

# General Instruction Following in a Large-Scale Biologically Plausible Brain Model

Xuan Choo (fchoo@uwaterloo.ca)

Chris Eliasmith (celiasmith@uwaterloo.ca)

Center for Theoretical Neuroscience, University of Waterloo

Waterloo, ON, Canada N2L 3G1

## Abstract

We present a spiking neuron brain model implemented in 318,870 LIF neurons organized with distinct cortical modules, a basal ganglia, and a thalamus, that is capable of flexibly following memorized commands. Neural activity represents a structured set of rules, such as “If you see a 1, then push button A, and if you see a 2, then push button B”. Synaptic connections between these neurons and the basal ganglia, thalamus, and other areas cause the system to detect when rules should be applied and to then do so. The model gives a reaction time difference of 77 ms between the simple and two-choice reaction time tasks, and requires 384 ms per item for sub-vocal counting, consistent with human experimental results. This is the first biologically realistic spiking neuron model capable of flexibly responding to complex structured instructions.

**Keywords:** neural engineering; spiking neuron model; instruction following; instruction processing; cognitive control; cognitive architectures

## Introduction

One of the hallmarks of complex cognition is the ability to perform a multitude of tasks using the same underlying architecture. When given an instruction, the human brain is capable of processing and executing the instruction without the need for extensive rewiring of the underlying neural connections. As far as we are aware, no neural model to date has been shown to exhibit this ability.

Eliasmith et al. (2012) describes what is currently the world’s largest functional brain model. While the model, called Spaun (for Semantic Pointer Architecture Unified Network), is able to perform 8 different cognitive tasks without necessitating changes to its architecture, the knowledge needed to complete these 8 tasks is hard-coded into the action selection mechanism (the basal ganglia) of the model, making it unable to perform any task other than the predefined 8. In this paper, we propose an extension to the Spaun action selection component making it capable of processing generic instructions.

## Terminology

Four key concepts are discussed in this paper: states, actions, rules, and instructions.

**States** are internal variables that the action selection system monitors to figure out what is the best action to perform. States can be both internal (e.g. goal memories, working memories (WM)) and external (e.g. visual input) to the system.

**Actions** are atomic commands within the architecture, and are typically motor commands (e.g. “write the number  $X$ ”, “push the  $X$  button”) or cognitive commands (e.g. “remember the word  $X$ ”, “route information from WM area  $X$  to WM area

$Y$ ”, “add 1 to the value in WM area  $X$ ”). Apart from motor and cognitive commands, actions can also be utilized to change the values of the model’s **states**.

**Rules** are conditional statements typically of the form “IF  $X$ , THEN  $Y$ ” (e.g. “If you see a 1, then push button A”) where  $X$  is a set of conditions which have to be met for the set of **actions**  $Y$  to be executed. More generally, in Spaun, rules are statistical maps between cortical states and actions.

An **instruction** is a combination of **rules** or **actions** that can be executed sequentially (e.g. “Remember the number 1; add 1 to that number; write the result”) or in any order (e.g. “If you see a 1, then push button A; If you see a 2, then push button B”).

## Spaun

The architecture of Spaun (the Semantic Pointer Architecture, or SPA) is composed of 9 distinct but interconnected modules (see Figure 1A). Of interest to this paper is how the action selection module interacts with the rest of the model. Fundamentally, the action selection module of Spaun is identical to the basal ganglia (BG) based production system described in (Stewart, Bekolay, & Eliasmith, 2012), and functions similarly to the action selection component of production system models (e.g. (Anderson, 1996)).

In these systems, action selection is hard-coded by a predefined set of **rules**. To select an **action**, the BG monitors internal cortical state variables and executes a rule whose antecedent best matches the values of the internal state variables (see Figure 1B). Critically, to encode **instructions**, the transitions between each **rule** in the **instruction** has to be hard-coded into the BG as well. For example, if the instruction was to perform *ACTION-A* followed by *ACTION-B*, and then *ACTION-C*, the following rules would have to be encoded into the BG:

IF *INIT*, THEN *state* = *ACTION-A*

IF *state* = *ACTION-A*, THEN *state* = *ACTION-B*

IF *state* = *ACTION-B*, THEN *state* = *ACTION-C*

Several ACT-R models (e.g. (Taatgen & Lee, 2003), (Taatgen, 1999)) able to follow instructions, however no neural implementation has been previously discussed.

Aside from its architecture, Spaun is also unique in the way information is represented. Information is encoded and represented using semantic pointers (Eliasmith, In Press). These representations are used in the SPA to define a type of vector symbolic architecture (VSA). In typical VSAs, the vector

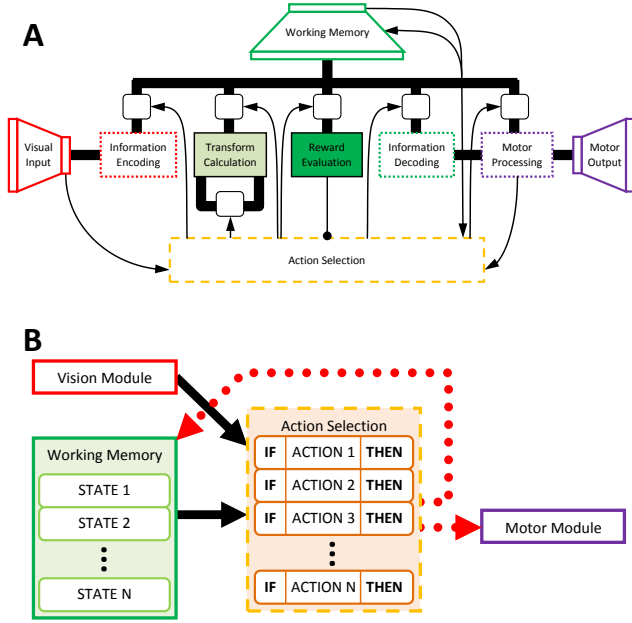


Figure 1: A) High-level architecture of Spaun. B) Method by which Spaun chooses an action. The action selection system monitors cortical state variables (solid arrows), selects an action that best matches these states, and effects the action on efferent modules (dotted arrows).

that represents the number **ONE** and the vector that represents the number **TWO** would be chosen from a random distribution and thus have no direct relation to each other. In the SPA however, the semantic pointer for the number **TWO** is computed as the bound combination of the semantic pointer **ONE** with a vector that represents the concept **ADD1**, thus imparting semantic meaning to each vector:

$$\mathbf{TWO} = \mathbf{ONE} \otimes \mathbf{ADD1}$$

Similarly, the semantic pointer for the number **THREE** can be computed as follows:

$$\begin{aligned} \mathbf{THREE} &= \mathbf{TWO} \otimes \mathbf{ADD1} \\ &= \mathbf{ONE} \otimes \mathbf{ADD1} \otimes \mathbf{ADD1} \end{aligned}$$

### Vector Symbolic Architectures

Vector symbolic architectures have four core properties. First, information is represented by high-dimensional vectors usually chosen from a random distribution.

Second, vectors can be combined using a superposition operation (denoted with a  $+$ ). Of note, the vector result of the superposition operation is similar to the original vector operands, where similarity is measured by a dot product.

Third, vectors can be bound together using a binding operation (denoted with a  $\otimes$ ). Unlike the superposition operator, the vector result of the binding operation is dissimilar to the original vector operands.

Last, an approximate inverse operator (denoted with  $*$ , such that  $A^*$  is the approximate inverse of  $A$ ) is defined such

that binding  $A$  with  $A^*$  results in approximately the identity vector  $\mathbf{I}$  ( $A \otimes A^* \approx \mathbf{I}$ ). This property of the approximate inverse can be used to unbind previously bound vectors.

Both the superposition and binding operations are analogous to addition and multiplication in scalar mathematics, and are often associative, commutative, and distributive.

In the SPA, vector addition is used for superposition, and circular convolution is used for binding, bearing close similarity to the Holographic Reduced Representation (Plate, 2003).

### Encoding Instructions

Instructions are encoded using a positional encoding schema similar to that used in Spaun and in the Ordinal Serial Encoding model of serial working memory (Choo, 2010). Each rule in the instruction is tagged (bound) to a position vector to indicate its relative order within the instruction. For example, the instruction “1. **RULE1**; 2. **RULE2**” is encoded as

$$\mathbf{INSTR} = \mathbf{P1} \otimes \mathbf{RULE1} + \mathbf{P2} \otimes \mathbf{RULE2}$$

where  $\mathbf{P1}$  and  $\mathbf{P2}$  are the position vectors. Importantly, since the position vectors are also semantic pointers, the position vectors have some relation. That is to say  $\mathbf{P2} = \mathbf{P1} \otimes \mathbf{ADD1}$ , and likewise for subsequent position vectors.

Individual rules in the instruction are encoded as a superposition of the conditions that make up the antecedent and the actions that make up the consequence of the rule. For example, the rule “IF **STATEA** THEN **ACTIONB**” is encoded as

$$\mathbf{RULE} = \mathbf{ant}(\mathbf{STATEA}) + \mathbf{ACTIONB}$$

where  $\mathbf{ant}()$  is a randomly generated linear operator applied to the **STATEA** vector that serves to disambiguate the antecedent and consequent components of the rule.

State conditions are encoded by binding vectors that describe the state being monitored with the state value required for the rule to be executed. Thus, the state condition “*state* = **A**” is constructed as

$$\mathbf{STATEA} = \mathbf{STATE} \otimes \mathbf{A}.$$

Other conditions can also be combined in this state representation. For example, if the state conditions was “*vision* = **3** and *state* = **A**” (i.e. looking at a 3 while in state A), then the state representation would be

$$\mathbf{VIS3\&STATEA} = \mathbf{VISION} \otimes \mathbf{3} + \mathbf{STATE} \otimes \mathbf{A}.$$

Actions are encoded by combining the bound result of an “action” descriptor with the specific action to be performed with an optional bound result of a “data” descriptor with the specific data to be used with the action. A “write the number 2” action is thus represented as

$$\mathbf{WRITE2} = \mathbf{ACTION} \otimes \mathbf{WRITE} + \mathbf{DATA} \otimes \mathbf{2}.$$

Combining all of the representations above, the full encoding of an instruction can be demonstrated. As an example the instruction:

1. **IF** *vision* = 0, **THEN** *push* button A
2. **IF** *vision* = 1, **THEN** *push* button B

is encoded as

$$\begin{aligned} \text{INSTR} &= \mathbf{P1} \otimes [\text{ant}(\text{VISION} \otimes \mathbf{0}) + \\ &\quad \text{ACTION} \otimes \text{PUSH} + \text{DATA} \otimes \text{BTNA}] + \\ &\quad \mathbf{P2} \otimes [\text{ant}(\text{VISION} \otimes \mathbf{1}) + \\ &\quad \text{ACTION} \otimes \text{PUSH} + \text{DATA} \otimes \text{BTNB}] \quad (1) \end{aligned}$$

It is important to note that at the end of this computation, the instruction is encoded as a single vector with the same dimensionality as the original atomic components.

### Decoding Instructions

With the instruction encoding schema presented above, the instructions can be decoded in one of two ways: by using positional information, and by using the values of the states in the system.

**Sequential Decoding of Instructions** A rule associated with a specific position within the instruction can be retrieved by binding the instruction vector with the inverse of the position vector.

$$\begin{aligned} \text{rule} &= \text{INSTR} \otimes \mathbf{P1}^* \quad (2) \\ &= \mathbf{P1}^* \otimes \mathbf{P1} \otimes \text{RULE1} + \mathbf{P1}^* \otimes \mathbf{P2} \otimes \text{RULE2} \\ &= \mathbf{I} \otimes \text{RULE1} + \mathbf{P1}^* \otimes \mathbf{P2} \otimes \text{RULE2} \\ &\approx \text{RULE1} \end{aligned}$$

Given the rule vector, it is possible to retrieve information related to the consequent by binding it with the inverse of the “action” descriptor or the “state” descriptor.

$$\begin{aligned} \text{action} &= \text{rule} \otimes \text{ACTION}^* \quad (3) \\ &= [\text{ant}(\text{VISION} \otimes \mathbf{0}) + \text{ACTION} \otimes \text{PUSH} + \\ &\quad \text{DATA} \otimes \text{BTNA}] \otimes \text{ACTION}^* \\ &\approx \mathbf{I} \otimes \text{PUSH} = \text{PUSH} \end{aligned}$$

Likewise,

$$\begin{aligned} \text{data} &= \text{rule} \otimes \text{DATA}^* \quad (4) \\ &= [\text{ant}(\text{VISION} \otimes \mathbf{0}) + \text{ACTION} \otimes \text{PUSH} + \\ &\quad \text{DATA} \otimes \text{BTNA}] \otimes \text{DATA}^* \\ &\approx \mathbf{I} \otimes \text{BTNA} = \text{BTNA} \end{aligned}$$

After the rule has been executed, the next rule can be computed by incrementing the position vector ( $\mathbf{P2} = \mathbf{P1} \otimes \text{ADD1}$ ) and repeating Equations 2, 3 & 4 with this new position vector.

**Conditionally Responsive Decoding of Instructions** An instruction can also be decoded using the values of the state conditions. In order to do so, the value of the state condition(s) is bound to its associated state descriptor(s), and the inverse of this result is bound to the instruction vector to yield the position of the rule that best matches the state condition(s). Using Equation 1 as an example, if the vision state condition had a value of 1, the position of the rule that best matches this can be found like so:

$$\begin{aligned} \text{pos} &= \text{INSTR} \otimes (\text{ant}(\text{state}) \otimes \text{state\_val})^* \quad (5) \\ &= \text{INSTR} \otimes (\text{ant}(\text{VISION}) \otimes \mathbf{1})^* \\ &= \mathbf{P1} \otimes [\text{ant}(\text{VISION} \otimes \mathbf{0}) \dots] \otimes (\text{ant}(\text{VISION}) \otimes \mathbf{1})^* + \\ &\quad \mathbf{P2} \otimes [\text{ant}(\text{VISION} \otimes \mathbf{1}) \dots] \otimes (\text{ant}(\text{VISION}) \otimes \mathbf{1})^* \\ &\approx \mathbf{P2} \otimes [\mathbf{I} + \dots] \approx \mathbf{P2} \end{aligned}$$

Once the position vector has been retrieved, the sequential instruction decoding equations can then be used to obtain the action and data associated with the rule.

### The Model

With the ability to encode and decode general instructions, modifying the existing Spaun action selection module to take advantage of this is straightforward. It only entails the addition of an instruction processing module that implements the instruction decoding equations (Eq 2 – 5) above. The output of this module then become new state variables which the action selection system monitors when selecting an appropriate action (see Figure 2).

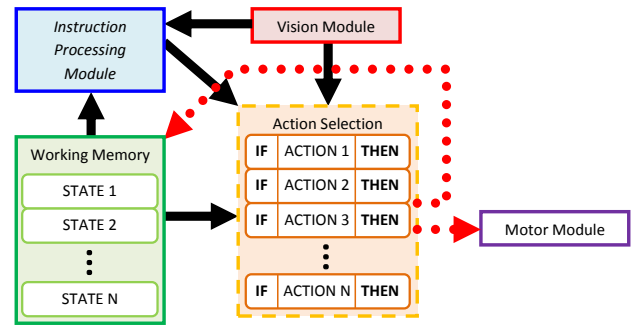


Figure 2: Proposed modification to Spaun’s action selection system with the addition of an instruction processing module (italicized). As in Figure 1, state monitoring is indicated with a solid arrow, and action effects with a dotted arrow.

Validation of the model comes in the form of behavioural analysis as well as matching the model dynamics to human timing data. The model is implemented with spiking neurons and biologically realistic synaptic time constants in order to generate realistic temporal dynamics.

### Neural Representation

Fundamental to the SPA is the vector-based representation of information. We use methods of the Neural Engineer-

ing Framework (NEF) to accomplish this in spiking neurons (Eliasmith & Anderson, 2003). Georgopoulos et al. (1986) demonstrated that motor neurons are well characterized as having responses driven by their preferred direction to movement in two dimensions. The NEF generalizes this notion to suggest that neurons can represent any number of dimensions, and the neuron's preferred direction determines its activity with regards to its input in a given vector space. Mathematically, the current  $J$  flowing into a neuron can be calculated using as

$$J(\mathbf{x}) = \alpha(\mathbf{e} \cdot \mathbf{x}) + J^{bias}, \quad (6)$$

where  $\alpha$  and  $J^{bias}$  are neuronal scaling terms,  $\mathbf{e}$  is the neuron's preferred direction (or encoding vector), and  $\mathbf{x}$  is the vector to be represented. The inner product computes the similarity between the encoding and input vector and determines how much current is being fed to the neuron. The leaky integrate-and-fire (LIF) neuron model equation is then used to convert this current into a firing rate.

$$a(\mathbf{x}) = G[J(\mathbf{x})] = \frac{1}{\tau^{ref} - \tau^{RC} \ln\left(1 - \frac{J^{th}}{J(\mathbf{x})}\right)} \quad (7)$$

In the equation above,  $\tau^{ref}$  is the neuron refractory time constant,  $\tau^{RC}$  is the neuron RC time constant, and  $J^{th}$  is the neuron threshold firing current. With a population of neurons, it is then possible to derive optimal decoding vectors that can be used to convert the neural activity back into the high dimensional vector space. Eliasmith and Anderson (2003) demonstrate how these decoders  $\mathbf{d}$  can be computed.

$$\mathbf{d} = \Gamma^{-1}\Upsilon, \text{ where} \quad \Gamma_{ij} = \int a_i(\mathbf{x})a_j(\mathbf{x}) d\mathbf{x} \quad \Upsilon_i = \int a_i(\mathbf{x})\mathbf{x} d\mathbf{x} \quad (8)$$

An estimate of the original vector  $\mathbf{x}$  can then be generated by multiplying each neuron's decoding vector with its activity.

$$\hat{\mathbf{x}} = \sum_i a_i(\mathbf{x})\mathbf{d}_i \quad (9)$$

The encoding and decoding vectors can also be used to determine the optimal connection weights between two neural populations.

$$w_{ij} = \alpha_j \mathbf{e}_j \mathbf{d}_i \quad (10)$$

Taking into account a specific function while solving for the decoding vectors yields the set of connection weights that will cause the neurons in the post-synaptic population to compute said function. For example,

$$\widehat{f(\mathbf{x})} = \sum_i a_i(\mathbf{x})\mathbf{d}_i^f, \quad (11)$$

where  $\mathbf{d}_i^f$  are the decoding vectors solved with the function  $f$  incorporated into Equation 8. In other words, these equations allow us to build a spiking neuron model that performs arbitrary specified computations. See Eliasmith and Anderson (2003) for additional details.

## Neural Implementation

The model proposed here relies on two key functions: the binding operation and working memory.

The binding operation is performed by a two step process. First the Fourier transform (FT) of both input vectors is computed, and these are multiplied element-wise. Performing an inverse Fourier transform (IFT) on this result provides the desired answer. That is,

$$\mathbf{A} \otimes \mathbf{B} = IFT(FT(\mathbf{A}) \odot FT(\mathbf{B})), \quad (12)$$

where  $\odot$  is the element-wise multiplication operation. The FT, IFT and element-wise multiplication are functions that can be computed by spiking neurons using the methods discussed in the previous section (see Equation 11).

The working memory component is identical to that used in Spaun. This component is implemented by a recurrent network that is able to stably store information over time. The storage and retrieval of information is determined by gates controlled by the basal ganglia.

## Response Timing

In this section we compare the behaviour of the model to two different tasks: the choice reaction time task, and a sub-vocal counting task. The choice reaction time task demonstrates the model's ability to perform unordered instructions, while the sub-vocal counting task demonstrates the model's ability to perform sequential instructions. Note that for both of these tasks, the architecture of the model remains the same, with the only difference being the instruction vector and visual stimuli that it is required to process.

### Conditionally Responsive Decoding - Two-Choice and Simple Reaction Time Task

To test the model's ability to account for human instruction processing time, it was tested with the two-choice (CRT) and simple reaction time (SRT) tasks described in Grice, Nullmeyer, & Spiker (1982). Since the input stimuli and motor action performed are similar in both of these tasks, any difference in reaction time can be attributed to the speed at which the different instructions are processed.

In the two-choice reaction time task, the subject is instructed to push one of two buttons, the identity of which is indicated by some sort of visual stimuli. To simulate this with the general instruction following model, it is given the instruction:

1. **IF vision = ZERO, THEN state = Push, motor = A**
2. **IF vision = ONE, THEN state = Push, motor = B**

Figure 3 demonstrates the model performing this instruction.

In the simple reaction time task, the subject is instructed to push a single button in response to a single stimulus. This task requires no instruction processing so the rule:

$$\mathbf{IF vision = TWO, THEN state = Push, motor = C}$$

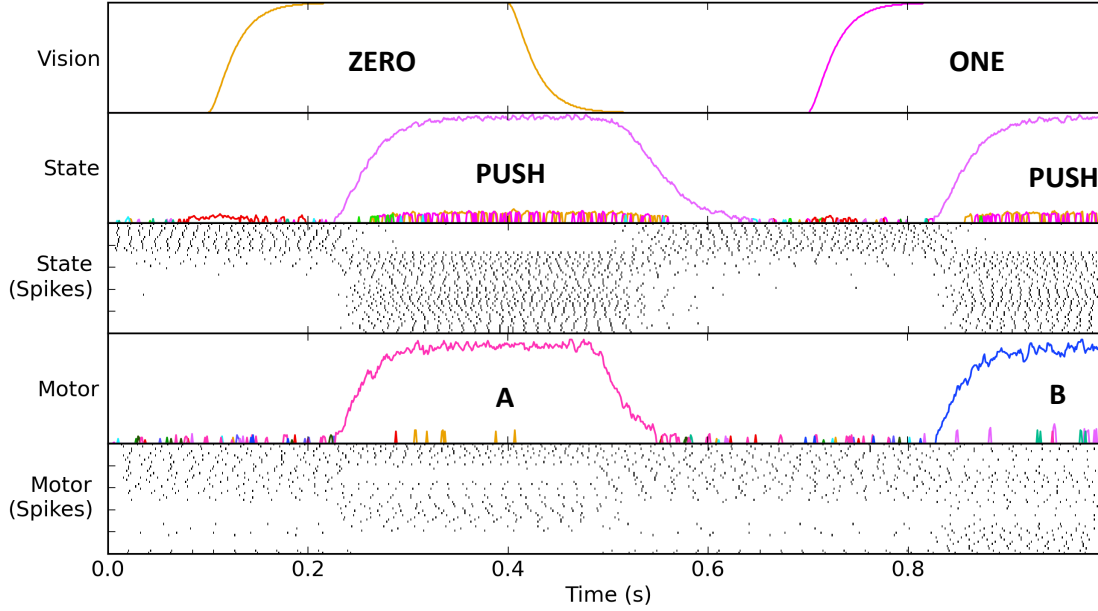


Figure 3: Neural response data for the two-choice reaction time task. Shown are the decoded representations for two neural populations (an internal state memory, and the motor output), and the visual stimulus provided. Also displayed is the spiking neural data associated with each of the neural populations. Note that only the cognitive components (i.e. no input stimuli processing lag nor motor lag) of the reaction time task are being simulated in this model.

is encoded directly in the basal ganglia. By doing so, the model is able to execute the desired action when presented with the appropriate stimuli without requiring any additional processing in the instruction processing module.

The model reports a reaction time difference of  $77 \pm 34$  ms between the two tasks, while Grice, Nullmeyer, & Spiker report a reaction time difference of  $81 \pm 72$  ms for human subjects.

### Sequential Decoding - Sub-vocal Counting

For this task, the model was given a sub-vocal counting instruction. This instruction is formatted as sequence of actions, and thus have no antecedent.

1. *memory* = Store, *data* =  $X$
2. *state* = Add1
3. *state* = Write, *motor* = *memory*

In the instruction above, the variable  $X$  is a vector representing a digit from 0 to 9. Instructions requiring more than one count (e.g. add 1 twice), have the second action repeated the appropriate number of times (and appropriately renumbered). Figure 4 illustrates the model performing the sub-vocal counting task for one count.

The mean reported count time per item is  $384 \pm 29$  ms which falls well between the reported human range of  $344 \pm 135$  ms (Landauer, 1962), and provides a much better match to the human data than Spaun's reported count time per item time of  $419 \pm 10$  ms (Eliasmith et al., 2012).

### Simulation Details

In total the model is made up of 318,870 spiking LIF neurons, and uses 256-dimensional semantic pointers. It should be noted that Spaun utilizes semantic pointers with 512 dimensions, and this was reduced for this model to decrease the amount of time required to simulate the experiments. It takes  $275 \pm 25$  seconds of CPU time to simulate 1 second of simulation time on a machine with a 3.40 GHz Core i7-3770 quad-core CPU and 16 GB of RAM.

### Discussion

The model presented in this paper demonstrates the ability to process and execute generic instructions without needing any changes to the underlying architecture. It is also able to reproduce response times in human reported ranges based purely on the temporal dynamics of the underlying neural implementation – without the need for data fitting of any kind.

Because the model utilizes semantic pointers to represent information, it is also highly scalable. The maximum number of concepts the model is able to represent is dependent on the dimensionality of the semantic pointer used, and not on the number of knowledge nodes present in the model. Crawford, Gingerich and Eliasmith (Crawford, Gingerich, & Eliasmith, 2013) demonstrate that the entirety of WordNet (117,659 concepts) can be represented using 512 dimensional semantic pointers. Increasing the proposed model to utilize 512 dimensional semantic pointers would add an additional 287,488 neurons to the model.

One major limitation to this model, however, is its inability to learn frequently executed instructions. In essence, even

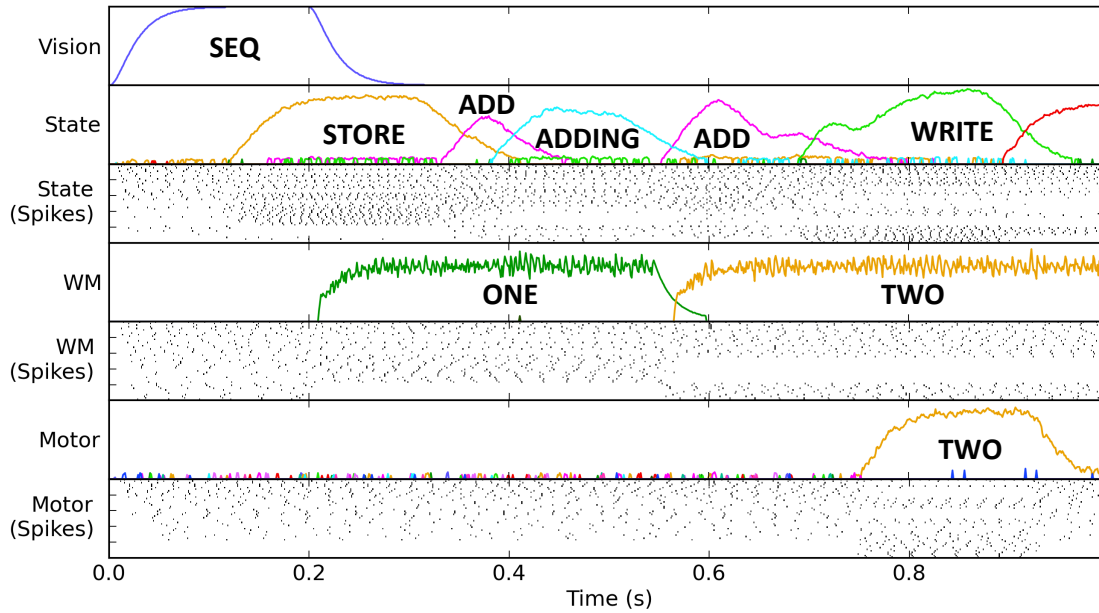


Figure 4: Neural response data for the sub-vocal counting task. Shown are the decoded representations for three neural populations (an internal state memory, working memory (WM), and the motor output), and the visual stimulus provided. Also displayed is the spiking neural data associated with each of the neural populations. Note that the **ADD** value for the state variable indicate both the start and end of the number addition action.

if it is presented with multiple instances of the same instruction, it is unable to form an expert action for that instruction. This issue is currently being investigated and integrating this ability in the proposed model remains as future work.

This paper also makes no mention of how the model could construct a new instruction vector given purely a visual stream of words or symbols. Concurrent work done by Stewart and Eliasmith (Stewart & Eliasmith, 2013) provides insight on how this issue can be made tractable.

## Acknowledgments

This research was supported by the Natural Sciences and Engineering Research Council of Canada, the Canadian Foundation for Innovation, and the Ontario Innovation Trust.

## References

- Anderson, J. R. (1996). Act: A simple theory of complex cognition. *American Psychologist*, 51, 355–365.
- Choo, X. (2010). *The ordinal serial encoding model: Serial memory in spiking neurons*. Unpublished master's thesis, The University of Waterloo.
- Crawford, E., Gingerich, M., & Eliasmith, C. (2013). Biologically plausible, human-scale knowledge representation. In *Proceedings of the 35th annual conference of the cognitive science society*.
- Eliasmith, C. (In Press). *How to build a brain: A neural architecture for biological cognition*. New York, NY: Oxford University Press.
- Eliasmith, C., & Anderson, C. H. (2003). *Neural engineering: computation, representation, and dynamics in neurobiological systems*. Cambridge, MA: The MIT Press.
- Eliasmith, C., Stewart, T. C., Choo, X., Bekolay, T., Dewolf, T., Tang, Y., et al. (2012). A large-scale model of the functioning brain. *Science*, 338(6111), 1202–1205.
- Georgopoulos, A. P., Schwartz, A., & Kettner, R. E. (1986). Neuronal population coding of movement direction. *Science*, 233, 1416–1419.
- Grice, G. R., Nullmeyer, R., & Spiker, V. A. (1982). Human reaction time: Toward a general theory. *Journal of Experimental Psychology: General*, 111(1), 135–153.
- Landauer, T. K. (1962). Rate of implicit speech. *Perceptual and Motor Skills*, 15, 646.
- Plate, T. A. (2003). *Holographic reduced representations: distributed representations for cognitive structures*. Stanford, CA: CSLI Publications.
- Stewart, T. C., Bekolay, T., & Eliasmith, C. (2012). Learning to select actions with spiking neurons in the basal ganglia. *Frontiers in Decision Neuroscience*, 6(2).
- Stewart, T. C., & Eliasmith, C. (2013). Parsing sequentially presented commands in a large-scale biologically realistic brain model. In *Proceedings of the 35th annual conference of the cognitive science society*.
- Taatgen, N. A. (1999). A model of learning task-specific knowledge for a new task. In *Proceedings of the twenty-first annual conference of the cognitive science society* (pp. 730–735). Mahwah, NJ: Erlbaum.
- Taatgen, N. A., & Lee, F. J. (2003). Production composition: A simple mechanism to model complex skill acquisition. *Human Factors*, 45(1), 61–76.