Supervised Learning of Action Selection in Cognitive Spiking Neuron Models

Terrence C. Stewart (tcstewar@uwaterloo.ca)
Sverrir Thorgeirsson (sverrir.thorgeirsson@uwaterloo.ca)
Chris Eliasmith (celiasmith@uwaterloo.ca)
Centre for Theoretical Neuroscience, University of Waterloo
200 University Avenue West, Waterloo, ON, Canada, N2L 3G1

Abstract

We have previously shown that a biologically realistic spiking neuron implementation of an action selection/execution system (constrained by the neurological connectivity of the cortex, basal ganglia, and thalamus) is capable of performing complex tasks, such as the Tower of Hanoi, n-Back, and semantic memory search. However, because the neural implementation approximates a strict rule-based structure of a production system, such models have involved hand-tweaking of multiple parameters to get the desired behaviour. Here, we show that a simple, local, online learning rule can be used to learn these parameters, resulting in neural models of cognitive behaviours that are more reliable and easier to construct than with prior methods.

Keywords: neural engineering framework; neural production systems; semantic pointer architecture; spiking neurons; basal ganglia; neural cognitive architectures

Introduction

In previous work (Stewart & Eliasmith, 2009; Stewart, Choo, & Eliasmith, 2010), we have shown how spiking neurons can be used to build biologically plausible approximations of traditional production systems, and that these models also harness the sophisticated pattern matching capabilities of neural networks, leading to novel capabilities. This formed the core of Spaun (Eliasmith et al., 2012), the first and so far only spiking neuron model capable of performing multiple cognitive tasks, and is central to the more general Semantic Pointer Architecture for building neually plausible cognitive models (Eliasmith, 2013). Since then, this system has formed an important part of biological models of the Tower of Hanoi (Stewart & Eliasmith, 2011), bandit tasks (Stewart, Bekolay, & Eliasmith, 2012), command parsing (Stewart & Eliasmith, 2013), sentence parsing (Stewart, Choo, & Eliasmith, 2014), the n-Back task (Gosmann & Eliasmith, 2015), action planning (Blouw, Eliasmith, & Tripp, 2016), speech production (Kröger, Bekolay, & Blouw, 2016) and the effects of reduction of dopamine on speech production (Senft et al., 2016), hierarchical reinforcement learning of navigation and abstract action rules (Rasmussen, Voelker, & Eliasmith, 2017), and semantic memory search (Kajić et al., 2017).

However, building neural implementations of these kinds of cognitive tasks imposes new challenges on the researcher. In particular, since the neurons *approximate* a production system, they do not provide the same simple algorithmic programmability that a production system affords with its IF-THEN rules. This has meant that creating the models listed above required careful hand-tuning of some parameters.

In this paper, we present a method for automatically learning these parameters such that the model's desired overall overt behaviour is achieved. This greatly simplifies the process of constructing neural models with complex rule-like behaviour, and sheds light on potential mechanisms for how neurobiological systems may learn to perform such tasks.

Neural Action Selection and Execution

The generic form of our neural approximation of a production system is shown in Figure 1. The neurons in cortex represent state information, i.e., the set of information that can be used for selecting which action to perform next. This can consist of visual information, the contents of working memory, the state of the motor system, and so on. (These approximately correspond to Buffers in ACT-R). We use the Neural Engineering Framework (Eliasmith & Anderson, 2003) to organize spiking neurons to form distributed representations of these values, which can be scalars, vectors, or functions.

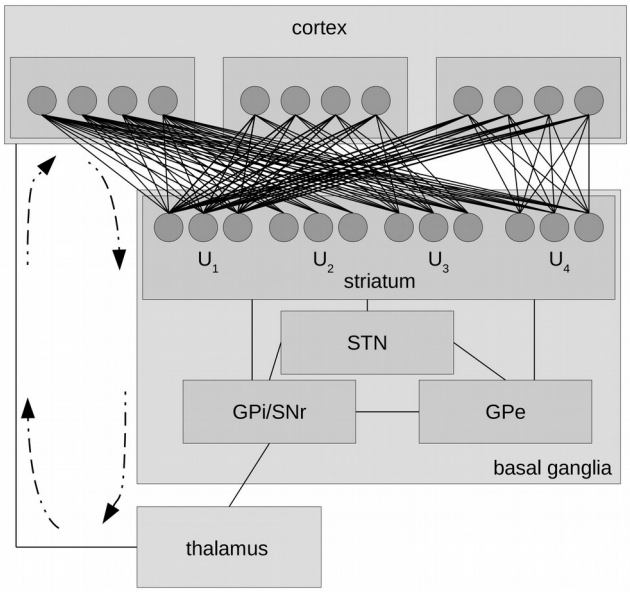

Figure 1: The cortex-basal ganglia-thalamus loop that forms the neural action selection and execution system. Neurons (dark circles) and connections are shown only for the inputs to the action selection system. These connections compute the utility (U_i) of each of the actions i, given the current cortex state. The basal ganglia selects the action with the highest utility, and the thalamus executes that action.

The connections between cortex and the first stage of the basal ganglia (the striatum) compute the *utility* of each action. The basal ganglia determines the largest of these utility values, which is the action that should be selected. Connections via the thalamus execute the action, resulting in changed cortical state.

More specifically, the neurons in the different cortical areas form a distributed representation of whatever *state* information is needed for the model. This state is, in general, a D-dimensional vector represented by the spiking activity of N neurons (where N is generally much larger than D). In Figure 1, we show three groups of four neurons each, representing three different state variables. In a typical model, these state variables would consist of ~25,000 neurons representing a ~500 dimensional value. Given such a large state space, we can represent symbols by using randomly chosen 500-dimensional vectors to represent different concepts (A, B, C, ONE, TWO, THREE, LETTERS, NUMBERS, etc.). Importantly, by using the approach of Vector Symbolic Architectures (Gayler, 2003), such a representation can also be generalized to represent complex combinations of symbols, giving the compositionality needed for cognitive models. The details of the construction and deconstruction of symbol-like structures in these systems are not needed for the purposes of this paper, but see (Stewart, Choo, & Eliasmith, 2010) and (Eliasmith, 2013) for further details.

The striatum neurons in the input to the basal ganglia represent the utility of each action. In Figure 1, we show four actions, and each action's utility is represented by three neurons. In the real model, we use 100 neurons per action (including both striatal D1 and D2 neurons). These 100 neurons again form a distributed representation of the utility of the action given the current cortical state information.

To cause this to happen, we need to determine the connection weights between the cortical neurons and the striatal neurons such that when the cortical neurons fire with a particular pattern, the striatal neurons will fire with the pattern that represents the correct utility value.

For example, suppose we have an action that should occur if the visual cortical neurons are representing the number TWO. We might then say that we want the connections from cortex to striatum to approximate the following function, where s is the vector currently represented in cortex:

$$U_1 = \begin{cases} 1, & \text{if } s = TWO \\ 0, & \text{otherwise} \end{cases}$$

However, neural firing patterns are never going to be exact, so the cortical neurons will never (or very rarely) fire with the exact ideal pattern that we have chosen to mean TWO. This means the above function will, in practice, always return 0 and so the connection weights that closely approximate that function will be all zeros. Instead, we want a function that will give a high utility if the represented value is *close* to TWO, and a low utility if it is farther from TWO. For this, we use the dot product:

$$U_1 = s \cdot TWO$$

We use the Neural Engineering Framework (NEF; Eliasmith & Anderson, 2003) to directly solve for the ideal connection weights that best approximate this function. Importantly, the NEF demonstrates that linear functions are extremely easy for neural connections to approximate, so we know that this function will be well-approximated.

The above technique (using neurons to represent vectors, and solving for connection weights that approximate functions on those vectors) is used for all the neurons and connections in the model. In previous work (e.g., Stewart, Choo, & Eliasmith, 2010; Eliasmith, 2013) we have shown how the basal ganglia model finds the largest of these utility values and how the thalamus model is used to route information between cortical areas to execute the actions. However, in this paper we take a closer look at the problems encountered when deciding upon the functions to be approximated to compute these utilities.

Approximating AND and OR

Suppose we want an action that will only occur if the visual cortex contains LETTER and the contents of working memory are A. In a standard production system, this is simple to specify, since AND is a primitive operation when defining production rules. However, if we instead try to specify this operation using neurons, it is difficult for the neurons to be precise. Rather, we might get the neurons to approximate this function:

$$U_2 = 0.5(v \cdot LETTER + m \cdot A)$$

Now, the utility will only be large (near 1) if both the visual cortex v is near the pattern for LETTER and the working memory neurons m are near the pattern for A. This seems to be a good implementation of AND, but it is important to note that if the memory contains B instead of A, this action *will still get chosen if no other actions have a utility higher than 0.5*. In other words, this attempt at implementing AND is highly dependent on the definition of the *other actions*. As the number of actions increases, the interaction between the various actions becomes more complex.

A similar problem occurs with OR. If we want an action that occurs if v is NUMBER or if m is C, we may compute the utility with something like this:

$$U_3 = v \cdot NUMBER + m \cdot C$$

This will produce a large value (near 1) if either case is true. However, if it turns out that both v is near NUMBER and m is near C, then this action will get an even higher utility (near 2), causing it to overwhelm other actions that may also have large utilities. Indeed, this action may be selected even if v is only somewhat near NUMBER and m is only somewhat near C.

These problems may be mitigated by adding scaling factors to these equations. For example, we may do the following:

$$U_3 = 0.9(v \cdot NUMBER + m \cdot C)$$

It is these scaling factors that are sometimes "hand-tweaked" when making complex neural action selection

models using our methods. We now present a method for eliminating this hand-tweaking by having the neural network learn the correct connection weights to perform the task correctly.

Supervised Learning of Action Selection

Rather than hand-tweaking the scaling factors in the U functions (that is, trying to find values that consistently produce the desired behaviour in the context of all the other U functions), we instead propose to use an online learning rule. That is, we initialize the model with zero for these parameters, and then adjust these values while the model is running based on its performance.

The simplest online error-driven learning rule is the delta rule (Widrow & Hoff, 1960):

$$\Delta\omega_{i,j} = \alpha x_i (t_j - y_j)$$

This is meant for situations where the adjustable parameters ω_{ij} are weights on values x_i that produce the weighted sum y_j (i.e. $y_j = \sum_i \omega_{i,j} x_i$). The desired target value is t_j and the learning rate is α. We have previously shown how this rule can be adapted to distributed representations in spiking neurons (Bekolay, Kolbeck, & Eliasmith, 2013), where we refer to it as the PES rule.

Since this is exactly the right configuration for the parameters in our utility equations U, we can directly apply this learning rule, if we can determine a target value t_j at every point in time while the simulation is running.

Determining the error signal (t_j - y_j)

To apply the above rule, we need a measure of the error that is currently being made. To get the target value, we need to know what the utilities *should* be right now. However, this is exactly what we don't know (since if we did know what the utilities of each action should be, we would just use that as the equation for U). However, what we can compute is *what would an ideal action selection system do in this case*. That is, we can use a standard production system (or any other appropriate action selection system) given the current state information and see what action it would take. This standard production system is not constrained to be neurally plausible, so it can do perfect AND and OR operations. We then use this to form an estimate of t_j:

$$t_j = \begin{cases} 1, & \text{if an ideal production system would chose this action right now} \\ 0, & \text{otherwise} \end{cases}$$

For y_j we have two options. The standard approach would be to simply use $y_j = \sum_i \omega_{i,j} x_i$ (i.e. the utility values as currently computed by the neural system). This is what this learning rule was designed for. However, given our estimate of t_j, this may not be the right rule. For example, if action 1 has a utility of 0.9 and all the other actions have a utility smaller than 0.9, then action 1 will be selected, but even if action 1 is supposed to be selected, this learning rule would treat that as an error and try to adjust the parameters such that the utility is larger. That is, even when the system is producing the correct behaviour, the weights will still be adjusted.

To avoid this, we can also estimate y_j as the *output of the action selection system*. That is,

$$y_j = \begin{cases} 1, & \text{if this action is currently selected by the neural system} \\ 0, & \text{otherwise} \end{cases}$$

This gives us two possible ways of measuring the error: we can either use the input to the action selection system (U_j) or we can use the output of the action selection system (y_j, above). Both cases are investigated here.

Determining the learning inputs x_i

We must also determine what to use for x_i. One straightforward option is to use the base terms from the equations used to generate the U functions. For example, if one of our U functions is the above-mentioned utility function $U_3 = 0.9(v \cdot NUMBER + m \cdot C)$, then we might use $v \cdot NUMBER$ and $m \cdot C$ as x_1 and x_2. The learning rule would then learn the weights $\omega_{1,3}$ and $\omega_{2,3}$. In a full system with multiple utility functions, all of the different terms would create a long list of x values.

However, since the cortical values are already stored in terms of neurons, and those values such as $v \cdot NUMBER$ are being computed by connection weights from those neurons, there is a second alternative. We can use the same learning rule, with the same error signal, to *directly adjust the connection weights from the neurons themselves*. That is, we let x_i be the neural activity of the cortical neurons, and ω_{ij} is then the connection strength between the cortical neurons and the striatal neurons.

By using the neurons themselves, we are increasing the range of possible functions that the learning rule can find. That is, when using the parameter-based approach, the U functions are constrained to the linear weighted sums of the particular terms that we have identified for x_i. But, if we use the neural activity for x_i, then the learning rule has access to the full space of possible functions that can be approximated by connection weights out of those neurons. Interestingly, using the learning rule in this way makes it mathematically identical to the PES learning rule that we have previously used to model reinforcement learning, and in that case it was also used to learn the connections between cortex and striatum (Stewart, Bekolay, & Eliasmith, 2012).

However, the drawback of using the neurons themselves is that we greatly increase the number of parameters ω_{ij} to learn. This makes the learning more computationally intensive, and will likely require a lower learning rate. Both options are investigated here.

Example Model

To test this approach to learning, we chose a simple cognitive task with two cortical state variables and six actions. If the visual system contains LETTER, then the production system should cycle the working memory state through A → B → C → A (and so on). If the visual system contains NUMBER, then the working memory should cycle through ONE → TWO → THREE → ONE (and so on).

This can be thought of as the following set of production rules:

IF	THEN
v=LETTER AND *m*=A	*m*=B
v=LETTER AND *m*=B	*m*=C
v=LETTER AND *m*=C	*m*=A
v=NUMBER AND *m*=ONE	*m*=TWO
v=NUMBER AND *m*=TWO	*m*=THREE
v=NUMBER AND *m*=THREE	*m*=ONE

When the system is run, an external input to the *visual* cortical neurons is set to change it from LETTER to NUMBER (and back) every second.

To convert this into a neural model, we need to define the utility functions for each action. If we were doing this by hand using the typical approach, we might use the following:

$$U_1 = 0.5v \cdot LETTER + 0.5m \cdot A$$
$$U_2 = 0.5v \cdot LETTER + 0.5m \cdot B$$
$$U_3 = 0.5v \cdot LETTER + 0.5m \cdot C$$
$$U_4 = 0.5v \cdot NUMBER + 0.5m \cdot ONE$$
$$U_5 = 0.5v \cdot NUMBER + 0.5m \cdot TWO$$
$$U_6 = 0.5v \cdot NUMBER + 0.5m \cdot THREE$$

However, performance may improve if those parameters are tweaked, and indeed if other terms are added.

Evaluation Metrics

To determine whether the optimization does, in fact, improve performance, we define two separate measures to characterize the quality of the model.

First, we report the number of correct transitions over 2.0 seconds (1.0 seconds with the input to *visual* being LETTER, and 1.0 with it being NUMBER). However, we have to be careful as to how to define a correct transition, since neurons are being used to represent the contents of working memory, and those contents are a numerical vector, not an abstract symbol. There is, however, an ideal (randomly chosen) vector for each of the basic terms (ONE, TWO, THREE, A, B, C, LETTER, NUMBER). Furthermore, given the neural activity of the working memory neurons, we can compute the vector they are currently representing (using the NEF). This numerical vector can then be compared to the ideal vectors. Here, we use the dot product for this comparison. We define a correct transition as going from a previous time point where, for example, the vector is closest to A, and at the next time point it is closest to B. If LETTER is currently in *visual*, then the correct transitions are A→B, B→C, and C→A. For NUMBER they are ONE→TWO, TWO→THREE, and THREE→ONE. For this metric, we simply count the number of times this correct transitioning occurs.

For the second metric, we determine the proportion of time that the working memory contains the correct type of value. That is, if the visual cortex neurons are representing LETTER, then the vector in working memory should be closer to A, B, or C than it is to ONE, TWO, or THREE (and vice-versa for NUMBER). This second metric is simply the proportion of time this is true.

When computing the metrics, we always measure when learning is off. That is, we perform one cycle with learning on, and then one cycle with learning off where we actually compute the metrics. With this approach, we ensure that the measured performance is based on the system actually learning to respond correctly to the input, rather than responding based on the training signal itself.

Results

We start by learning using the input to the action selection system to compute the error, and using the separate mathematical terms to provide a small set of parameters ω_{ij} to learn. This is shown in Figure 2.

We note that slower learning rates lead to improved performance on the first metric (number of correct transitions), but worse performance on the second metric (percentage of time spent in the correct states). Furthermore, the model reaches a peak performance of around 20 transitions per cycle, and does not improve with more training.

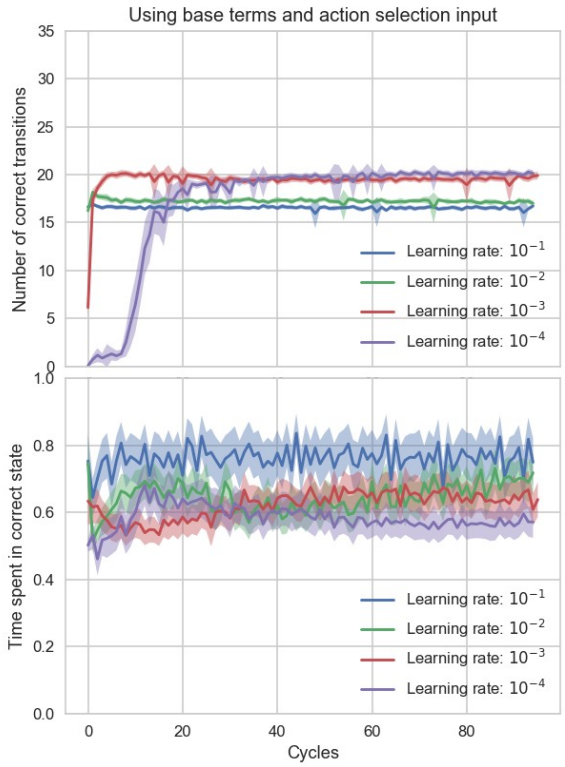

Figure 2: Effects of different learning rates when learning from the base terms and the action selection input. Shaded areas are 95% bootstrap confidence intervals.

Figure 3 shows the model when we use neurons for learning and the action selection input for the error value. Here we find a marked improvement in both metrics, although with too low a learning rate the system does not improve.

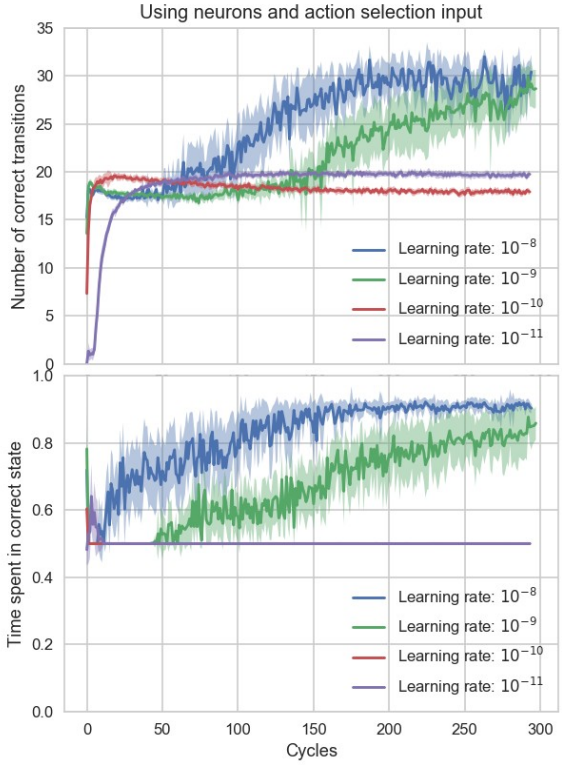

Figure 3: Effects of different learning rates when learning from the base terms and the action selection input. Shaded areas are 95% bootstrap confidence intervals.

Figures 4 and 5 show the same systems as 2 and 3, but with the output of the action selection system as the source of the error in the training signal. Both of these show extremely poor performance, including effects where performance increases temporarily, but then decreases back down to the (poor) baseline performance.

Conclusions and Future Work

The presented results indicate that we can use online supervised learning to learn the utility functions needed for cognitive models that use neural action selection.

Interestingly, we found the best performance when learning directly off of the neurons themselves. This makes the learning process harder (in terms of computational power requirements and the number of ω_{ij} terms that must be learned), but it also greatly reduces the effort required by the modeller. In particular, rather than having to define the basic terms in the utility functions (e.g. $v \cdot NUMBER$), the learning system will directly deal with the neural representations, allowing it to find much more complex functions that produce the desired behaviour more reliably.

Oddly, using the output from the action selection system for training does not work at all. This is somewhat surprising, and more work is needed to investigate this.

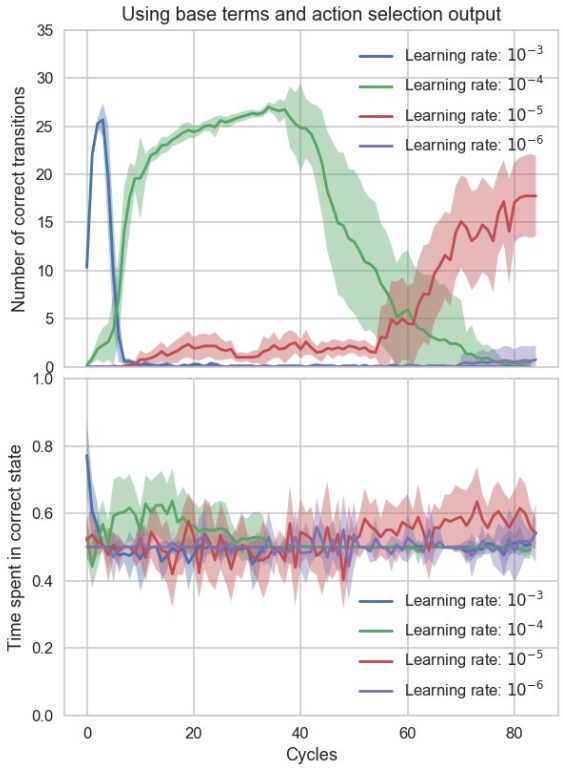

Figure 4: Effects of different learning rates when learning from the base terms and the action selection output. Shaded areas are 95% bootstrap confidence intervals.

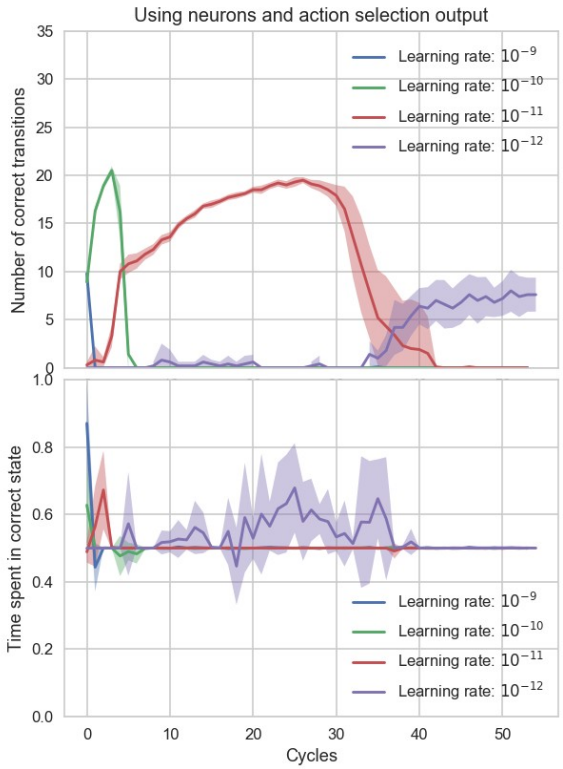

Figure 5: Effects of different learning rates when learning from the base terms and the action selection output. Shaded areas are 95% bootstrap confidence intervals.

It should also be noted that, as in many learning systems, the learning rate itself is extremely important. Too small a learning rate leads to systems that take long amounts of time (and computational power) to learn (or do not learn at all). Too high a learning rate and the system does not reliably improve. Fortunately, there are numerous techniques in the machine learning literature for dealing with this difficulty, (e.g., changing learning rates based on performance).

More importantly, however, we next need to evaluate this learning system on more complex cognitive tasks, such as our existing the Tower of Hanoi and left-corner parser models. This will provide a more robust test of the practical benefits of the learning system.

Finally, there is a deeper theoretical question and possibility that needs to be examined further. In particular, can the supervised learning system presented here be considered of a model of how people learn complex cognitive tasks? All we have shown so far is that this learning system is a useful tool for constructing neural cognitive models. We could simply treat it as such a technical tool that provides a novel way of more easily generating neural cognitive models that perform desired tasks. However, the fact that this learning system applies directly to the very same neural connections between the cortex and striatum that are used in models of reinforcement learning, and the fact that the learning rule itself is one that is local and biologically plausible, suggests that perhaps the technique we have presented here could be biologically implemented. That is, rather than treating the resulting model as a simulation of someone who has *previously learned the task* (as is common in both our neural models and in non-neural cognitive modelling), we can treat this supervised learning system as a model of *the process of learning the task*. However, in order to do this, we would need to tackle one significant question: where does this ideal target value t_j come from? That is, what neural mechanism provides an indication of what action ought to be performed next? It seems possible that some combination of mental models and learning by instruction may be able to provide such a training signal, but more research must be done to investigate this possibility.

Acknowledgments

This work was supported by AFOSR grant FA9550-17-1-0026, NSERC Discovery grant 261453, and the Canada Research Chairs program.

References

Bekolay, T., Kolbeck, C., and Eliasmith, C. (2013). Simultaneous unsupervised and supervised learning of cognitive functions in biologically plausible spiking neural networks. *35th Annual Conference of the Cognitive Science Society*, 169–174.

Blouw, P., Eliasmith, C, and Tripp, B. (2016). A scaleable spiking neural model of action planning. *38th Annual Conference of the Cognitive Science Society*

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

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

Eliasmith, C., Stewart, T.C., Choo, X., Bekolay, T., DeWolf, T, Tang, Y., Rasmussen, D. (2012). A large-scale model of the functioning brain. *Science,* 388:6111, 1202-1205.

Gayler, R. (2003). Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience. *International Conference on Cognitive Science*.

Gosmann, J. and Eliasmith, C. (2015). A spiking neural model of the n-back task. *37th Annual Meeting of the Cognitive Science Society.*

Kajic, I., Gosmann, J., Komer, B., Orr, R., Stewart, T.C., and Eliasmith, C. (2017). A Biologically Constrained Model of Semantic Memory Search. *Annual Meeting of the Cognitive Science Society.*

Kröger, B.J., Bekolay, T., and Blouw, P. (2016). Modeling motor planning in speech production using the neural engineering framework. *Electronic Speech Signal Processing (ESSV),* 15–22.

Rasmussen, D., Voelker, A.R., and Eliasmith, C. (2017). A neural model of hierarchical reinforcement learning. *PLoS ONE,* 12:7, 1–39.

Senft, V., Stewart, T.C., Bekolay, T., Eliasmith, C., and Kröger, B.J. (2015). Reduction of dopamine in basal ganglia and its effects on syllable sequencing in speech: a computer simulation study. *Basal Ganglia* 6:1, 7-17.

Stewart, T.C., Bekolay, T., and Eliasmith, C. (2012) Learning to select actions with spiking neurons in the basal ganglia. *Frontiers in Neuroscience,* 6:2, 1-14.

Stewart, T.C., Choo, X., and Eliasmith, C. (2010). *Symbolic reasoning in spiking neurons: A model of the cortex/basal ganglia/thalamus loop.* Annual Meeting of the Cognitive Science Society.

Stewart, T.C., Choo, X., and Eliasmith, C. (2014). Sentence processing in spiking neurons: A biologically plausible left-corner parser. *Annual Meeting of the Cognitive Science Society.*

Stewart, T.C. and Eliasmith, C. (2009) *Spiking neurons and central executive control: The origin of the 50-millisecond cognitive cycle.* International Conference on Cognitive Modelling.

Stewart, T.C. and Eliasmith, C. (2011) Neural cognitive modelling: A biologically constrained spiking neuron model of the Tower of Hanoi task. *33rd Annual Meeting of the Cognitive Science Society.*

Stewart, T.C. and Eliasmith, C. (2013). Parsing Sequentially Presented Commands in a Large-Scale Biologically Realistic Brain Model. *35th Meeting of the Cognitive Science Society.*

Widrow, B. and Hoff, M. E. Jr. (1960). Adaptive switching circuits. *IRE Western Electric Show and Convention Record, Part 4*, 96-104.