AISB Quarterly

THE NEWSLETTER OF THE SOCIETY FOR THE STUDY OF
ARTIFICIAL INTELLIGENCE AND SIMULATION OF BEHAVIOUR

No. 135 October, 2012

The Neural Engineering Framework

The Neural Engineering Framework (NEF) is a general methodology that allows the building of large-scale, biologically plausible, neural models of cognition [1]. The NEF acts as a neural compiler: once the properties of the neurons, the values to be represented, and the functions to be computed are specified, it solves for the connection weights between components that will perform the desired functions. Importantly, this works not only for feedforward computations, but also for recurrent connections, allowing for complex dynamical systems including integrators, oscillators, Kalman filters, etc. [2]. The NEF also incorporates realistic local error-driven learning rules, allowing for the online adaptation and optimisation of responses [3]. The NEF has been used to model visual attention [4], inductive reasoning [5], reinforcement learning [6] and many other tasks. Recently, we used it to build *Spaun*, the world's largest functional brain model, using 2.5 million neurons to perform eight different cognitive tasks by interpreting visual input and producing hand-written output via a simulated 6-muscle arm [7,8]. Our open-source software *Nengo* was used for all of these, and is available at http://nengo.ca, along with tutorials, demos, and downloadable models.

Motivation

Despite the additional constraints and computational overheads involved in building biologically plausible models, there are two major reasons for doing so. First, using biologically realistic neurons not only allows the modelling of behaviour, it also allows the comparison of network properties (e.g., firing patterns, timing effects, and neural connectivity) with real brains and the potential for more accurate investigation of neural degeneration, lesioning, deep brain stimulation, and even various drug treatments.

As an example, when we constructed a NEF implementation of a production system constrained by the properties of the various neuron types found in the brain regions involved, it not only produced the classic 50 millisecond cognitive cycle time *without parameter fitting* [9], it also produced a novel prediction that some types of productions take ~40 milliseconds, while others take ~70 milliseconds, which matches well to some unexplained behavioural data [10].

The second reason for building biologically plausible models is that it can suggest new types of algorithms. The NEF does not produce an exact implementation of whatever algorithm you specify but an approximation, the accuracy of which de-

pends not only on the neural properties but also on the functions being computed. As a consequence, the computations used in a NEF model are constrained by the basic operations of neurons. This has allowed us to make strong claims about the classes of algorithms that cannot be implemented in the human brain (given the constraints on timing, robustness, and numbers of neurons involved) [11].

For example, in attempting to find a plausible implementation of symbol-like cognitive reasoning we were led towards a relatively unexplored family of algorithms which, upon further investigation, we discovered to be particularly useful for induction and pattern completion tasks that are difficult to explain with classical symbol structures [5,11].

In this article I will outline how vectors are encoded into the distributed activity of population of neurons, and how to interpret that activity back into a vector.

Representation

The NEF uses distributed representations and draws a sharp distinction between the activity of a group of neurons and the value (usually thought of as a vector $\mathbf{x}$) being represented. For example, 100 neurons may represent a 2D vector, with different vector values corresponding to different patterns of activity across those neurons.

To map between $\mathbf{x}$ and neuron activity a, every neuron i has an encoding vector $\mathbf{e}_i$ which can be considered the *preferred direction vector* for that neuron (i.e., the vector for which that neuron will fire most strongly). This fits with the general neuroscience methodology of establishing tuning curves for neurons, where the activity of a neuron peaks for some stimulus or condition. The NEF embodies the strong claim that the input current to a neuron is a linear function of the value being represented. If G is the neural non-linearity, α_i is a gain parameter, and β_i is the constant background bias current for the neuron, the neural activity given $\mathbf{x}$ is $a_i = G\left(\alpha_i \mathbf{e}_i \cdot \mathbf{x} + \beta_i\right)$

Importantly, G can be any neural model, including simple rate-based sigmoidal neurons, spiking Leaky-Integrate-and-Fire neurons, or more complex biologically detailed models. The only requirement is that there be some mapping between input current and neuron activity, which can include complex spiking behaviour.

While a vector $\mathbf{x}$ can be converted into neural activity a_i, it is also important to do the opposite. Finding $\mathbf{x}$ given a_i provides a measure of accuracy and a high-level interpretation of spiking activity. NEF does this by finding a set of decoding weights $\mathbf{d}$ such that $\mathbf{x} \approx \sum a_i \mathbf{d}_i$. These weights can be found using any standard er-

ror minimization technique. Crucially for the NEF, these weights are also used to directly solve for the neural connection weights that perform computations.

Computation

For the NEF, any connection between groups of neurons computes a function. The trick is to find a set of connection weights ω_{ij} such that if the first group of neurons (A) represents $\mathbf{x}$ then this will cause the second group (B) to represent $y = f(x)$.

The first step is to imagine an intermediate group of perfectly ideal linear neurons (Figure 1a), with one neuron per vector dimension. If we connect A to these ideal neurons using the connection weights $\mathbf{d}$ found above, then these ideal neurons will be driven to represent the vector $\mathbf{x}$. In general, we can also optimize $\mathbf{d}$ to approximate any function $f(x)$ by adjusting the error minimization. We then connect the ideal neurons to B using the encoder values for group B ($\mathbf{e}_j$). This causes group B to also represent $f(x)$.

Once $\mathbf{d}$ and $\mathbf{e}$ have been found, the intermediate layer is then removed, directly connecting A to B by multiplying the two sets of weights (Figure 1b). This produces the optimal weights to compute an arbitrary function $f(x)$ between A and B.

This approach can be used to approximate any function and has the valuable property that nonlinear functions can be computed with a single layer of connections—no back-propagation of error is required. Not every function can be computed however; the more nonlinear and discontinuous the function, the lower the accuracy. Accuracy is also affected by the neuron properties, such that having a wide variety of neural parameters (as in biological neurons) greatly increases accuracy. This also allows you to determine the neuron properties that would be ideal for particular computations, which can then be used as neurological predictions [1].

It is important to note that the NEF cannot only produce biologically realistic models capable of computing functions of the form $y = f(x)$ but it can also compute dynamic functions of the form $\frac{dx}{dt} = A(x) + B(u)$, where x is the value being represented, u is some input, and A and B are arbitrary functions. A particularly useful special case of this equation is $\frac{dx}{dt} = u$, (an *integrator*) as this sort of component appears in many models of working memory and in accumulator models of decision making.

Symbol Processing

While manipulating vectors is extremely powerful, many cognitive algorithms rely on manipulating symbols with some sort of syntactic structure. How can neurally realistic models possibly represent some-

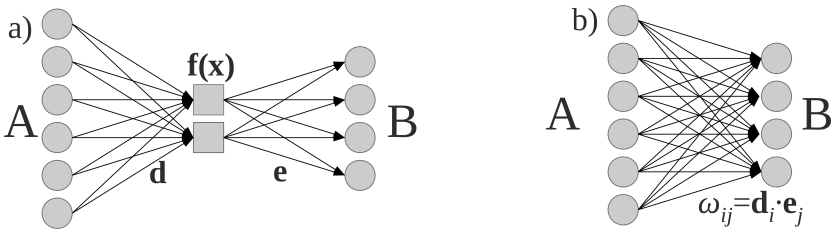

Figure 1: Connecting populations of neurons (circles) via idealized perfectly linear components (squares). (a) **x** is computed from α_i using weights **d**. **x** is then combined with **e** to compute the input current to the next layer of neurons B. (b) Idealized components are eliminated, giving a realistic neuron model functionally identical to (a)

thing like "Dogs chase cats" in such a way as to distinguish it from "Cats chase dogs"? How can we manipulate these representations in useful ways?

It turns out that there are a family of models that already exist for converting symbolic logic into vector manipulations. These are known as Vector Symbolic Architectures [12], and all follow the approach of using high-dimensional vectors for each basic symbol, and then combining these vectors with various mathematical operations to produce new vectors that encode full symbol structures. Unlike ideal classic symbol systems however, VSAs are lossy, in that as the symbol tree structure gets more complex, the accuracy of extracting the original vectors from that combined vector gradually decreases.

Furthermore, the vectors maintain similarity, so that if "pink" and "red" have similar vectors, then "pink square" and "red square" will also have similar vectors. This feature allows inductive reasoning over complex patterns. For example, our neural model of the Raven's Progressive Matrix task (a standard intelligence test where participants are given 8 visual patterns in a 3×3 grid and are asked to determine what pattern should be placed in the missing square) works by forming the vector representation of each pattern and computing the average transformation that that will take one pattern to the next [5].

As a simple example of this approach, you can create high-dimensional (~500 dimensions for adult-level vocabularies) unit vectors for each basic symbol (DOG, CAT, CHASE, SUBJECT, OBJECT, VERB, etc.). These can be cho-

sen randomly, or so as to reflect standard similarity measures. Two operations are required to create a symbol structure: addition (+) and circular convolution (⊗). The sentence "Dogs chase cats" would then be S = DOG ⊗ SUBJECT + CHASE ⊗ VERB + CAT ⊗ OBJECT). Given this sentence, a particular component can be extracted by computing S ⊗ SUBJECT^{-1} ≈ DOG, where the inverse operation is a simple reordering of the elements in the vector. Interestingly, while circular convolution seems like a complicated operation, you can break it down into a linear transformation, a large number of pairwise multiplications, and another linear transformation. All of these operations are accurately approximated by the NEF methods.

Spaun

The ability to perform symbol-like manipulations using vectors allows you to build very large-scale cognitive models. Our largest model to date is Spaun, a 2.5 million spiking neuron model with a vision system (formed by implementing a Restricted Boltzmann Machine Deep Belief Network with the NEF), a single 6-muscle 3-joint arm for output, and a selective routing system (analogous to a production system) implemented in spiking neurons comprising the cortex (for working memory storage), the basal ganglia (for action selection), and the thalamus (for selectively routing information between cortical areas) [7]. Various other cortical areas are also modeled, allowing for transformations between visual, conceptual, and motor spaces, inductive pattern finding, and list memory. The model is capable of performing eight different psychological tasks, including recognizing hand written digits, memorizing digit lists and recalling particular items, pattern completion, reinforcement learning, and mental addition. No changes to the model are made between tasks: instead, a visual input is provided telling the model which task to perform next. We are aware of no other realistic neural model with this combination of flexibility and biological realism.

Nengo

Nengo is an open-source cross-platform Java application which implements the NEF and can be used as both a teaching tool (with hands-on classroom demos) and a research tool (all of our large-scale models are built with it, including Spaun). Neural groups can be created through a drag-and-drop interface or Python scripting. The functions to approximate are similarly specified, with Nengo automatically computing the optimised connection weights. Also included is a visualisation interface for viewing and interacting with running models, including support for

simulated environments and physical robots. The software, extensive documentation, and various tutorials are available at http://nengo.ca.

References

1. Eliasmith, C., & Anderson, C. H. (2003). *Neural engineering: Computation, representation and dynamics in neurobiological systems.* Cambridge, MA: MIT Press.

2. Eliasmith, C. (2005). A unified approach to building and controlling spiking attractor networks. *Neural computation. 7*, 1276–1314.

3. MacNeil, D., & Eliasmith C. (2011). Fine-tuning and the stability of recurrent neural networks. *PLoS ONE. 6(9).*

4. Bobier, B., Stewart T. C., & Eliasmith C. (2011). *The attentional routing circuit: receptive field modulation through nonlinear dendritic interactions.* Cognitive and Systems Neuroscience Poster.

5. Rasmussen, D., & Eliasmith, C. (2011). A neural model of rule generation in inductive reasoning. *Topics in Cognitive Science, 3*, 140–153.

6. Stewart, T.C., Bekolay, T., & Eliasmith, C. (2012). Learning to select actions with spiking neurons in the basal ganglia. *Frontiers in Decision Neuroscience, 6.*

7. Stewart, T., Choo, F-X, & Eliasmith, C. (2012). Spaun: A perception-cognition-action model using spiking neurons. *Proceedings of the 34th Annual Conference of the Cognitive Science Society.*

8. Eliasmith, C. (2013). *How to build a brain: A neural architecture for biological cognition.* New York: Oxford University Press.

9. Stewart, T.C., Choo, F-X., & Eliasmith, C. (2010). Dynamic behaviour of a spiking model of action selection in the basal ganglia. In *Proceedings of the 10th International Conference on Cognitive Modeling*, 235–240.

10. Gunzelmann, G., Moore, R., Salvucci, D., & Gluck, K. (2011). Sleep loss and driver performance: Quantitative predictions with zero free parameters. *Cognitive Systems Research, 12(2)*, 154–163.

11. Stewart, T., & Eliasmith C. (2012). Compositionality and biologically plausible models. *Oxford Handbook of Compositionality.*

12. Gayler, R. (2003). Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience. *ICCS/ASCS International Conference on Cognitive Science*, Sydney, Australia: University of New South Wales. 133–138.

Terry Stewart PhD
Centre for Theoretical Neuroscience,
University of Waterloo, Canada