A Spiking Neuron Model of Serial-Order Recall

Feng-Xuan Choo (fchoo@uwaterloo.ca)
Chris Eliasmith (celiasmith@uwaterloo.ca)
Center for Theoretical Neuroscience, University of Waterloo
Waterloo, ON, Canada N2L 3G1

Abstract

Vector symbolic architectures (VSAs) have been used to model the human serial-order memory system for decades. Despite their success, however, none of these models have yet been shown to work in a spiking neuron network. In an effort to take the first step, we present a proof-of-concept VSA-based model of serial-order memory implemented in a network of spiking neurons and demonstrate its ability to successfully encode and decode item sequences. This model also provides some insight into the differences between the cognitive processes of memory encoding and subsequent recall, and establish a firm foundation on which more complex VSA-based models of memory can be developed.

Keywords: Serial-order memory; serial-order recall; vector symbolic architectures; holographic reduced representation; population coding; LIF neurons; neural engineering framework

Introduction

The human memory system is able to perform a multitude of tasks, one of which is the ability to remember and recall sequences of serially ordered items. In human serial recall experiments, subjects are presented items at a fixed interval, typically in the range of two items per second up to one item every 4 seconds. After the entire sequence has been presented the subjects are then asked to recall the items presented to them, either in order (serial recall), or in any order the subject desires (free recall). Plotting the recall accuracy of the subjects, experimenters often obtain a graph with a distinctive U-shape. This unique shape arises from what is known as the primacy and recency effects. The primacy effect refers to the increase in recall accuracy the closer the item is to the start of the sequence, and the recency effect refers to the same increase in recall accuracy as the item gets closer to the end of the sequence.

Many models have been proposed to explain this peculiar behaviour in the recall accuracy data. Here we will concentrate on one class of models which employ vector symbolic architectures (VSAs) to perform the serial memory and recall. Using VSAs to perform serial memory tasks would be insufficient however, if the VSA-based model cannot be implemented in spiking neurons, and thus, cannot be used to explain what the brain is actually doing. In this paper, we thus present a proof-of-concept VSA-based model of serial recall implemented using spiking neurons.

Vector Symbolic Architecture

There are four core features of vector symbolic architectures. First, information is represented by randomly chosen vectors that are combined in a symbol-like manner. Second, a superposition operation (here denoted with a $+$) is used to combine vectors such that the result is another vector that is similar to the original input vectors. Third, a binding operation ($\otimes$) is used to combine vectors such that the result is a vector that is dissimilar to original vectors. Last, an approximate inverse operation (denoted with *, such that A^* is the approximate inverse of A) is needed so that previously bound vectors can be unbound.

$$A \otimes B \otimes B^* \approx A \tag{1}$$

Just like addition and multiplication, the VSA operations are associative, commutative, and distributive.

The class of VSA used in this model is the Holographic Reduced Representation (HRR) (Plate, 2003). In this representation, each element of an HRR vector is chosen from a normal distribution with a mean of 0, and a variance of $1/n$ where n is the number of elements there are in the vector. The standard addition operator is used to perform the superposition operation, and the circular convolution operation is used to perform the binding operation. The circular convolution of two vectors can be efficiently computed by utilizing the Fast Fourier Transform (FFT) algorithm:

$$\mathbf{x} \otimes \mathbf{y} = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{x}) \odot \mathcal{F}(\mathbf{y})), \tag{2}$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ are the FFT and inverse FFT operations respectively, and $\odot$ is the element-wise multiplication of the two vectors. The circular convolution operation, unlike the standard convolution operation, does not change the dimensionality of the result vector. This makes the HRR extremely suitable for a neural implementation because it means that the dimensionality of the network remains constant regardless of the number of operations performed.

The VSA-based Approach to Serial Memory

There are multiple ways in which VSAs can be used to encode serially ordered items into a memory trace. The CADAM model (Liepa, 1977) provides a simple example of how a sequence of items can be encoded as a single memory trace. In the CADAM model, the sequence containing the items **A**, **B**, and **C** would be encoded as in single memory trace, M_{ABC} as follows:

$$\begin{aligned} M_A &= \mathbf{A} \\ M_{AB} &= \mathbf{A} + \mathbf{A} \otimes \mathbf{B} \\ M_{ABC} &= \mathbf{A} + \mathbf{A} \otimes \mathbf{B} + \mathbf{A} \otimes \mathbf{B} \otimes \mathbf{C} \end{aligned}$$

The model presented in this paper, however, takes inspiration from behavioural data obtained from macaque monkeys. This data suggests that each sequence item is encoded using

ordinal information (Orlov, Yakovlev, Hochstein, & Zohary, 2000), rather than being "chained" together as in the CADAM model. To achieve this, additional vectors are used to represent the ordinal information of each item. In the subsequent equations, this ordinal vector is represented as P_i, where i indicates the item's ordinal number in each sequence. The memory trace M_{ABC} would thus be computed like so:

$$M_A = P_1 \otimes \mathbf{A} \tag{3}$$

$$M_{AB} = P_1 \otimes \mathbf{A} + P_2 \otimes \mathbf{B} \tag{4}$$

$$M_{ABC} = P_1 \otimes \mathbf{A} + P_2 \otimes \mathbf{B} + P_3 \otimes \mathbf{B} \tag{5}$$

The encoding strategy presented above does not seem to have any mechanism by which to explain the primacy or recency effects seen in human behavioural data. In order to achieve these effects, additional components are added to the model. These components are discussed further below.

Neural Representation

To implement any of these models, we need to determine how a vector can be represented by a population of spiking neurons. In 1986, Georgopoulos et al. demonstrated that in the brain, 2D movement directions are encoded by a large population of neurons, with each neuron being most active for one specific direction – their preferred direction. The activity of each neuron would then indicate the similarity of the input vector to each neuron's preferred direction vector. Since the movement direction is essentially a two-dimensional vector, this method of vector representation can be extended to multiple dimensions as well. For a population of neurons, the current J flowing into neuron i can then be calculated by the following equation.

$$J_i(\boldsymbol{x}) = \alpha_i(\tilde{\phi}_i \cdot \boldsymbol{x}_i) + J_i^{bias} \tag{6}$$

In the above equation, the dot product computes the similarity between the input vector $\boldsymbol{x}$ and the neuron's preferred direction vector $\tilde{\phi}$. The neuron gain is denoted by α, while J^{bias} denotes a fixed background input current. The current J_i can then be used as the input to any neuron model $G[\cdot]$ to obtain the activity for neuron i. In this model, we use the leaky integrate-and-fire (LIF) neuron model, characterized as such:

$$a_i(\boldsymbol{x}) = G_i[J_i(\boldsymbol{x})] = \frac{1}{\tau^{ref} - \tau^{RC} \ln\left(1 - \frac{J^{th}}{J_i(\boldsymbol{x})}\right)}, \tag{7}$$

where $a_i(\boldsymbol{x})$ is the average firing rate of the neuron i, τ^{ref} is the neuron refractory time constant, τ^{RC} is the neuron RC time constant, and J^{th} is the neuron threshold firing current. For a time-varying input $\boldsymbol{x}(t)$, the equations remain the same, with the exception that the activity of the neuron is no longer an average firing rate, but rather a spike train:

$$a(\boldsymbol{x}(t)) = \sum_n \delta(t - t_n) \tag{8}$$

Since the spike train represents the neuron's response to the input vector $\boldsymbol{x}$, given the spike trains from all the neurons in the population, it should be possible to derive decoding vectors ϕ that can be used to estimate the original input. Eliasmith and Anderson (2003) demonstrate that these decoding vectors can be found using the following equation.

$$\begin{aligned} &\phi = \Gamma^{-1}\Upsilon, \text{ where} \\ &\Gamma_{ij} = \int a_i(x) a_j(x)\, \mathrm{d}x \qquad \Upsilon_i = \int a_i(x) x\, \mathrm{d}x \end{aligned} \tag{9}$$

By weighting the decoding vectors with the post-synaptic current $h(t)$ generated by each spike, it is then possible to construct $\hat{x}(t)$, an estimate of the input vector. Equation (10) demonstrates how this is achieved. The parameters used to generate the shape of $h(t)$ is determined by the neurophysiology of the neuron population.

$$\begin{aligned} \hat{x}(t) &= \sum_{i,n} \delta(t - t_{in}) * h(t) \phi_i \\ &= \sum_{i,n} h(t - t_{in}) \phi_i \end{aligned} \tag{10}$$

The encoding and decoding vectors also provides a method by which the optimal connection weights between two neural populations can be. If for example, the transformation between two populations of neurons is a simple scaling operation, where the output of the second group of neurons should be $C\boldsymbol{x}$, then the connection weights w between the populations should be

$$w_{ij} = C \alpha_j \tilde{\phi}_j \phi_i \tag{11}$$

Extending Equation (7) for linear operations is also straightforward. Consider three neural populations: one to represent the input x, another to represent the input y, and a third that we wish to have compute the linear combination $Cx + Dy$. The activity of the neurons in final population can be determined by

$$c_k(Cx + Dy) = G_k \left[\sum_i w_{ki} a_i(x) + \sum_j w_{kj} b_j(y) + J_k^{bias} \right], \tag{12}$$

where a_i, b_j, and c_k are the activities of the neurons in the first, second and third neural populations respectively. Employing Equation (11), the synaptic connection weights can also be determined. Letting w_{ki} be the connection weights between the first and third population, and w_{kj} be the connection weights between the second and third population, they work out to be:

$$w_{ki} = \alpha_k \tilde{\phi}_k \phi_i^x \qquad \text{and} \qquad w_{kj} = \alpha_k \tilde{\phi}_k \phi_j^y \tag{13}$$

Note that in the equation above, the superscripts serves to disambiguate the decoders, where ϕ^x signifies the decoders that represent x, and likewise for ϕ^y. Eliasmith and Anderson (2003) go into greater detail on how to use this general framework, known as the Neural Engineering Framework, to derive the appropriate decoders and connection weights to perform arbitrary non-linear operations as well.

The Neural Model

The neural model implemented in this paper is divided into two neural processes. One encodes an item sequence into a single memory trace, and the other decodes an encoded memory trace to retrieve its constituent items.

Sequence Encoder

Analysis of Equations (3) to (5) show that the memory trace for an arbitrary sequence of items can be constructed by computing the convolution of the last item vector with its ordinal vector, and then adding the result of the convolution to the memory trace of the sequence less the final item. From this, a generic sequence encoding equation can be derived (from here on referred to as the *basic encoding equation*).

$$M_i = M_{i-1} + P_i \otimes I_i \tag{14}$$

In the equation above, M_i represents the memory trace after encoding the i^{th} item. P_i and I_i represents the i_{th} item's ordinal vector and item vector respectively.

As mentioned previously, the encoding equation in its basic form does not account for the primacy and recency effects seen in human behavioural data. To achieve the primacy effect, rehearsal is simulated by adding an additional weighted copy of the old memory trace to the memory trace being calculated for the current item. In essence, as each item is rehearsed, a weighted copy of the item is added to the memory trace to "boost" the item's representation within the memory trace. In the equation below, the memory trace of the rehearsal-based encoding is denoted by R_i and the weight applied to the rehearsed contribution of the old memory trace is denoted by α. In the model implemented for this paper, α was set to 0.3.

$$R_i = R_{i-1} + P_i \otimes I_i + \alpha R_{i-1} \tag{15}$$

$$= (1+\alpha)R_{i-1} + P_i \otimes I_i \tag{16}$$

To achieve the recency effect, an separate memory component is added to play the role of a sensory input buffer. The input buffer encodes items in a similar fashion to Equation (14) with a decay added to the old memory trace. This decay causes the input buffer to store only the most recently presented items, thus mimicking the basic recall characteristics of the human working memory system. In the neural implementation of this model, the decay is achieved by tuning the integrators used in the memory modules to slowly drift to zero if no additional input is applied to them. Equation (17) illustrates how this decay can be represented mathematically, with the memory trace of the input buffer represented by B_i, and the rate of decay represented by β.

$$B_i = \beta B_{i-1} + P_i \otimes I_i \tag{17}$$

The final memory trace of the encoded item sequence is then computed by combining the memory trace from the rehearsal component and the memory trace from the input buffer component.

$$M_i = R_i + B_i \tag{18}$$

From the above encoding equations, several issues become evident. First, two operations need to be implemented – a circular convolution and an addition operation. Second, a memory module is needed to hold the value of M_{i-1} while the new memory trace M_i is computed. With these components, and the rehearsal and decay mechanisms described above, a high level block diagram of the complete encoding network can be constructed, as shown in Figure 1.

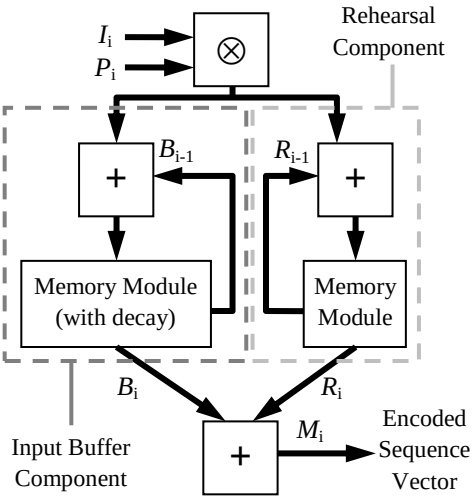

Figure 1: Encoding network functional block diagram.

Sequence Decoder

The decoding process is much simpler than the encoding process. The first step of the decoding process is to convolve the encoded memory trace with the inverse of the desired ordinal vector. For example, if the system is trying to decode the second item in the sequence, the encoded memory trace would be convolved with the inverse of P_2. Next, the result of this convolution is fed to a cleanup memory module. The cleanup memory module contains a copy of all the item vectors in the original sequence, and when given an input, will determine which of the original item vectors best matches the input vector. An example of this decoding process follows. To simplify the example, only the basic encoding equation is used.

$$\begin{aligned} M_{ABC} &= P_1 \otimes \mathbf{A} + P_2 \otimes \mathbf{B} + P_3 \otimes \mathbf{B} \\ C_B &= M_{ABC} \otimes P_2^* \\ &= P_1 \otimes \mathbf{A} \otimes P_2^* + P_2 \otimes \mathbf{B} \otimes P_2^* + P_3 \otimes \mathbf{B} \otimes P_2^* \\ &\approx P_1 \otimes \mathbf{A} \otimes P_2^* + B + P_3 \otimes \mathbf{B} \otimes P_2^* \\ I_B &= cleanup(C_B) \approx B \end{aligned}$$

From the example above, we see that convolving the memory trace M_{ABC} with the inverse of P_2 results in a vector with the desired item vector B combined with the unwanted vectors $(P_1 \otimes \mathbf{A} \otimes P_2^*)$ and $(P_3 \otimes \mathbf{B} \otimes P_2^*)$. However, since the cleanup memory module only contains the item vectors from

the original sequence and not the superfluous vectors, feeding the result of the convolution through the cleanup memory isolates the item vector B, producing the desired result. Figure 2 illustrates the high level block diagram used to implement the decoding network.

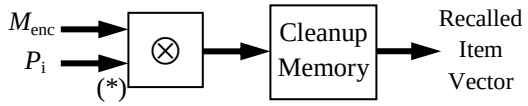

Figure 2: Decoding network functional block diagram.

Performing the Binding Operation Referring back to Equation (2) we see that the binding operation can be calculated using the FFT and IFFT algorithms, so the first step to implementing the binding operation in neurons is to implement these two operations. The equations that compute the FFT and IFFT algorithms are as follows:

$$\begin{aligned} \text{FFT}: &\qquad X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0,...,N-1 \\ \text{IFFT}: &\qquad x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N} kn} \qquad n = 0,...,N-1 \end{aligned} \tag{19}$$

Taking a closer look at the equations above, we see that they can be implemented efficiently as a multiplication between the input vector and a matrix containing the FFT (or IFFT) coefficients. From Equation (11), we can then set the synaptic connection weight matrix as the Fourier transform coefficients to calculate the required FFT and IFFT operations. The one caveat to this approach is that the real and imaginary components of the Fourier transform have to calculated separately and then recombined (with the appropriate sign changes) when the final result is calculated.

With the neural implementation of the Fourier transforms solved, the implementation of the circular convolution binding operation becomes trivial since the only other operation needed is an element-wise multiplication. This can be achieved by utilizing multiple neural populations, each handling one element in the element-wise multiplication.

The Memory Module Since the circular convolution and addition operations are essentially feed-forward neural networks, the memory module in this model needs to be able to drive the network with a constant value and store the new value at the same time. This is achieved by the use of gated integrators. When the integrator is not being gated, it attempts to match the value of the input signal. When the integrator is gated, it no longer responds to the input value, and outputs the previously stored value. By placing two gated integrators in parallel controlled by complementary gating signals, the memory module is able to simultaneously store the new input value while outputting the previously stored value.

Cleanup Memory The cleanup memory network used in this model is an extension of the cleanup memory presented in (Stewart, Tang, & Eliasmith, 2009). In essence, the implementation of cleanup memory involves creating multiple neural populations, each assigned to one item vector from the original item sequence. The preferred direction vectors $\tilde{\phi}$ for each neuron in one population is predefined to match the item vector it is meant to clean up. From Equation (6), we see that the the similarity (dot product) is calculated to determine the activity of the neuron. By predefining $\tilde{\phi}$, we can then determine the similarity of the decoded item vector to each of the original item vectors, thus determining which of the original item vectors best matches the decoded item vector.

Combining the Encoder and Decoder

Getting the spiking neuron model to encode a sequence, and subsequently decode the memory trace is achieved by chaining the encoder and the decoder together. Control signals are used to ensure that the decoding network only commences after the encoder has finished encoding the last item vector. Figure 3 shows the results of the complete network encoding and decoding a example twenty-dimensional 4-itemed sequence.

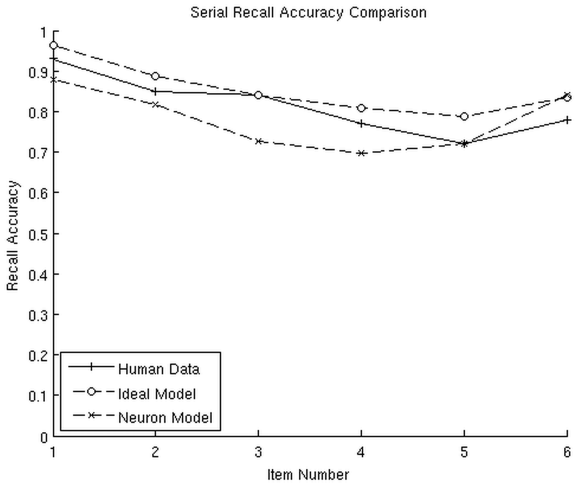

Figure 4: Plot of the recall accuracy data comparing results from human behavioural studies (from Henson et al. (1996), Figure 1), an ideal model implemented in Matlab®, and the spiking neuron model.

Results

The results of the simulation of the spiking neuron implementation of the ordinal serial encoding process is displayed in Figure 4. From the graph it can be seen that both the ideal Matlab®-implemented model and the spiking neuron model are a good match to the human data. The slightly reduced primacy in the neuronal implementation suggests that the simplistic implementation of the rehearsal mechanism can be improved. Figure 5 compares the transposition gradients – which is the count of the recall occurrences of each item for

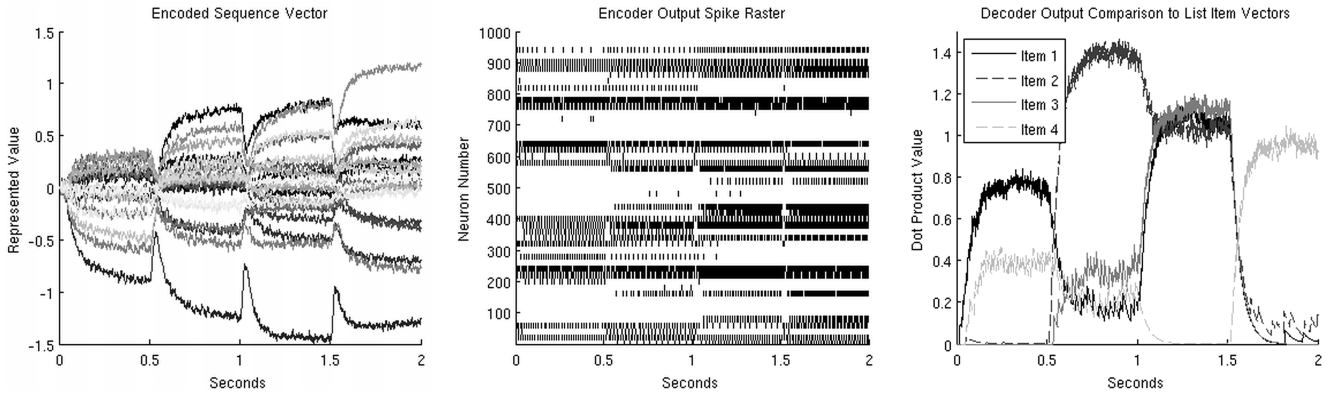

Figure 3: Simulation results from the spiking neuron implementation of the sequence encoder network. A 4-itemed sequence of 20-dimensional item vectors was presented to the network at a half-second interval (two items per second).
(Left) The output of the encoder, M_i, showing the encoded memory trace for each item vector presented. Referring to Equation (18) , the graph at $t = 0.5$ seconds shows $M_1 = R_1 + B_1$, the graph at $t = 1$ second shows $M_2 = R_2 + B_2$, and so forth for. The final encoded memory trace for the entire sequence is the output of the encoder network at $t = 2$ seconds.
(Center) The spike raster plot of the neurons in the output neuron population of the encoder network as it is encoding the sequence in the top figure. The spike raster is displayed for every 20^{th} neuron.
(Right) The similarity plot of each extracted item vector to each one of the four original item vectors. The similarity value between the vectors is obtained using the dot product operation. The graph shows the network correctly identifying the first, second, and last item. The third item is incorrectly identified because the similarity measures of the first three items are too close together for the system to accurately distinguish the correct answer.

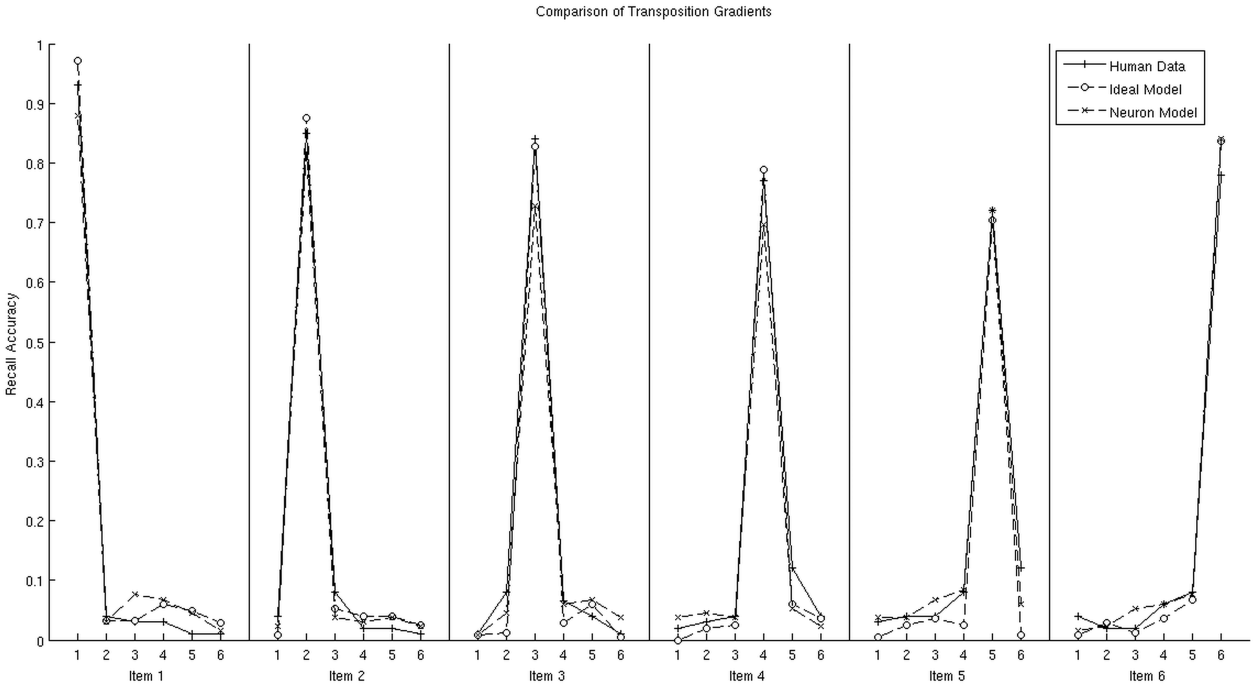

Figure 5: Plot of the transposition gradients comparing results from human behavioural studies (from Henson et al. (1996), Figure 2, for non-confusable items), an ideal model implemented in Matlab®, and the spiking neuron model. Comparing the plots, both the ideal model and the spiking neuron model are able to replicate the transposition curves in the human data.

each serial position – also reveals that both the ideal implementation and the spiking neuron implementation are able to reproduce the transposition effects seen in humans. Both of these simulations were run using six-itemed sequences consisting of fifty-dimensional HRR vectors, and were run for an average of 200 trials each.

Discussion

From the results it can be seen that both the ideal implementation and the spiking neuron model demonstrate the ability to reproduce the primacy, recency, and transposition effects seen in human data. Furthermore, unlike other models which entail a host of tunable parameters to fit the human data, this model only utilizes two tunable parameters: the amount of contribution to the memory trace in the rehearsal component, and the decay rate of the input buffer component.

The model presented here also provides some insight into the neurophysiological requirements of serial memory. It demonstrates the need for a working memory system capable of simultaneous storage and retrieval. This model also maps on very well to Baddeley's model of working memory (Baddeley, 2007), with the input buffer component acting as the phonological loop, and the rehearsal component functioning as the episodic buffer.

Despite their complexity, there are advantages of creating a spiking neuron model in comparison to theoretical models, or models implemented using rate neurons. It provides the ability to compare the spike data of the model to data collected from neural recordings. For example, data collect in Warden and Miller's 2007 paper shows that the neurons change their preferred items as more items are introduced into the system. Although the analysis has yet to be completed at the time this paper was written, it can be inferred that because the encoded sequence vector changes as more items are added, a neuron that is responsive to one configuration of the sequence vector would either be less responsive or not responsive at all when a new item is added – changing the configuration of the encoded sequence vector – as it does in this model.

Several studies (e.g. Chein & Fiez, 2001) have also identified brain areas that are active during serial memory tasks. Moreover, the studies have demonstrated that there are similarities and more importantly, differences, between the areas of activity during the encoding phase and recall phase. By assigning different components of the model to different brain areas (for example, the input buffer component to the temporal lobe, near the auditory cortex, and the rehearsal component to the lateral prefrontal cortex), it would be possible to determine if the pattern of activities recorded in these studies matches the pattern of activities produced by this model.

Future Work

As mentioned in the results section, the performance of the rehearsal component needs to be improved slightly. Possible ways of doing this is by having an active rehearsal mechanism which decodes and then re-encodes the stored memory trace within the inter-item interval (time between each item presentation). Such a rehearsal mechanism will also enable the model to be compared with serial recall studies involving list sizes that exceed the typical human memory span of 4 to 7 items.

Additionally, the current implementation of cleanup memory has a fixed vocabulary of item vectors that is predefined when the network is created. This means that the items in cleanup memory are static and do not change over time. It seems inconceivable that this is what occurs in the brain. Rather, future cleanup memory implementations should be dynamic, with the ability to "load" and "unload" arbitrary item vectors into its vocabulary.

References

Baddeley, A. (2007). *Working memory, thought, and action*. Oxford: Oxford University Press.

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

Georgopoulos, A. P., Schwartz, A., & Kettner, R. E. (1986). Neuronal population coding of movement direction. *Science*, *233*, 1416–1419.

Henson, R. N., Norris, D. G., Page, M. P., & Baddeley, A. D. (1996). Unchained memory: error patterns rule out chaining models of immediate serial recall. *The Quarterly Journal of Experimental Psychology*, *49A*(1), 80–115.

Liepa, P. (1977). *Models of content addressable distributed associative memory.* (Unpublished manuscript)

Orlov, T., Yakovlev, V., Hochstein, S., & Zohary, E. (2000, March). Macaque monkeys categorize images by their ordinal number. *Nature*, *404*.

Plate, T. A. (2003). *Holographic reduced representations: distributed representations for cognitive structures*. Stanford, CA: CSLI Publications.

Stewart, T. C., Tang, Y., & Eliasmith, C. (2009). A biologically realistic cleanup memory: autoassociation in spiking neurons. *9th International Conference on Cognitive Modelling*.

Warden, M. R., & Miller, E. K. (2007). The representation of multiple objects in prefrontal neuronal delay activity. *Cerebral Cortex*, *17*, 141–150.