The only requirements are that the components sum their inputs and that there is a low-pass filter on each connection. This leads to the ability to perform useful computation with a very different type of component than is seen in conventional computing.

We begin in Section II with the NEF, the "neural compiler" that takes a vector-based description of a system (and its dynamics) and converts it into a network of interacting components (in this case, spiking neurons). In Section III, we describe the SPA, a method for taking cognitive algorithms and converting them into vector-based descriptions consistent with mammalian neurobiological constraints. Sections II and III both end with descriptions of our open-source software Nengo that implements these ideas. Finally, in Section IV, we show how these tools work and can be used in concert to produce Spaun. Material throughout this paper is adapted from [16], [19], and [51].

II. THE NEURAL ENGINEERING FRAMEWORK

The NEF is a general-purpose system for taking algorithms and implementing them using components such as spiking neurons [17]. This can be thought of as a "neural compiler" where algorithms written in a high-level language are converted into neurons with connections between them. This compilation process works for arbitrary neuron types, and can be constrained in biologically realistic ways. Importantly, the high-level algorithms must be expressed in terms of vectors and functions on those vectors (including ordinary differential equations). The resulting neural networks approximate the desired functions, and the error of this approximation can be made arbitrarily small by increasing the number of neurons. This makes the NEF ideal for expressing algorithms typically seen in domains such as control theory, and determining their relevance to brain function.

While the NEF can be used to build arbitrary abstract systems such as controlled attractor networks [15], we have primarily used it to show how particular capabilities found in real animals might be implemented biologically. This has included path integration in rodents [11], working memory [57] and arm movements [13] in monkeys, and decision making in rats [36], [40] and humans [39]. We have also taken into account biological constraints such as Dale's principle [45] and incorporated biologically realistic learning rules to construct these networks [6], [42]. Several overviews of the NEF are available [14], [15], [19], [62]. The rest of this section serves as a summary of the three main principles within the NEF, plus our software tool Nengo.

A. Principle 1: Representation

The core of the NEF is the idea that groups of neurons represent vectors, and connections between groups of neurons compute functions on those vectors. The first NEF principle shows how the activity of a group of neurons can be said to represent a vector, and how changes in the activity of those neurons corresponds to changes in the represented vector.

We start with the notion of a "preferred direction vector." In the brain, many neurons have a particular stimulus (or response) for which they will fire most strongly. As the stimulus (or response) changes to become less similar to the preferred vector, the neuron will fire less quickly. This was originally identified in the motor system [26] and has since been seen in the head direction system [69], the visual system [54], and the auditory system [20]. For a more detailed exploration of this idea, see [63].

For the NEF, we generalize this concept to all neural populations. In particular, we quantify it by stating that the total current going into a neuron will be proportional to the dot product of the vector to be represented $\mathbf{x}$ and the preferred direction vector for the neuron $\mathbf{e}_i$ (plus a constant bias term). The response of a neuron i for any given input vector $\mathbf{x}$ is thus

$$\delta_i(\mathbf{x}) = G_i\left[\alpha_i \mathbf{e}_i \mathbf{x} + J_i^{\text{bias}}\right] \tag{1}$$

where δ_i is the spiking output of the neuron, G_i is the neuron model, α_i is a randomly chosen gain term, $\mathbf{e}_i$ is the preferred direction vector, and J_i^{bias} is a randomly chosen fixed background current. We use $\mathbf{e}$ for *encoder* to indicate that (1) captures a transformation between spaces: $\mathbf{x}$ is encoded in the activity space of the neurons. Fig. 1 shows this encoding for the case where $\mathbf{x}$ is 2-D and the neural

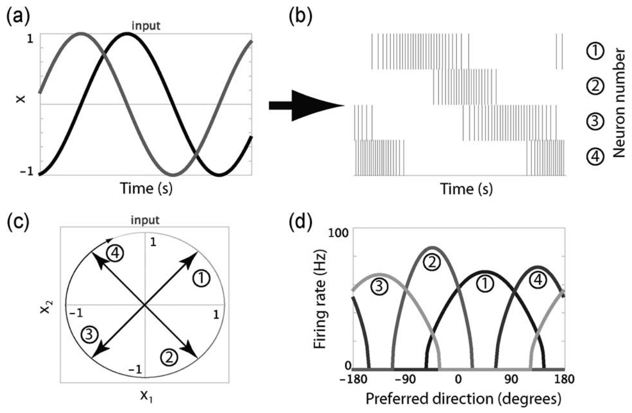

Fig. 1. ***NEF encoding of a 2-D signal using four neurons. (a) The input signal $x_1 = \sin(6t)$ (black), $x_2 = \cos(6t)$ (gray) over 1.2 s. (b) The spikes generated by the neurons when driven by the input in (a). (c) The same input shown in the vector space. The path of the input is a unit circle. Older inputs are in progressively lighter gray. The preferred direction vectors e_i of all four neurons are also shown. (d) The firing rates of the four neurons for different inputs around the unit circle (firing rates). Gains α_i and biases J_i^{bias} are randomly chosen. (Figure reproduced from [19] with permission.)***

population consists of four neurons. Importantly, the NEF applies to a large variety of neuron models (including both spiking and nonspiking models) since it makes no commitment to a specific function G (whose input is the total current flowing into the neuron). The LIF model is a common choice, for reasons of computational efficiency (and is used in Fig. 1), but a wide variety of neural models work with the NEF.

Given an encoding operation, it is natural to define a decoding operation, in order to characterize the information processing characteristics of the system (in this case, an ensemble of neurons). To decode a continuous estimate of the input from neural activity, the NEF focuses on the postsynaptic filtered activity generated by the reception of a spike at a synapse. This activity is taken to be a linear filter applied to the spiking activity

$$a_i(\mathbf{x}) = \sum_j h_i(t) * \delta_i\big(t - t_j(\mathbf{x})\big)$$

where $h_i(t)$ is the synaptic response function (usually a decaying exponential) with a time constant τ_{PSC} determined by a neurotransmitter type at the synapse, "$*$" is the convolution operator, and $\delta_i(t - t_j(\mathbf{x}))$ is the spike train produced by neuron i in response to input $\mathbf{x}$, with spike times indexed by j. Having defined this continuous variable, which is equal to the spike rate as $\tau_{\text{PSC}} \to \infty$, we can specify a decoding operation for estimating the input $\mathbf{x}$.

For reasons that will be apparent in a moment, we use a *linear* decoder

$$\hat{\mathbf{x}} = \sum_i^N a_i(\mathbf{x})\mathbf{d}_i \tag{2}$$

where N is the number of neurons in the group, $\mathbf{d}_i$ are the linear decoders, and $\hat{x}$ is the estimate of the original $\mathbf{x}$ value that produced the neural activity (1).

Any optimization method can be used to find these decoders. The simplest is to use standard least squares optimization

$$\underset{\mathbf{d}_i}{\arg\min} \int \left[\mathbf{x} - \sum_i^N a_i(\mathbf{x})\mathbf{d}_i\right]^2 d\mathbf{x} \tag{3}$$

where $\mathbf{d}_i$ are the decoding vectors over which this error is minimized and the integral is over all $\mathbf{x}$ values. It has been shown that linear decoders are sufficient to decode ∼95% of the information available in a single spike train generated by a stimulus [53]. Furthermore, as the number of neurons N increases, the mean squared error decreases as $1/N$. The NEF decoding process is depicted in Fig. 2, where the optimal linear decoders have been found and

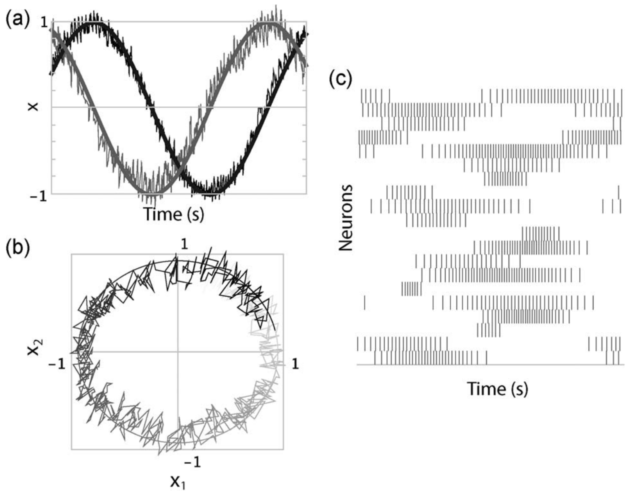

Fig. 2. ***NEF decoding of a 2-D signal using 20 neurons. Inputs are the same as in Fig. 1. (a) The original input and the decoded estimate over 1.2 s (black is x1; gray is x2). (b) The same data shown in the vector space. Older states are lighter gray. For both (a) and (b), smooth lines represent the ideal x values, while noisy lines represent the estimate $\hat{\mathrm{x}}$. (c) The spikes generated by the 20 neurons during the simulation, used to generate the decodings shown in (a) and (b). (Figure reproduced from [19] with permission.)***

used for 20 neurons. Temporal decoding is also linear, as described, and is performed using the postsynaptic, current-based filters (see Section II-C for further discussion).

While linear decoders are useful for visualizing the information encoded within the activity of a group of neurons, they also provide a direct way to solve for connection weights between groups of neurons. This is a key advantage of the NEF: rather than using a learning rule to optimize over the entire space of all connection weights, we instead solve the simpler problem of optimizing over the space of linear decoders, and then use that result to solve for the connection weights. Importantly, given the characterization of neural activity with preferred direction vectors, there is no difference between using this smaller space and using the equivalent full connection matrix. Interestingly, this general technique has started to be applied in broader domains; see [68] for an overview.

For example, if a connection between neural groups is meant to compute the identity function $\mathbf{y} = \mathbf{x}$ (where $\mathbf{y}$ is the vector space represented by the second population B and $\mathbf{x}$ is the vector space represented by the first population A), the connections between individual neurons are given by

$$\omega_{ij} = \alpha_j \mathbf{d}_i \mathbf{e}_j \tag{4}$$

where i indexes the neurons in group A and j indexes the neurons in B. Behavior of this network is shown in Fig. 3(a).

While the least squares method for optimization we use here is not biologically plausible on its own, we have also shown that biologically realistic learning rules will converge on a similar solution [42]. These realistic rules are, however, several orders of magnitude more computationally expensive.

B. Principle 2: Transformation

Connections between groups of neurons can also compute functions other than the identity function. That is, instead of $\mathbf{y} = \mathbf{x}$, we can do $\mathbf{y} = f(\mathbf{x})$. We do this by finding decoders $\mathbf{d}_i^f$ for the particular function $f(\mathbf{x})$ by substituting $f(\mathbf{x})$ for $\mathbf{x}$ in (3)

$$\underset{\mathbf{d}_i^f}{\arg\min} \int \left[f(\mathbf{x}) - \sum_i a_i(\mathbf{x}) \mathbf{d}_i^f \right]^2 d\mathbf{x}. \tag{5}$$

The connection weights can then be computed using (4). For the special case of linear functions $\mathbf{y} = \mathbf{L}\mathbf{x}$, rather than solving for a new decoder, we can simply put $\mathbf{L}$ directly into the weight equation itself, resulting in

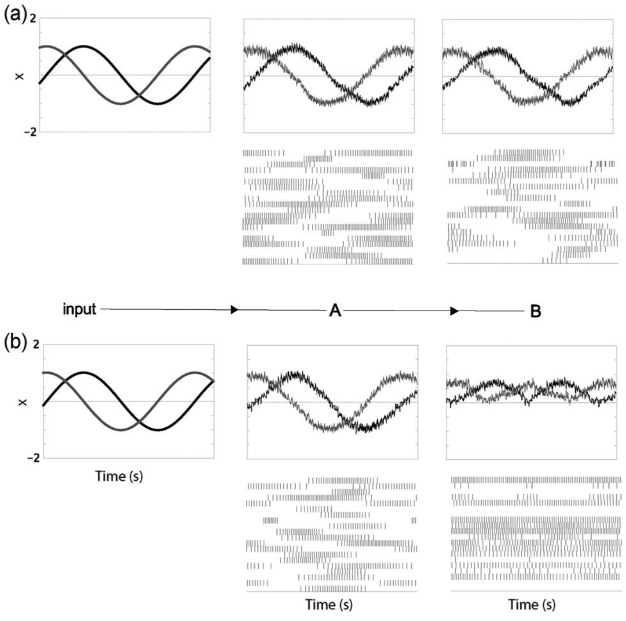

Fig. 3. ***Connecting neurons using the NEF. (a) Computing the identity function between A and B. (b) Computing the elementwise square between A and B. Simulations are 1.2 s long, and all populations have 20 neurons with randomly chosen encoders*** e_i***, gains*** α_i***, and biases*** J_i^{bias}***. (Figure reproduced from [19] with permission.)***

$\omega_{ij} = \alpha_j \mathbf{d}_i \mathbf{L} \mathbf{e}_j$. Combining these two approaches, the neural connection weights needed to approximate the function $\mathbf{y} = \mathbf{L} f(\mathbf{x})$ are

$$\omega_{ij} = \alpha_j \mathbf{d}_i^f \mathbf{L} \mathbf{e}_j. \tag{6}$$

Fig. 3(b) shows the computation of the elementwise square $(f(\mathbf{x}) = [x_1^2, x_2^2])$.

C. Principle 3: Dynamics

While the first two principles are sufficient to build neural approximations of any desired function of the vector $\mathbf{x}$, the NEF also provides a method for computing functions of the form $\mathrm{d}x/\mathrm{d}t = f(\mathbf{x}, \mathbf{u})$, where $\mathbf{u}$ is the input from some other population. We do this by exploiting the fact that neurons do not simply accept input as spikes. Rather, when a spike is transmitted from one neuron to another, the actual current that flows into the second neuron is a low-pass-filtered version of that spike. In particular, the postsynaptic current is well approximated by $h(t) = u(t)e^{-t/\tau}$, where $u(t)$ is the step function and τ is the time constant of the neurotransmitter used. This time constant varies throughout the brain, ranging from 2–5 ms [35] up to ~100 ms [55]. The effect of this filter is that instead of a connection computing the

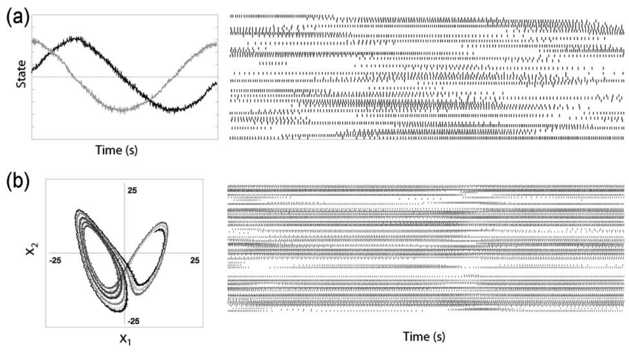

Fig. 4. ***Two dynamical systems implemented with the NEF. (a) A simple linear harmonic oscillator using 200 neurons. (b) A nonlinear dynamical system, specifically a chaotic Lorenz attractor, using 2000 neurons. Both are recurrently coupled populations of neurons, and the NEF is used to compute the coupling connections to implement the two different dynamical systems.***

function $\mathbf{y}(t) = f(\mathbf{x}(t))$ it will compute $\mathbf{y}(t) = f(\mathbf{x}(t)) * h(t)$ [or, in the Laplace domain, $\mathbf{Y}(s) = \mathbf{F}(s)H(s)$].

It turns out that this implicit low-pass filter can be used to generate neural models that compute arbitrary dynamics. Given a neural population representing $\mathbf{x}$, an input $\mathbf{u}(t)$, and a feedback connection from $\mathbf{x}$ back to itself computing $g(\mathbf{x}(t))$, we note that in the Laplace domain, we get $\mathbf{X}(s) = (\mathbf{G}(s) + \mathbf{U}(s))H(s)$. Since the Laplace transform of $h(t) = u(t)e^{-t/\tau}$ is $H(s) = 1/(1 + s\tau)$, we can rearrange this to get $s\mathbf{X}(s) = (\mathbf{G}(s) - \mathbf{X}(s))/\tau + \mathbf{U}(s)/\tau$. Converting back to the time domain, $(\mathrm{dx}/\mathrm{dt}) = (g(\mathbf{x}(t)) - \mathbf{x}(t)/\tau) + (\mathbf{u}(t)/\tau)$. Thus, if we desire the dynamics $\mathrm{dx}/\mathrm{dt} = f(\mathbf{x}(t)) + \mathbf{u}(t)$, we introduce a feedback connection that uses the previous two NEF principles to find connection weights that compute $g(\mathbf{x}(t)) = \tau f(\mathbf{x}) + \mathbf{x}$, and we scale the input $\mathbf{u}(t)$ by τ. For arbitrary dynamics of the form $\mathrm{dx}/\mathrm{dt} = f(\mathbf{x}, \mathbf{u})$, we do a change of variables $\mathbf{x}' = \langle \mathbf{x}, \mathbf{u} \rangle$.

This exploitation of the inherent first-order low-pass filter found in synaptic connections allows for the implementation of a very wide variety of systems, including oscillators, integrators, and arbitrary attractor networks [15]. Fig. 4(a) shows a single neural population with a recurrent feedback connection computing the standard linear oscillator $\mathrm{dx}/\mathrm{dt} = [-x_2, -x_1]$. Here, the postsynapic time constant τ is 100 ms, but the period of oscillation is 2π ($\sim$6.28 s). Importantly, we achieve different periods of oscillation without changing the neural property τ; rather, we change the function being computed by the feedback connection, resulting in a different set of connection weights. For example, we could compute $\mathrm{dx}/\mathrm{dt} = [-x_2/2, -x_1/2]$, giving a period of π, instead. This same method works for nonlinear functions as well; in Fig. 4(b), we show the classic Lorenz attractor $\mathrm{dx}/\mathrm{dt} = [10(x_2 - x_1), x_1(28 - x_3) - x_2, x_1x_2 - (8/3)x_3]$.

This approach allows for the construction of neural models that correspond to a very large family of functions, including those typically employed by the modern control theory and the dynamic systems theory.

It should be noted that networks constructed with this approach are fundamentally different from echo state networks [29] and liquid state machines [41]. In both of those approaches, the recurrent connections are randomly chosen, and then a linear combination of the outputs are found that compute the desired function. With the NEF, we also find a linear combination of the outputs ($\mathbf{d}_i$), but we solve for the ideal recurrent connection weights to achieve the desired dynamics. This makes the NEF much more efficient and capable of computing a much wider range of dynamics.

D. Neural ENGineering Objects (Nengo)

These three principles are sufficient to implement all of our neural models. However, to simplify the process of constructing these models, we have developed an open-source software package known as Neural ENGineering Objects (Nengo) that creates and runs these models. Models can be built in Nengo using a drag-and-drop graphical user interface or specified using the Python scripting language. Full details and documentation can be found online at http://nengo.ca, and in other publications [65].

For example, to create the model shown in Fig. 3(a), we use the following script.

```
net = nef.Network('Identity Function')
net.make('A', neurons = 20, dimensions = 1)
net.make('B', neurons = 20, dimensions = 1)
net.connect('A','B')
```

For Fig. 3(b), we need to compute the elementwise square. This is specified in a Python function as follows.

```
net = nef.Network('Elementwise Square')
net.make('A', neurons = 20, dimensions = 1)
net.make('B', neurons = 20, dimensions = 1)
def square(x) :
   return x[0] * x[0], x[1] * x[1]
net.connect('A','B', func = square)
```

Nengo will automatically solve for the connection weights that will best approximate the provided function.

Nengo also provides an interactive interface for displaying the results of a simulation while it is running, allowing for real-time interaction with a running model. This interface allows the generated plots to be exported, and was used to produce Figs. 1–5. Furthermore, Nengo scales up to our largest models: the 2.5 million neuron Spaun model is run in Nengo, and can be downloaded at http://models.nengo.ca/spaun.

III. THE SEMANTIC POINTER ARCHITECTURE

While the NEF specifies how to convert vector-based algorithms into spiking neural networks, a separate theory is needed to describe cognitive function in terms of vector-based algorithms. Our approach to this problem is called the semantic pointer architecture (SPA). While a full description can be found elsewhere [16], the core of our approach is to suggest a vector-based cognitive architecture: i.e., a set of basic functional components, each of which can be defined in terms of vector operations, and an organization of those components that can work together to implement cognitive algorithms. In addition to these components, we provide a hypothesis as to how structured representations (like sentences) can be represented using vectors and what basic operations need to be performed on those vectors to achieve memory, planning, pattern matching, and other behaviors. We refer to our proposed form of neurally plausible representation as "semantic pointers."

A. Structure

A central concern for modeling cognitive processing using neurons (or vectors) is how to effectively represent structured information. Structure is vital to explanations of cognitive behavior [2]. As an example, consider the sentence "cats chase mice." If this is to be represented as a vector, then we need to represent it in such a way that "cats chase mice" is different from "mice chase cats." In an artificial language, like those typically used in computers, such a phrase may be represented with a structured representation like $\texttt{chase}(\texttt{cats}, \texttt{mice})$. The majority of theories that attempt to explain human cognition rely on the ability to store and manipulate these representations. However, the problem of how neurons could possibly perform such manipulations has been a long-standing problem in cognitive science [21].

The approach we present here is speculative, but it is the only approach we know of that is both flexible enough for us to develop large-scale cognitive models and implementable within known biological constraints [16], [58]. This approach is based on two different compression operators [47], [63]. The first of these is simple vector addition. This takes in two vectors and produces a single new vector as output. If all the terms to be combined together are themselves vectors, then we could write the full sentence as follows, where terms in bold are particular vectors for each concept:

$$\texttt{cats} + \texttt{chase} + \texttt{mice}.$$

However, this cannot serve to represent structure, since with this approach "cats chase mice" would be exactly equal to "mice chase cats." As a result, we need a second "binding" operator: an operator that takes two vectors as input and produces a third that is very dissimilar to the original inputs (as opposed to vector addition, which produces an output that is highly similar to the inputs). Denoting this operation as $\circledast$, and introducing new vectors for the *roles* that terms in the sentence take on, we can represent the sentence $\mathbf{S}$ as

$$\mathbf{S} = \texttt{agent}\circledast\texttt{cats} + \texttt{verb}\circledast\texttt{chase} + \texttt{object}\circledast\texttt{mice}.$$

Importantly, we also need to reverse (i.e., decompress) these operations. Given a sentence, we need to be able to identify what the verb is, for example. For this, we need an inverse operation such that

$$\mathbf{S}\circledast\texttt{verb}' \approx \texttt{chase}$$

where $\texttt{verb}'$ is an inverse of $\texttt{verb}$ (i.e., something that, when bound with $\mathbf{S}$, will produce $\texttt{chase}$).

There are a number of different vector operations that can fulfill the role of the binding and inverse operators, and this family of approaches is known as vector symbolic architectures (VSAs) [25]. One that is natural to implement in neurons via the NEF is circular convolution, which was originally explored by Plate in his holographic reduced representations [46]. To efficiently implement this operation in neurons, we note that: 1) circular convolution is elementwise multiplication in the Fourier transform space; and 2) the Fourier transform of a vector is a linear operation (multiplication by $\mathbf{F}$, the discrete Fourier transform matrix). Thus, the binding of any two vectors $\mathbf{A}$ and $\mathbf{B}$ can be computed by

$$\mathbf{C} = \mathbf{A}\circledast\mathbf{B} = \mathbf{F}^{-1}(\mathbf{FA} \odot \mathbf{FB})$$

where $\odot$ is used to indicate elementwise multiplication of the two vectors (i.e., $\mathbf{x} \odot \mathbf{y} = (x_1y_1, \ldots, x_ny_n)$). Given the NEF, this is easily computed using a standard, two-layer feedforward network (see Fig. 5), with the imaginary components of the $\mathbf{F}$ matrix treated just as separate elements in the vector.

To unbind vectors (i.e., to extract information out of a sentence), we follow Plate [46] in noting that circular convolution has an approximate inverse: circular correlation. Furthermore, circular correlation is the same as circular convolution, except that the second vector has its elements permuted. In other words, $\mathbf{A} \approx (\mathbf{A}\circledast\mathbf{B})\circledast\mathbf{B}'$ where $\mathbf{B}'_i = \mathbf{B}_{(N-i)\,\mathrm{mod}\,N}$ and i indexes the N elements of $\mathbf{B}$. Since the permutation is a linear operation (denoted here

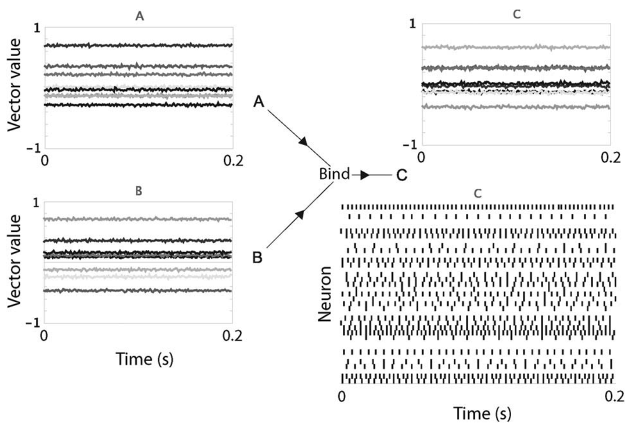

Fig. 5. ***Network computing the binding operation (circular convolution) on two 8-D vectors. There are four groups of neurons: A, B, Bind, and C. Groups A, B, and C have 150 neurons and group Bind has 760 neurons. The decoded vectors from A, B, and C are shown, and can be seen to be roughly constant. The spiking activity over 200 ms of 38 randomly chosen neurons in the output population is shown. The Bind neurons encode*** FA ***and*** FB***, where*** F ***is the discrete Fourier transform matrix. The connections from Bind to C compute*** $\mathrm{F}^{-1}(\mathrm{FA} \odot \mathrm{FB})$***, which is the circular convolution of*** A ***and*** B ***(from [16] with permission).***

as $\mathbf{S}$), a very similar network to that shown in Fig. 5 can be used to compute the following:

$$\mathbf{A} \approx \mathbf{C} \circledast \mathbf{B}' = \mathbf{F}^{-1}(\mathbf{FC} \odot \mathbf{FSB}).$$

This method for representing structured information allows us to perform symbol-like manipulations: building larger structures out of basic vectors and extracting (approximations of) those vectors. However, the fact that these representations are vectors rather than symbols adds new functionality beyond that of standard symbol systems. In particular, we have shown that these representations can be used to perform pattern matching *across* structured information. For example, given the sequence "3," "33," "333," a person can quickly conclude that the next item is "3333." This type of pattern completion has proven difficult to do computationally without simply giving a computer a set of predetermined patterns to look for. However, semantic pointers allow the system to induce the pattern. To solve this problem with semantic pointers, each item is converted into a structured representation as above (so "3" becomes $\mathtt{three} \circledast \mathtt{item1}$ and "33" becomes $\mathtt{three} \circledast \mathtt{item1} + \mathtt{three} \circledast \mathtt{item2}$ and so on), then the transformation between each vector pair is estimated, and finally the estimates are averaged together. The averaging gives an estimate of a generic "next" transformation that can be applied to the last item to give the next predicted item in the sequence. For example, if the three items in the pattern are $\mathbf{P}_1$, $\mathbf{P}_2$, and $\mathbf{P}_3$, the following is computed:

$$\begin{aligned}
\mathbf{T}_1 &= \mathbf{P}_2 \circledast \mathbf{P}'_1 \\
\mathbf{T}_2 &= \mathbf{P}_3 \circledast \mathbf{P}'_2 \\
\mathbf{T} &= \frac{1}{2}(\mathbf{T}_1 + \mathbf{T}_2) \\
\mathbf{P}_4 &= \mathbf{P}_3 \circledast \mathbf{T}.
\end{aligned}$$

We have shown that this basic method for producing $\mathbf{P}_4$, a prediction of the next item in the list, can be used to account for human performance on Raven's Progressive Matrices [48], [50], the leading measure of general intelligence. In the case of the sequence "3," "33," "333," the result is approximately $\mathtt{three} \circledast \mathtt{item1} + \mathtt{three} \circledast \mathtt{item2} + \mathtt{three} \circledast \mathtt{item3} + \mathtt{three} \circledast \mathtt{item4}$, or "3333." The necessary steps are natural to implement in neurons using the NEF, and the result is not only the first neural explanation of this cognitive behavior, but also the first explanation in any form which does not "build in" a large set of different pattern types. The key advantage here is that semantic pointers can not only contain structured information, but can also be manipulated using vector operations (e.g., averaging), to provide statistically and syntactically meaningful results.

B. Cognitive Control: Action Selection and Execution

The above approach to structured representation can be used for many purposes. For example, to remember the list "seven, six, four," we can represent the vector $\mathbf{M} = \mathtt{seven} \circledast \mathtt{item1} + \mathtt{six} \circledast \mathtt{item2} + \mathtt{four} \circledast \mathtt{item3}$. To actually store such a representation in a "working memory," we can build a network whose dynamics are essentially $dx/dt = \mathbf{u}$ (i.e., an integrator). This will hold its value over time given no input $(\mathbf{u} = 0)$. However, in order to make use of such a component within a larger cognitive system, we need a method for controlling the inputs and outputs to this component. However, we cannot do this by having connections between components appear and disappear at different times during the simulation: real biological synaptic connections do not change so quickly. Rather, we need a biologically plausible method for selectively routing the output of one component to the input of another component, depending on task demands. We call these routings "actions," but note that they can be both physical actions and cognitive actions (moving vectors from one area of the brain to another).

This control problem can be broken down into two parts: action selection (determining which routing is appropriate right now) and action execution (implementing the routing). For action selection, we take the standard approach of computing the "utility" of each action (how good it would be to perform this action in the current context) and then choosing the action with the highest utility. If different brain areas contain the current cognitive states $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$, then we can use the NEF to compute the utility of an action i of the form $U_i = f(\mathbf{x}) + g(\mathbf{y}) + h(\mathbf{z})$. This can be done using (6) and connection weights coming from the neural populations encoding $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$. Note that if we want the more general $U_i = f(\mathbf{x}, \mathbf{y}, \mathbf{z})$, we can achieve this by forming a new neural group representing $\mathbf{q} = \langle \mathbf{x}, \mathbf{y}, \mathbf{z} \rangle$ and making connections from the original areas to the new area. However, for the models discussed here, we find that most of the time the functions needed to compute utility are extremely simple linear function such as $U_i = \mathbf{L} \cdot \mathbf{x}$, where $\mathbf{L}$ is a vector and $\mathbf{x}$ is the state stored in a particular brain area.

Determining which utility value is the largest is, however, more complicated. The most obvious approach is to take the vector of all utilities for each action $\mathbf{U}$ and compute the function $\max(\mathbf{U})$. Unfortunately, this function turns out to be difficult for neurons to approximate, requiring more neurons than exist in the human brain for a vector of only 100 actions. Another approach is to build a winner-take-all system by implementing the dynamical system $\mathrm{d}\mathrm{U}_i/\mathrm{d}t = \mathbf{U}_i - \sum_{i \neq j} \mathbf{U}_j$. This approach, however, requires a fair bit of time, as the system must wait for the value $\mathbf{U}$ to settle to a stable equilibrium.

For our models, we have chosen an alternate approach based on the basal ganglia, a highly interconnected cluster of brain areas found underneath the neocortex and near the thalamus. This brain area has been consistently implicated in the ability to choose between alternative courses of action. Damage to the basal ganglia occurs in several diseases of motor control, including Parkinson's and Huntington's diseases, and results in significant cognitive defects [22]. Neuroscientists [52] and cognitive scientists [1] consider the basal ganglia as being responsible for action selection in both motor and cognitive domains [37], [38].

The anatomical and physiological structure of the basal ganglia suggests that it computes a particular dynamical system which effectively approximates the maximum function [28]. The input is exactly the vector $\mathbf{U}$ described above, the list of utilities for each action. The output is a vector of the same length, but which is zero for the largest element in $\mathbf{U}$ and positive for the other elements. By implementing this dynamical system using the NEF, we achieve a biologically realistic spiking implementation of action selection that gives a close approximation to a maximum operation with a reasonable number of neurons and which responds quickly to changing inputs. For further details, see [59].

To execute the actions chosen by this mechanism, we turn to the well-known cortex/basal ganglia/thalamus loop through the brain (see Fig. 6). Roughly speaking, the SPA assumes that cortex provides, stores, and manipulates representations, the basal ganglia determine which action

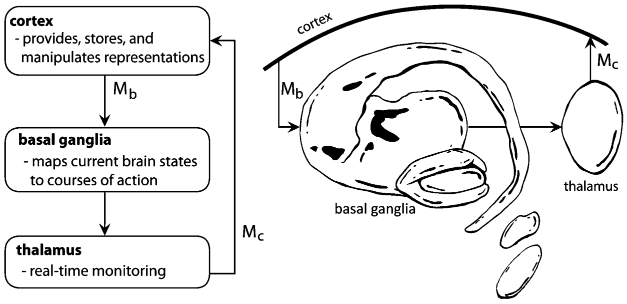
cortex
- provides, stores, and manipulates representations
M_b
basal ganglia
- maps current brain states to courses of action
thalamus
- real-time monitoring
M_c
cortex
M_b
M_c
thalamus
basal ganglia

Fig. 6. ***Proposed mapping between the SPA action selection system and the neuroanatomy of the cortex-basal ganglia/thalamus loop. Arrows indicate connections between the three areas. At a functional level, brain states from the cortex are mapped through the M_b matrix to the basal ganglia. Each row in such a matrix specifies a known context for which the basal ganglia will choose an appropriate action. The product of the current cortical state and M_b provides a measure of how similar the current state is to each of the known contexts. The output of the basal ganglia disinhibits the appropriate areas of thalamus. Thalamus, in turn, is mapped through the matrix M_c back to the cortex. Each column of this matrix specifies an appropriate cortical state that is the consequence of the selected action. The relevant anatomical structures are pictured on the right (from [16] with permission).***

to take based on those representations, and the thalamus implements the desired routing action between areas of the cortex. To perform this routing, we need to organize neurons such that there is a connection from one cortical area to another such that this connection will compute $\mathbf{y} = \mathbf{x}$ if the action is selected, and $\mathbf{y} = \mathbf{0}$ if it is not selected. This can be done in two ways. First, we could use the NEF to approximate that nonlinear function. However, for this particular special case there is a simpler approach. We start by forming an intermediate group of neurons $\mathbf{z}$ and forming connections from $\mathbf{x}$ and to $\mathbf{y}$ such that $\mathbf{y} = \mathbf{z} = \mathbf{x}$. As it stands, this will always route information from $\mathbf{x}$ to $\mathbf{y}$. However, an interesting feature of the basal ganglia (and of our model of the basal ganglia) is that the output neurons are active for all actions *except* for the one that has been selected. This means we can simply connect the output neurons for this action from the basal ganglia to the neural population $\mathbf{z}$ with strong negative weights (i.e., without using the NEF). This means that whenever the action is *not* selected, the neurons in $\mathbf{z}$ will stop firing, resulting in a value of 0 being passed to $\mathbf{y}$. These $\mathbf{z}$ neurons form our model of the thalamus.

Using this architecture, we can implement a controlled cognitive system. To understand how this structure operates, we can begin by thinking of it as a neural implementation of a classical *production system*, a standard approach for explaining human cognition (although the SPA is more computationally powerful [16, ch. 7]). A production system consists of a large set of IF–THEN rules. At any moment in time, only one of those rules can be active (determined by which rule's IF condition best matches the current situation). When a rule "fires," the THEN part of that rule is taken to indicate the cognitive action that should be taken (e.g., moving information from the visual system to working memory, modifying certain information, moving information from working memory to the speech areas, etc.). Psychologists have consistently inferred from behavior that the human brain requires approximately 50 ms to select and execute such a cognitive action [2]. The timing within our simulations is similar [59], but in that case, the timing is due to the neurotransmitter time constants measured from the relevant physiological characteristics of neurons, providing a more biophysical explanation of timing phenomena.

In building this neural architecture, we generalize the IF portion of a rule to be a utility calculation, and the THEN portion of the rule to be a routing control signal. As an example, consider the following rule:

$$U_i = \texttt{visual} \cdot (\mathbf{A} + \mathbf{B} + \mathbf{C} + \mathbf{D} + \cdots)$$
$$R_i : \texttt{visual} \rightarrow \texttt{memory}$$

where U_i is the utility of the ith rule and R_i is the routing that should occur when this rule's utility is higher than all the other utility values. The utility calculation for this rule consists of taking the dot product between whatever is currently represented in the visual system, and the sum of already known representations of the letters A–Z. This means the utility will be high if there is a letter in the visual system, and low otherwise. If utility is high, this rule is likely to be selected and then the resulting routing would be affected: i.e., the visual item would be put into working memory. In short, this particular rule says IF the visual system currently contains a letter (i.e., any of the semantic pointers for A, B, C, D, and so on), THEN remember that letter. Note that even if the vector stored in the visual field is not exactly one of these letters, this rule will still have a high utility and will be chosen by the basal ganglia (assuming no other rule has a higher utility). To implement this rule, we form connections between the `visual` neural population and the neurons in the basal ganglia representing the input utility of the ith rule. These connections are set using the NEF to compute the linear function $U_i = \texttt{visual} \cdot (\mathbf{A} + \mathbf{B} + \mathbf{C} + \mathbf{D} + \cdots)$. To implement the effect of the rule, we add neurons to the thalamus that simply take in the value from the `visual` neurons and output that same value to the `memory` neurons (computing the identity function each time). A purely inhibitory connection is then made from the ith output of the basal ganglia to these new neurons in the thalamus. Thus, when this action is *not* chosen (i.e., when it is not the action with the highest utility), the output from the basal ganglia will inhibit these neurons, stopping the information from passing through. When the action is chosen, this inhibition is released, allowing the effect of the rule to occur.

Note that despite our characterization of the functioning of the system as implementing a rule, it is doing something much more computationally powerful. The particular cases mentioned here only make use of linear functions to compute U_i, but the NEF allows for nonlinear functions as well. Furthermore, the action selection system gives a more flexible comparison system than the strict if-and-only-if approach seen in standard production systems. Indeed, this same system can be used for Bayesian expectation maximization (see [16, ch. 7] for details). As a result, the "rules" of the SPA are significantly more computationally powerful than the logical structures found in standard production systems.

Returning to the example at hand, we can add other rules; for instance, rules to traverse the alphabet. One way to implement traversing the alphabet is to add 25 rules of the same form as the following:

$$U_i = \texttt{memory} \cdot \mathbf{G}$$
$$R_i : \mathbf{H} \rightarrow \texttt{memory}.$$

This rule says IF the item in memory is G, THEN replace it with H. In essence, a set of rules like this determines a set of state transitions that can either terminate or be cyclic. The result of simulating this particular rule set can be seen in Fig. 7.

To increase the flexibility of the system, it is possible to introduce routing actions that manipulate vectors as they are passed from one component to another. This is vital for performing transformations of structured representations. For example, simple question answering tasks that query sentence-like representations can be implemented in this manner. Consider the sentence

$$\mathbf{S} = \texttt{statement} + \texttt{blue} \circledast \texttt{circle} + \texttt{red} \circledast \texttt{square}$$

which indicates that there is a blue circle and a red square presented to the system. We would like to be able to ask an arbitrary question of this kind of sentence, perhaps "What is red?" This question can be formulated as

$$\texttt{question} + \texttt{red}.$$

We expect the system to respond with the vector for `square` when provided with this question. In this simple example, we do not consider the problem of sequentially presenting each word and constructing the relevant representation. For a description of the rules and

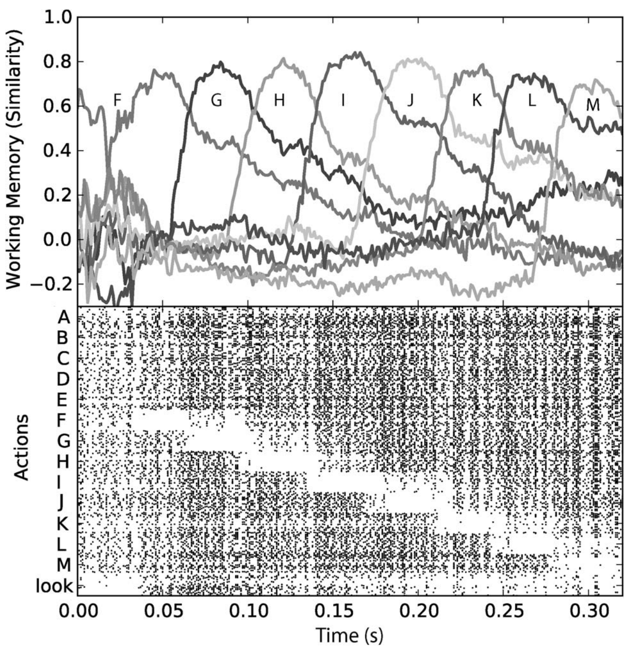

Fig. 7. ***Routing information to give a sequence of actions. The decoded content of working memory is shown on top. This is the similarity (dot product) of the decoded vector with the (randomly chosen) ideal vectors for the different letters of the alphabet [see (2)]. The lower half of the graph shows spiking output from the basal ganglia indicating the action to perform. The output from the basal ganglia is inhibitory, so the chosen action is the one for which the neurons do not fire. The "look" action takes information from visual cortex and routes it to working memory (in this case* F*; from [16] with permission).***

processing units needed to perform that additional function, see [9] and [61].

As a result, this simple case can be implemented with two basal ganglia rules. The first handles remembering the initial statement about the environment

$$U_i = \texttt{visual} \cdot \texttt{statement}$$
$$R_i : \texttt{visual} \rightarrow \texttt{memory}.$$

This rule will move whatever is in the visual system to working memory. The second handles the question

$$U_i = \texttt{visual} \cdot \texttt{question}$$
$$R_i : \texttt{memory} \circledast \texttt{visual}' \rightarrow \texttt{output}$$

which indicates that the question should be used to decode the memory to produce an answer

These two rules will handle any question answering task of this form. If we provide the network with the input described above, the first rule will cause **S** to be stored in memory. When the question appears, the second rule will cause that value, bound with the inverse of the question, to be output. This gives the following result, as depicted in Fig. 8:

$$\begin{aligned}
&(\texttt{statement} + \texttt{blue} \circledast \texttt{circle} + \texttt{red} \circledast \texttt{square}) \\
&\quad \circledast (\texttt{question} + \texttt{red})' \\
&\qquad = (\texttt{statement} + \texttt{blue} \circledast \texttt{circle} + \texttt{red} \circledast \texttt{square}) \\
&\qquad\quad \circledast (\texttt{question}' + \texttt{red}') \\
&\qquad = \texttt{red} \circledast \texttt{square} \circledast \texttt{red}' + \texttt{red} \circledast \texttt{square} \\
&\qquad\quad \circledast \texttt{question}' + \cdots \\
&\qquad \approx \texttt{square} + \texttt{red} \circledast \texttt{square} \circledast \texttt{question}' + \cdots
\end{aligned}$$

Since the binding operation produces vectors that are highly dissimilar to the inputs, the extra terms will all be dissimilar to `square`, and essentially act as a noise term. Therefore, the resulting output vector will be similar to `square`, in the sense that it will have a larger dot product with `square` than with any other item in the vocabulary. We have shown that this representation is sufficient to accurately decode up to eight items from a standard adult vocabulary size of 60 000–100 000 items [64]. To achieve this, we need approximately 500-dimensional vectors. Interestingly, we find that the neural models generated by the NEF necessary to implement circular convolution on 500-dimensional vectors are consistent with the anatomical connectivity found in human cortex [16].

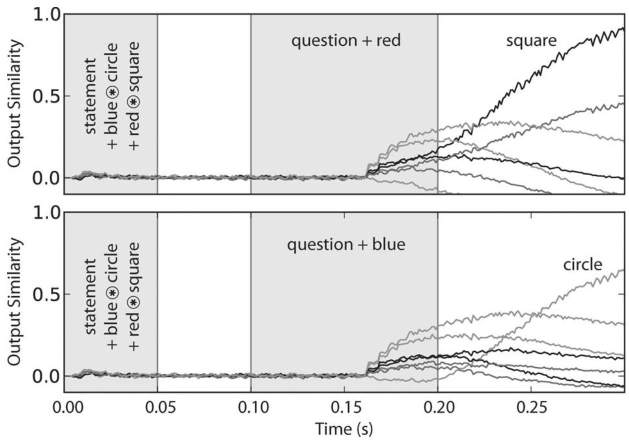

Fig. 8. ***Question answering with neurons. The indicated visual input is provided to the network during the periods shaded in gray. For these two examples, the same statement is provided to the network, but then a different question is given. The decoded vector from the output population is shown by giving its similarity to the ideal vectors for the possible answers. For the first question, the correct answer is*** `square` ***and for the second question the correct answer is*** `circle` ***(from [60] with permission).***

Finally, we note that the 55 million neurons [3] found in the input nucleus of the basal ganglia (i.e., striatum) give an upper bound of one to two million rules for human cognition. This is because those neurons need to accurately represent $\mathbf{U}$ and we generally find that 30–50 neurons per $\mathbf{U}_i$ value is sufficient accuracy for the models we have built. Determining what these rules may be is a significant, outstanding research question, but we believe that the action selection system described here will continue to function as expected at that scale.

C. Learning

The NEF allows very large-scale neural models to be built via an analytic closed-form optimization, rather than through online or batch learning, as is standard for neural network research. However, complex cognitive systems exhibit adaptive behavior based on experience. To account for adaptation, we can use the NEF to generate the initial connection weights of a model, and then introduce a learning rule on *some* of those connections. Importantly, the same rule is not applied everywhere throughout the model; as in the real brain, some connections are more malleable than others.

We have developed a spike-timing-dependent plasticity (STDP) learning rule that is a more realistic version of the standard Hebbian "neurons that fire together, wire together" principle [32]. It is a local rule, meaning that all the information needed to implement it is available locally at a particular synapse, and it matches observed adaptive timing effects in individual neurons [6]. This learning rule combines the well-known homeostatic Bienenstock–Cooper–Munro (BCM) rule [7] with an error-driven spiking rule [42], making it a homeostatic, prescribed error sensitivity (hPES) rule.

The hPES rule is as follows:

$$\Delta\omega_{ij} = \kappa\alpha_j a_i \left(S\mathbf{e}_j \cdot \mathbf{E} + (1-S)a_j - \theta\right)$$

where $0 \le S \le 1$ is the relative weight of the supervised and unsupervised terms, $\Delta\omega_{ij}$ is the change in synaptic weight, κ is a learning rate, α_j is the gain on the postsynaptic neuron, a_i is the postsynaptic filtered spiking activity from the presynaptic neuron, $\mathbf{e}_j$ is the encoding vector of the postsynaptic neuron, $\mathbf{E}$ is an error signal (e.g., via a modulatory or nonmodulatory neural signal), a_j is the filtered postsynaptic neural activity, and θ is a modification threshold that imposes homeostasis on the postsynaptic neuron.

The hPES rule simultaneously addresses limitations of both standard Hebbian learning rules and STDP. In particular, unlike most standard rules, hPES is able to account for precise spike time data, and unlike most STDP (and standard Hebbian) rules, it is able to relate synaptic plasticity directly to the vector space represented by an ensemble of neurons. That is, the rule can tune connection weights that compute nonlinear functions of the high-dimensional vector being represented by the spiking patterns in the neurons. Unlike past proposals, hPES is able to do this because it relies on an understanding of the NEF decomposition of connection weights. This decomposition makes it evident how to take advantage of high-dimensional error signals, which past rules have either been unable to incorporate or attempted to sidestep [27]. Thus, hPES can be usefully employed in a biologically plausible network that can be characterized as representing and transforming a vector in spiking neurons; i.e., precisely the kind of networks employed in the SPA.

We have shown that the hPES rule can learn the connections needed to perform a wide variety of functions [6], [42]. That is, instead of using the NEF equations to solve for connection weights, it would also be possible to learn those connections using our hPES rule. For example, Fig. 9 shows this rule learning a 2-D identity function. This optimization is much slower than simply using the NEF to solve for the resulting connection weights, but it demonstrates the rule's ability to learn the same functions online. Importantly, learning a function in this manner requires an error signal. The system will learn whatever function is consistent with that error signal.

In the real brain, error signals have many forms. Perhaps most famously modulatory learning has been associated with the neurotransmitter dopamine in both the cortex and basal ganglia. The dopamine signal is often considered a reinforcement or reward signal, indicating when actions are rewarded (or not). Dopamine acts to help modify the connections from the cortex to basal ganglia, and is thought to play a central role adjusting actions to deal with a changing environment [34], [43].

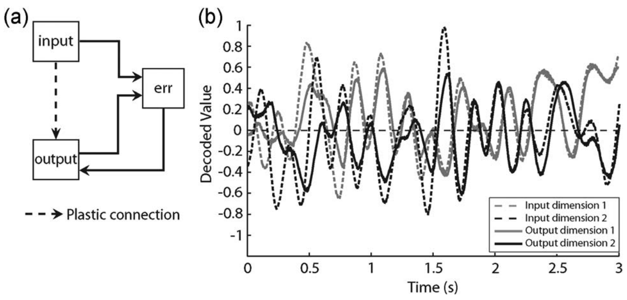

Fig. 9. *Learning a function. (a) Rather than solving for the ideal connection weights to compute the 2-D identity function, we start with random connections on the plastic connection and provide an error signal from a separate neural population. The hPES rule adjusts the connections between the input and output. A network learning a 2-D communication channel. (b) The network learns to approximate the desired function. Over time the decoded output (solid lines) converges to the desired output (dotted lines). The error signal is computed by the neural population "err" as the difference between the two. The same network structure can be used to learn most nonlinear vector functions (from [16] with permission).*

We incorporate this learning into the SPA by providing a reward signal and using it to adjust the utility of the rules in the basal ganglia. This gives the system the capability of reinforcement learning (RL) [66]. We have included basic RL capabilities in Spaun (see Section IV), and in more recent work described a model that is able to perform RL in semi-Markov decision process environments (i.e., environments where there are arbitrary delays between action and reward), which is significantly more challenging problem than that addressed by Spaun [49].

D. The Overall Architecture

Sections III-A–III-C detail how to make models of particular neural areas that perform particular cognitive functions such as memory (integration) or the binding operation (circular convolution). It also describes how these components may incorporate learning, and how the flow of information between the components can be controlled. The result is a general cognitive architecture where every component can be implemented in neurons.

When we design such systems, we have found that the models of particular neural areas (or modules) are all of a particular form, shown in Fig. 10. This core structure can be used for vision, audition, motor control, working memory, pattern completion, and so on. The details of exactly what function is being computed, of course, vary by brain area.

Fig. 11 combines these basic components together with a control system. The separate components can be thought of as physically distinct areas of the brain, each of which can perform one type of operation on its inputs. The control system is responsible for ensuring that the right information is routed to the right component at the right time.

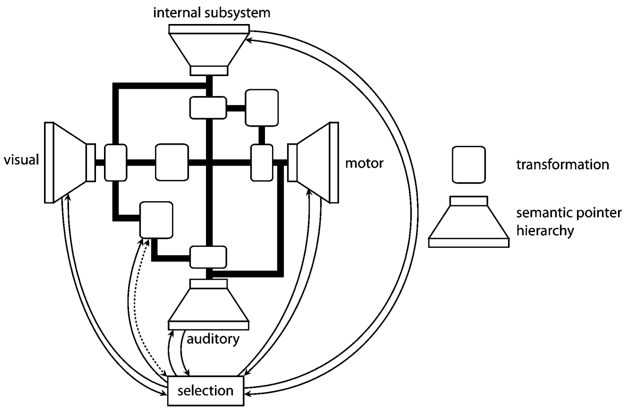

Fig. 11. ***General schema for SPA models. A full SPA model consists of multiple components as in Fig. 10. Dark black lines are projections carrying semantic pointer representations between parts of the system, while thinner lines indicate control and error signals. (Reproduced with permission from [16].)***

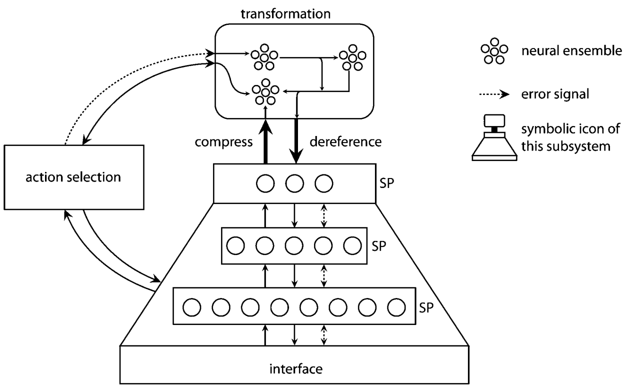

Fig. 10. ***Generic component within the SPA. Inputs are generally presented at the bottom interface, and the module preforms some function(s) on the input to create a compressed representation. The action selection system (described below) can take these results, transform them, and route them to other components. Many components can also be run in reverse, computing an (approximate) inverse operation, which decompresses or dereferences the pointer. This provides for hierarchical, structured representation and cognitive processing. (Reproduced with permission from [16].)***

In particular, the "semantic pointers" used by the SPA are compressed representations, where the compression *maintains similarity information*. That is, the vector for "dogs chase cats" will be similar to the vector for "dogs chase cars" (where similarity is measured as the dot product between vectors), but will be much less similar to "cats chase dogs" and not at all similar to "cats ignore dogs." These combined vectors are of the same length as the original vectors, so there is certainly a loss of information in this compression of multiple vectors into a single vector. However, it is generally possible to recover the original high-dimensional representation (or something very similar) thanks to the inverse operation. Because of this compression/decompression relationship, similar inputs will produce similar outputs.

In short, semantic pointers are compact ways of referencing large amounts of data; consequently, they function similarly to "pointers" as understood in computer science. Typically, in computer science, a "pointer" is the address of some large amount of data stored in memory. Pointers are easy to transmit, manipulate, and store, so they can act as an efficient proxy for the data they point to. Semantic pointers provide the same kind of efficiency benefits in a neural setting. This is why we use them to transfer information between cognitive modules.

Unlike pointers in computer science, however, semantic pointers are *semantic*. That is, they are systematically related to the information that they are used to reference. This means that semantic pointers carry similarity information that is derived from their source, in contrast to an arbitrary index that does not contain semantic information, as generally used in modern computers.

Of course, we do not have to use the same compression operation everywhere, and indeed there are many we have not explored in the SPA (e.g., [56]). For vision (and other sensory modalities), we learn this compression from the structure of the sensory input. This is the same approach taken by vision researchers who use deep belief networks [33], [67]. However, for cognitive operations such as combining concepts together into structured representations, we use the circular convolution described above.

E. SPA in Nengo

All of the SPA components discussed above can be built in Nengo using the scripting system. However, to simplify the creation of these cognitive models, we have developed an additional library that assists in the creation of models based on the SPA characterization of the cortex, basal ganglia, and thalamus loop.

Individual processing modules are specified using the standard NEF libraries. To connect them, however, we use a specialized syntax for defining the rules that can be used to route information between areas. For example, the following code creates the question answering model from Fig. 8:

```
class Rules :
   def store(visual =′ STATEMENT′) :
      set(memory = visual)
   def recall(visual =′ QUESTION′) :
      set(output = memory* ~ visual)
class QuestionAnsweringModel(spa.SPA) :
   dimensions = 512
   visual = spa.Buffer(feedback = 0)
   output = spa.Buffer(feedback = 0)
   memory = spa.Buffer(feedback = 1
   bg = spa.BasalGanglia(Rules)
   thalamus = spa.Thalamus(bg)
net = nef.Network(′Question Answering′)
model = QuestionAnsweringModel(net)
```

In this model, the initial rules specify the two routing rules discussed above. The second part of the code describes the cortical structure of the system. Here, there are three "buffers," which are neural ensembles that can store a vector representation. In this case, the number of dimensions for that vector is set to 512. The feedback argument specifies whether these neural ensemble use the dynamics principle to store their own state. When feedback is set to 0, there is no recurrence. As noted in Section II-C, this means that if input is given to the visual system, it will decay very quickly, due to the short time constant of the filter $h(t) = e^{-t/\tau}$ introduced by the postsynaptic current. However, with feedback equal to 1, the dynamics principle is used to construct an integrator $\mathrm{d}x/\mathrm{d}t = \mathbf{u}$. This, ideally, would result in a system with an effectively infinite time constant. However, the neural approximation of the integrator will never be perfect (with a finite number of neurons). This means that the stored $\mathbf{x}$ value will slowly drift, resulting in information decay over time, as expected.

IV. SEMANTIC POINTER ARCHITECTURE UNIFIED NETWORK

Sections II and III described our general approach to building large-scale models using the SPA. In sum, our approach is to: 1) identify particular computational functions that need to be computed by a cognitive system; 2) build neural components that implement those functions; and 3) determine how to route information between the components using the basal ganglia to provide integrated function. This approach has recently allowed us to build what is currently the first brain model that is capable of performing cognitive tasks. The resulting 2.5 million neuron model is called the semantic pointer architecture unified network (Spaun) [19]. Spaun is able to perceive visual input through a 28 × 28 pixel retina, remember that information, act on it as appropriate, and generate motor output that moves a physically modeled arm to write numbers [see Fig. 13(a)].

Many of the elements of Spaun have been described in earlier work, including components for vision [67], recognition [64], serial working memory [8], pattern matching [48], reward processing [4], and motor control [13]. Together, these and other components allow Spaun to cover 20 anatomical brain areas, while being consistent in terms of the large-scale connectivity between and within these regions and the neurotransmitter time constants observed within these regions. To put this in perspective, the human brain has approximately 1000 areas [30] (and 80 billion neurons), so much work remains to be done.

Nevertheless, one key feature of Spaun is that it can use its neural components flexibly, thanks to the control structure provided by the basal ganglia. As a result, Spaun can perform eight different tasks, while the model itself remains unchanged. While each task uses many of the components of Spaun, they do so in a variety of ways. For example, in the list memory task, Spaun is shown a list and then repeats it back, while in the question answering task, it is first shown a list, and then it waits for a question about the list. Importantly, we do not make any changes to Spaun to allow it to perform different tasks. Rather, we *tell* Spaun to change tasks. Since the only input to Spaun is its 28 × 28 visual field, we do this by presenting the letter "A" followed by a number that indicates the task to perform.

Spaun responds appropriately to this perceptual input given these two control rules

$$\begin{aligned}
&U_1 = \texttt{visual} \cdot \mathbf{A} \\
&R_1 : \texttt{NONE} \rightarrow \texttt{task} \\
&U_2 = (\texttt{visual} \cdot (\texttt{ZERO} + \texttt{ONE} + \texttt{TWO} + \cdots) \\
&\quad + \texttt{task} \cdot \texttt{NONE})/2 \\
&R_2 : \texttt{visual} \rightarrow \texttt{task}
\end{aligned}$$

The first rule clears the part of memory that keeps track of whatever task Spaun is currently doing when Spaun sees the letter A (by setting it to the random vector NONE). The second rule will only have a high utility if the task is NONE and a number has been presented. When this occurs, that number is sent to the task memory. To implement the various tasks, more rules are added, each of which has, as part of their utility calculation, a dependency on the task memory. This makes the rules task specific, but the actual neural components that the rules make use of are general across tasks. The eight tasks Spaun can perform (digit recognition, digit style copying, list memory, question answering, addition by counting, reinforcement learning, pattern completion, and the Raven progressive matrix intelligence test) are all implemented with only 19 basal ganglia rules. More information on these tasks can be found in [19] and Spaun's performance can be viewed in online videos (http://nengo.ca/build-a-brain/spaunvideos). The overall functional and anatomical architecture of Spaun is shown in Fig. 12.

As an example of Spaun performing a task, Fig. 13 shows the list memory task. The input starts with an "A" followed by a "3," telling Spaun which task to perform. The next input is a series of numbers that it should attempt to remember. When it sees a question mark, it responds by writing out the remembered list. Fig. 13(c) shows spiking activity of several parts of the model. Of particular note is dorso–lateral prefrontal cortex (DLPFC), where Spaun is storing the list. The list is stored as a vector, using semantic pointers as described in Section III-A. The DLPFC graph shows the similarity (as measured by the dot product) between the vector stored there (decoded from the spikes using the NEF) and the ideal vector for numbers in different positions. This activity decays over time and becomes less accurate the more items there are in the list. In this particular case, this inaccuracy is enough to cause Spaun to make a mistake: it forgets the fourth item in the list (the eight). Indeed, the model follows a similar forgetting curve to that seen in people [8].

In fact, Spaun can be used to match a wide variety of data across many scales. Its components have been shown to reproduce spike patterns in the basal ganglia during an RL task [4], single neuron tuning curves found in primary visual cortex [19], population spectrogram shifts during a working memory task [19], recognition accuracy on naturalistic stimuli (i.e., handwritten digits) [19], and reaction times during a counting task [19], and similar results apply to the complete model as well [16]. In short, because the model implements cognitive behaviors using spiking neurons that are anatomically and physiologically matched to their biological counterparts, its performance can be constrained by data from single cell physiology through to behavior. By making successful comparisons to diverse constraints, we begin to build the case that Spaun is capturing some important aspects of the biological mechanisms at work in real brains.

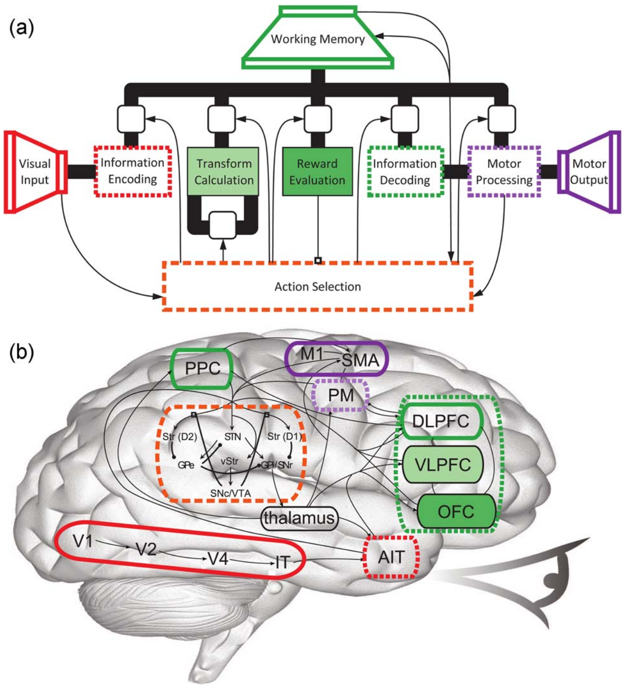

Fig. 12. ***Functional and anatomical architecture of Spaun. (a) A breakdown of the separate logical components within Spaun and their connections. The working memory, visual input, and motor output components are typical hierarchies as per Fig. 10, while the other components compute functions of these representations. From left to right, the components: 1) map visual inputs to conceptual representations; 2) induce relationships between representations; 3) associate input with reward; 4) map conceptual representations to motor actions; and 5) map motor actions to specific patterns of movement. (b) The corresponding neuroanatomical architecture, with matching colors and line styles indicating corresponding functional components. Abbreviations: V1/V2/V4 (primary/secondary/extrastriate visual cortex), AIT/IT (anterior/inferotemporal cortex), DLPFC/VLPFC/OFC (dorso–lateral/ventro–lateral/orbito–frontal cortex), PPC (posterior parietal cortex), M1 (primary motor cortex), SMA (supplementary motor area), PM (premotor cortex), v/Str (ventral/striatum), STN (subthalamic nucleus), GPe/i (globus pallidus externus/internus), SNc/r (substantia nigra pars compacta/reticulata), VTA (ventral tegmental area). (Reproduced with permission from [18].)***

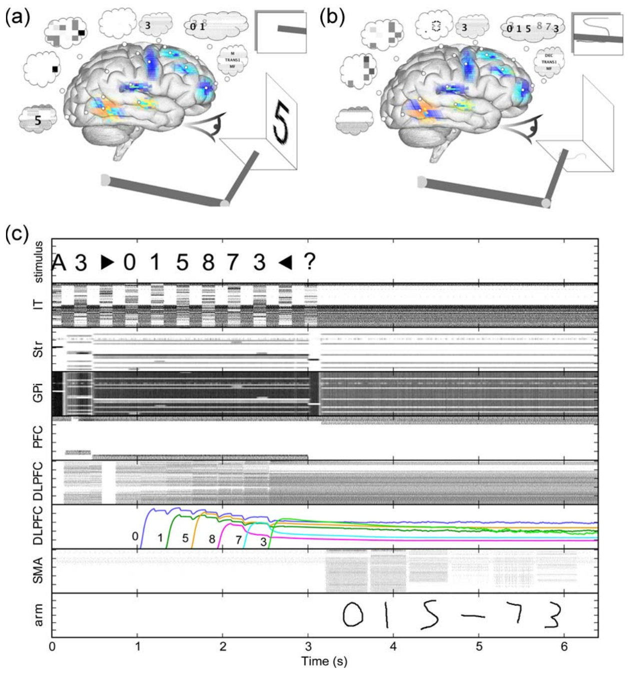

Fig. 13. *Spaun performing a serial recall task. Spaun must remember and then write out the sequence "0, 1, 5, 8, 7, 3." It has never seen this particular sequence before. (a) A snapshot of Spaun while it is seeing the third digit in the list. (b) A snapshot of Spaun while it is writing its result. The internal processing of the model is shown in thought bubbles with decoded values overlaid on the spiking activity. Colors on the brain image show overall firing rates in different regions. (c) The neural spiking activity of various components of the model (see Fig. 12 for details). In this particular trial, Spaun makes a mistake, forgetting the digit in the middle of the list (just as human subjects often do). The spiking activity in DLPFC is decoded to reveal the cause of this mistake (the vector stored in memory has become too dissimilar to the ideal representation of an 8). (Reproduced with permission from [16].)*

V. DISCUSSION

While we believe that models like Spaun are moving us toward a better understanding of brain function, there remain many challenges ahead. It is important to keep in mind that Spaun has 40 000 times fewer neurons than the human brain. Consequently, it is still not clear how well the methods of the SPA will scale, despite encouraging initial results. Similarly, Spaun includes several simplifying assumptions regarding the number and kind of neurotransmitters, and physiological properties of individual neurons. Again, past work using the same methods has incorporated a wider variety of such properties than are found in Spaun, but it remains to be seen how additional biological detail will affect Spaun's functioning.

In addition to these limitations, it is clear that there are many kinds of brain function not well reflected in Spaun. For instance, the ability of the model to lay down new long-term memories is minimal (this only occurs in one of the eight tasks). Similarly, there is little environmental interaction of the model: its single eye remains fixed, and it does not see its own output. Much work remains to be done to determine what additional functions are necessary to allow such a model to be embedded in a dynamic, open-ended environment. Relatedly, learning new cognitive behaviors in such an environment is a well-known challenge with few general, effective solutions. It has not yet been shown how these kinds of models can be used to effectively tackle such challenges.

From a computational perspective, simulating large-scale neural models on conventional computational hardware is difficult. For Spaun, it took approximately 2.5 h of simulation time to generate 1 s of behavior on a high-end workstation. While we believe that this simulation can be made much more efficient (and in the latest version of our software we have made significant improvements [5]), it is clear that alternate computing approaches would be advantageous.

A wide variety of these brain-inspired computing devices exist, all based around the idea of having a large number of simple neuron-like components whose spiking activity is based on the sum of their inputs. We have used the NEF as a general method for programming such neuromorphic computers. Examples of using the NEF in this way can be found on efficient digital architectures employing thousands or millions of ARM cores [24], analog architectures that directly incorporate forms of learning [12], and hybrid architectures [10]. There are many benefits to this new computing paradigm, including orders of magnitude better power efficiency per computation, robustness to noise and variability, and massive parallelism [31].

Because the NEF was developed to address systems with these same properties (but in a biological setting), it has proven an effective means of programming such hardware, by indicating what the connection weights should be to achieve different computational results. For example, the NEF has been used to control a robot that can learn by treating training examples as the function to be approximated, to do operational space control on a 3-joint arm [44], and to implement a model of the rat hippocampus' path integration ability on a mobile robot [23]. In all of these examples, the algorithms being implemented are well suited to approximation using the NEF, and so are much more efficient when implemented on neuromorphic hardware than on traditional computing devices.

While current hardware implementations remain small scale compared to models like Spaun, we expect that in the near future, there will be a significant increase in the size of neuromorphic platforms available. Indeed, we expect that a full hardware implementation of Spaun will be completed within the next two years. This codevelopment of algorithms, programming techniques, and infrastructure in the neuromorphic space provides fertile ground for designing and testing brain-like models. We believe that such models will allow us both to better understand biological brain function, and to develop a new class of solution to challenging information processing problems.

VI. CONCLUSION

The theoretical methods and software suite described in this paper form a comprehensive tool chain for connecting high-level behavior to low-level neural processes. The NEF compiles algorithms expressed in terms of vectors and functions on those vectors (or their temporal derivatives) into a neural network that approximates those functions. A wide variety of single cell models can be employed, and accuracy can be improved by increasing the number of neurons used. The NEF thus provides a generic framework for implementing a very large class of functions in networks of biologically plausible spiking neurons.

The SPA is our suggestion of a means of organizing neural models that is consistent with contemporary neuroscience. While it is clearly too early to claim that this is provably how the brain works, we believe that SPA provides a testable ongoing research program, and at the very least provides what is currently the only standard for expressing and manipulating structured, symbol-like representations while being consistent with biological constraints [16], [58]. It provides a method for flexibly manipulating representations and passing them between different brain areas as appropriate for a given cognitive task. Both the NEF and the SPA can also optionally include neural learning rules, providing the system a way to adapt online from experience.

To make these theoretical ideas more practically useful, we have also briefly presented Nengo, a software tool that implements both the NEF and the SPA. This allows a user to specify the high-level function to be implemented (along with whatever neural constraints are appropriate), and Nengo will use the NEF to solve for the connection weights needed, run the resulting model, and collect the results. Nengo has also been shown to scale up well, as it was used to create Spaun, the world's first functional brain simulation, with 2.5 million neurons and over 60 billion synapses.

Combined, these approaches provide a novel method for creating large-scale neural networks that can exhibit high-level cognitive behavior. Our ongoing research involves testing psychological theories by implementing them in neurons and comparing the model performance to human (and animal) performance. Importantly, we can do this comparison based not only on the overt behavior, but also on low-level neural measurements, such as firing patterns in different brain areas. This provides strong constraints on theories of brain function. For example, we find that overall reaction time is strongly connected with the neurotransmitter time constants. Furthermore, we believe accurate models of these neural functions could lead to improved understanding of how particular neural disorders (such as Parkinson's disease or Alzheimer's disease) produce their behavioral effects. More research, however, is needed to improve the neural details of these models such that it is possible to damage them in the same way that they are damaged in those diseases.

A more surprising consequence for this research is that it provides a novel method for programming highly parallel hardware. After all, the human brain can be thought of as 100 billion interconnected processors (neurons). Each of these processors is slow, noisy, and can only compute one operation (the neural nonlinearity G), but by connecting all of them in different ways we can approximate a wide variety of functions. This is a new approach for neuromorphic engineering, and our ongoing collaborations [10], [24] are examining the possibilities for implementing complex cognitive algorithms efficiently.

Simulating the human brain is a monumental task and clearly beyond what a single research group can accomplish. This is why we are interested in helping to create general, open tools that can be applied to many different brain areas and many different kinds of cognitive tasks. We have developed tutorials and documentation to introduce the open-source Nengo software (available at http://nengo.ca). To aid instruction, we have included both a complete scripting system and an integrated drag-and-drop graphical user interface (GUI) for Nengo [65]. We hope that being able to integrate ideas from psychology, neuroscience, and artificial intelligence and construct large-scale neural models that connect sensory systems, cognitive system, and motor systems makes for an exciting new approach to brain research. We believe these sorts of models will be extremely beneficial for understanding human cognition, treating brain disorders, and developing efficient parallel computation. ■

REFERENCES

[1] J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Qin, "An integrated theory of the mind," *Psychol. Rev.*, vol. 111, no. 4, pp. 1036–1060, 2004.

[2] J. R. Anderson, *How Can the Human Mind Occur in the Physical Universe?* Oxford, U.K.: Oxford Univ. Press, 2007.

[3] H. Beckmann and M. Lauer, "The human striatum in schizophrenia. II. Increased number of striatal neurons in schizophrenics," *Psychiatry Res.*, vol. 68, no. 2–3, pp. 99–109, Mar. 1997.

[4] T. Bekolay and C. Eliasmith, "A general error-modulated STDP learning rule applied to reinforcement learning in the basal ganglia," presented at the Cogn. Syst. Neurosci. Conf. (COSYNE), Salt Lake City, UT, USA, Feb. 24–27, 2011.

[5] 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," *Front. Neuroinf.*, vol. 7, Jan. 2014, DOI: 10.3389/fninf.2013.00048.

[6] T. Bekolay, C. Kolbeck, and C. Eliasmith, "Simultaneous unsupervised and supervised learning of cognitive functions in biologically plausible spiking neural networks," in *Proc. 35th Annu. Conf. Cogn. Sci. Soc.*, 2013, pp. 169–174.

[7] E. Bienenstock, L. N. Cooper, and P. Munro, "On the development of neuron selectivity: Orientation specificity and binocular interaction in visual cortex," *J. Neurosci.*, vol. 2, pp. 32–48, 1982.

[8] X. Choo, *The ordinal serial encoding model: Serial memory in spiking neurons*, M.S. thesis, Comput. Sci. Dept., Univ. Waterloo, Waterloo, ON, Canada, 2010.

[9] X. Choo and C. Eliasmith, "General instruction following in a large-scale biologically plausible brain model," in *Proc. 35th Annu. Conf. Cogn. Sci. Soc.*, 2013, pp. 322–327.

[10] S. Choudhary, S. Sloan, S. Fok, A. Neckar, Eric, Trautmann, P. Gao, T. Stewart, C. Eliasmith, and K. Boahen, "Silicon neurons that compute," in *Proc. Int. Conf. Artif. Neural Netw.*, 2012, pp. 121–128.

[11] J. Conklin and C. Eliasmith, "An attractor network model of path integration in the rat," *J. Comput. Neurosci.*, vol. 18, pp. 183–203, 2005.

[12] F. Corradi, C. Eliasmith, and G. Indiveri, "Mapping arbitrary mathematical functions and dynamical systems to neuromorphic VLSI circuits for spike-based neural computation," presented at the Int. Symp. Circuits Syst., Melbourne, Australia, Jun. 1–5, 2014.

[13] T. Dewolf and C. Eliasmith, "The neural optimal control hierarchy for motor control," *Neural Eng.*, vol. 8, no. 6, 2011, DOI: 10.1088/1741-2560/8/6/065009.

[14] C. Eliasmith, "Neural engineering: Unraveling the complexities of neural systems," *IEEE Can. Rev.*, vol. 43, pp. 13–15, 2003.

[15] C. Eliasmith, "A unified approach to building and controlling spiking attractor networks," *Neural Comput.*, vol. 17, no. 6, pp. 1276–1314, 2005.

[16] C. Eliasmith, *How to Build a Brain: A Neural Architecture for Biological Cognition*. New York, NY, USA: Oxford Univ. Press, 2013.

[17] C. Eliasmith and C. H. Anderson, *Neural Engineering: Computation, Representation and Dynamics in Neurobiological Systems*. Cambridge, MA, USA: MIT Press, 2003.

[18] C. Eliasmith, "The complex systems approach: Rhetoric or revolution," *Top. Cogn. Sci.*, vol. 4, no. 1, pp. 72–77, Jan. 2012, discussion 94-102.

[19] 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, pp. 1202–1205, 2012.

[20] B. J. Fischer, J. L. Peña, and M. Konishi, "Emergence of multiplicative auditory responses in the midbrain of the barn owl," *J. Neurophysiol.*, vol. 98, no. 3, pp. 1181–1193, Sep. 2007.

[21] J. A. Fodor and Z. W. Pylyshyn, "Connectionism and cognitive architecture: A critical analysis," *Cognition*, vol. 28, pp. 3–71, 1988.

[22] M. J. Frank and R. C. O'Reilly, "A mechanistic account of striatal dopamine function in human cognition: Psychopharmacological studies with cabergoline and haloperidol," *Behav. Neurosci.*, vol. 120, no. 3, pp. 497–517, Jun. 2006.

[23] F. Galluppi, J. Conradt, T. Stewart, C. Eliasmith, T. Horiuchi, J. Tapson, B. Tripp, S. Furber, and R. Etienne-Cummings, "Live Demo: Spiking ratSLAM: Rat hippocampus cells in spiking neural hardware," in *Proc. IEEE Biomed. Circuits Syst. Conf.*, 2012, DOI: 10.1109/BioCAS.2012.6418493.

[24] F. Galluppi, S. Davies, S. Furber, T. Stewart, and C. Eliasmith, "Real time on-chip implementation of dynamical systems with spiking neurons," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, Jun. 2012, DOI: 10.1109/IJCNN.2012.6252706.

[25] R. W. Gayler, "Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience," in *Proc. ICCS/ASCS Int. Conf. Cogn. Sci.*, Sydney, Australia, 2003, pp. 133–138.

[26] A. P. Georgopoulos, J. T. Lurito, M. Petrides, A. Schwartz, and J. Massey, "Mental rotation of the neuronal population vector," *Science*, vol. 243, pp. 234–236, 1989.

[27] S. Gershman, J. Cohen, and Y. Niv, "Learning to selectively attend," in *Proc. 32nd Annu. Conf. Cogn. Sci. Soc.*, 2010, pp. 1270–1275.

[28] K. Gurney, T. Prescott, and P. Redgrave, "A computational model of action selection in the basal ganglia," *Biol. Cybern.*, vol. 84, pp. 401–423, 2001.

[29] H. Jaeger, "The 'echo state' approach to analyzing and training recurrent neural networks," GMD German Nat. Res. Inst. Comput. Sci., Tech. Rep. 148, 2001.

[30] P. Hagmann, L. Cammoun, X. Gigandet, R. Meuli, C. J. Honey, V. J. Wedeen, and O. Sporns, "Mapping the structural core of human cerebral cortex," *PLoS Biol.*, vol. 6, no. 7, Jul. 2008, e159.

[31] J. Hasler and H. B. Marr, "Finding a roadmap to achieve large neuromorphic hardware systems," *Front. Neurosci.*, vol. 7, no. 118, 2013, DOI: 10.3389/fnins.2013.00118.

[32] D. O. Hebb, *The Organization of Behavior*. New York, NY, USA: Wiley, 1949.

[33] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006.

[34] J. R. Hollerman and W. Schultz, "Dopamine neurons report an error in the temporal prediction of reward during learning," *Nature Neurosci.*, vol. 1, no. 4, pp. 304–309, Aug. 1998.

[35] P. Jonas, G. Major, and B. Sakmann, "Quantal components of unitary EPSCs at the mossy fibre synapse on CA3 pyramidal cells of rat hippocampus," *J. Physiol.*, vol. 472, no. 1, pp. 615–663, 1993.

[36] M. Laubach, M. S. Caetano, B. Liu, N. J. Smith, N. S. Narayanan, and C. Eliasmith, "Neural circuits for persistent activity in medial prefrontal cortex," in *Soc. Neurosci. Abstracts*, 2010, p. 200.18.

[37] P. Lieberman, *Toward an Evolutionary Biology of Language*. Cambridge, MA, USA: Belknap Press, 2006.

[38] P. Lieberman, "The evolution of human speech: Its anatomical and neural bases," *Current Anthropol.*, vol. 48, no. 1, pp. 39–66, 2007.

[39] A. Litt, C. Eliasmith, and P. Thagard, "Neural affective decision theory: Choices, brains, and emotions," *Cogn. Syst. Res.*, vol. 9, pp. 252–273, 2008.

[40] B. Liu, M. Caetano, N. Narayanan, C. Eliasmith, and M. Laubach, "A neuronal mechanism for linking actions to outcomes in the medial prefrontal cortex," presented at the Cogn. Syst. Neurosci. Conf., Salt Lake City, UT, USA, Feb. 24–27, 2011.

[41] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Comput.*, vol. 14, no. 11, pp. 2531–2560, 2002.

[42] D. MacNeil and C. Eliasmith, "Fine-tuning and the stability of recurrent neural networks," *PLoS ONE*, vol. 6, no. 9, Sep. 2011, e22885.

[43] T. V. Maia, "Reinforcement learning, conditioning, and the brain: Successes and challenges," *Cogn. Affective Behav. Neurosci.*, vol. 9, no. 4, pp. 343–364, Dec. 2009.

[44] S. Menon, S. Fok, and K. A. Boahen, "Controlling robot dynamics with spiking neurons," presented at the Neural Inf. Process. Syst. Conf., Lake Tahoe, NV, USA, Dec. 5–8, 2013.

[45] C. Parisien, C. H. Anderson, and C. Eliasmith, "Solving the problem of negative synaptic weights in cortical models," *Neural Comput.*, vol. 20, pp. 1473–1494, 2008.

[46] T. A. Plate, "Holographic reduced representations: Convolution algebra for compositional distributed representations," in *Proc. 12th Int. Joint Conf. Artif. Intell.*, J. Mylopoulos, Ed., 1991, pp. 30–35.

[47] T. A. Plate, "Estimating analogical similarity by dot-products of holographic reduced representations *Advances in Neural Information Processing Systems 6*, J. D. Cowan, G. Tesauro, and J. Alspector, Eds. San Mateo, CA, USA: Morgan Kaufmann, 1994, pp. 1109–1116.

[48] D. Rasmussen and C. Eliasmith, "A neural model of rule generation in inductive reasoning," *Top. Cogn. Sci.*, vol. 3, no. 1, pp. 140–153, 2011.

[49] D. Rasmussen and C. Eliasmith, "A neural reinforcement learning model for tasks with unknown time delays," in *Proc. 35th Annu. Conf. Cogn. Sci. Soc.*, 2013, pp. 3257–3262.

[50] D. Rasmussen and C. Eliasmith, "A spiking neural model that accounts for human performance and cognitive decline on Raven's advanced progressive matrices," *Intelligence*, vol. 42, pp. 53–82, 2014.

[51] D. Rasmussen and C. Eliasmith, "Modeling brain function: Current developments and future prospects," *JAMA Neurol.*, vol. 70, no. 10, pp. 1325–1329, Oct. 2013.

[52] P. Redgrave, T. Prescott, and K. Gurney, "The basal ganglia: A vertebrate solution to the selection problem?" *Neuroscience*, vol. 86, pp. 353–387, 1999.

[53] F. Rieke, D. Warland, R. de Ruyter van Steveninick, and W. Bialek, *Spikes: Exploring the Neural Code*. Cambridge, MA, USA: MIT Press, 1997.

[54] N. C. Rust, V. Mante, E. P. Simoncelli, and J. A. Movshon, "How MT cells analyze the motion of visual patterns," *Nature Neurosci.*, vol. 9, no. 11, pp. 1421–1431, Nov. 2006.

[55] P. Sah, S. Hestrin, and R. A. Nicoll, "Properties of excitatory postsynaptic currents recorded *in vitro* from rat hippocampal interneurones," *J. Physiol.*, vol. 430, no. 1, pp. 605–616, 1990.

[56] R. Salakhutdinov and G. Hinton, "Semantic hashing," presented at the SIGIR Workshop Inf. Retrieval Appl. Graph. Models, Amsterdam, The Netherlands, Jul. 23–27, 2007.

[57] R. Singh and C. Eliasmith, "Higher-dimensional neurons explain the tuning and dynamics of working memory cells," *J. Neurosci.*, vol. 26, pp. 3667–3678, 2006.

[58] T. Stewart and C. Eliasmith, "Compositionality and biologically plausible models," in *Oxford Handbook of Compositionality*. Oxford, U.K.: Oxford Univ. Press, 2011.

[59] T. C. Stewart, X. Choo, and C. Eliasmith, "Dynamic behaviour of a spiking model of action selection in the basal ganglia," in *Proc. 10th Int. Conf. Cogn. Model.*, D. D. Salvucci and G. Gunzelmann, Eds., 2010, pp. 235–240.

[60] T. C. Stewart, F.-X. Choo, and C. Eliasmith, "Symbolic reasoning in spiking neurons: A model of the cortex/basal ganglia/thalamus loop," in *Proc. 32nd Annu. Meeting Cogn. Sci. Soc.*, 2010, pp. 1100–1105.

[61] T. C. Stewart, F. Galluppi, and J. Conradt, "Learning by example with SpiNNaker, Nengo, and an omni-directional robot," presented at the Bernstein Sparks Workshop, Tutzing, Germany, Dec. 2–4, 2013.

[62] T. C. Stewart, "The neural engineering framework," *AISB Quarterly*, pp. 2–7, 2012.

[63] T. C. Stewart, T. Bekolay, and C. Eliasmith, "Neural representations of compositional structures: Representing and manipulating vector spaces with spiking neurons," *Connection Sci.*, vol. 23, no. 2, pp. 145–153, Jun. 2011.

[64] T. C. Stewart, Y. Tang, and C. Eliasmith, "A biologically realistic cleanup memory: Autoassociation in spiking neurons," *Cogn. Syst. Res.*, vol. 12, no. 2, pp. 84–92, Jun. 2011.

[65] T. C. Stewart, B. Tripp, and C. Eliasmith, "Python scripting in the Nengo simulator," *Front. Neuroinf.*, vol. 3, pp. 7–16, Jan. 2009.

[66] R. S. Sutton and A. G. Barto, "Toward a modern theory of adaptive networks: Expectation and prediction," *Psychol. Rev.*, vol. 88, pp. 135–170, 1981.

[67] Y. Tang and C. Eliasmith, "Deep networks for robust visual recognition," presented at the 27th Int. Conf. Mach. Learn., Haifa, Israel, J. Fürnkranz and T. Joachims, Eds., Jun. 21–24, 2010.

[68] J. Tapson and A. van Schaik, "Learning the pseudo inverse solution to network weights," *Neural Netw.*, vol. 45, pp. 94–100, 2013.

[69] J. S. Taube, "The head direction signal: Origins and sensory-motor integration," *Annu. Rev. Neurosci.*, vol. 30, pp. 181–207, Jan. 2007.

ABOUT THE AUTHORS

Terrence C. Stewart received the B.A.Sc. degree in systems design engineering from the University of Waterloo, Waterloo, ON, Canada, in 1999, the M.Phil. degree in computer science and artificial intelligence from Sussex University, Brighton, U.K., in 2000, and the Ph.D. degree in cognitive science from Carleton University, Ottawa, ON, Canada, in 2007.

Currently, he is a Postdoctoral Research Associate in the Department of Systems Design Engineering, Centre for Theoretical Neuroscience, University of Waterloo. His core interests are in understanding human cognition by building biologically realistic neural simulations, and he is currently focusing on language processing and motor control.

Dr. Stewart is a Behavioral & Brain Sciences Associate, a member of the Cognitive Science Society, and a founding member of the Biologically Inspired Cognitive Architecture Society. He also cochaired the 2013 International Conference on Cognitive Modelling, held in Ottawa, ON, Canada.

Chris Eliasmith received the B.A.Sc. degree in systems design engineering and the M.A. degree in philosophy from the University of Waterloo, Waterloo, ON, Canada, in 1994 and 1995, respectively, and the Ph.D. degree in philosophy from Washington University in St. Louis, St. Louis, MO, USA, in 2000.

From 2000 to 2001, he was a Postdoctoral Researcher with C. Anderson in David van Essen's lab at Washington University Medical School. Since then he has been an Assistant Professor, an Associate Professor, and a full Professor at the University of Waterloo. He is currently Director of the Centre for Theoretical Neuroscience, University of Waterloo and holds a Canada Research Chair in Theoretical Neuroscience. He has authored or coauthored two books and over 90 publications in philosophy, psychology, neuroscience, computer science, and engineering venues.

Prof. Eliasmith holds a P.Eng designation and is an Associate Editor of *Cognitive Science*. He is a member of the Society for Neuroscience, the Canadian Philosophical Association, and the Society for Cognitive Science.