The Ordinal Serial Encoding Model: Serial Memory in Spiking Neurons

by

Feng-Xuan Choo

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2010

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In a world dominated by temporal order, memory capable of processing, encoding and subsequently recalling ordered information is very important. Over the decades this memory, known as serial memory, has been extensively studied, and its effects are well known. Many models have also been developed, and while these models are able to reproduce the behavioural effects observed in human recall studies, they are not always implementable in a biologically plausible manner. This thesis presents the Ordinal Serial Encoding model, a model inspired by biology and designed with a broader view of general cognitive architectures in mind. This model has the advantage of simplicity, and we show how neuro-plausibility can be achieved by employing the principles of the Neural Engineering Framework in the model's design. Additionally, we demonstrate that not only is the model able to closely mirror human performance in various recall tasks, but the behaviour of the model is itself a consequence of the underlying neural architecture.

Acknowledgements

I would like to thank my parents and my twin sister for the never-ending support they have provided me throughout my studies. I would also like to thank my supervisor, Chris, for being the best supervisor anyone could have hoped for. Lastly, I would like to acknowledge my colleagues at the Centre for Theoretical Neuroscience, especially Terrance Stewart, whose collective knowledge and wisdom have proven to be an invaluable asset to my research.

Dedication

This is dedicated to all of the human subjects who sacrificed their free time, and to all of the laboratory animals who gave their lives, in order to provide the data used in this thesis.

Contents

List of Figures

Chapter 1

Introduction

In a world defined by time, the presentation order of information is often important to animals. Tasks such as route memory and planning, foraging, and tool use are just a few examples in which serially-ordered information is important. For example, not knowing the order in which landmarks along a predefined route are supposed to appear will lead to the animals getting lost. For humans, serial information plays an even more important role in our lives. Not only is it necessary for daily activity – the process of opening a locked door, for example – it is also necessary for abstract concepts such as language and mathematics. How is the brain able to process, encode and subsequently recall such information; or in other words, how does serial memory work?

Serial memory has been extensively studied as part of on-going research into the memory systems of humans and animals. As a consequence of this body of research, many models of serial memory have already been developed. Some models are implemented in a purely mathematical framework (e.g. the Start-End Model [Henson, 1998], Primacy Model [Page and Norris, 1998]), while others are implemented with production systems (e.g. ACT-R [Anderson and Matessa, 1997]). Others still use a learning-based neural network to model serial memory (e.g. recurrent neural networks [Botvinick and Plaut, 2006]).

All of the methods defined so far, however, have some disadvantages. Purely mathematical approaches tend to be highly abstract and ignore the constraints inherent in biological systems – such as the number of neurons needed to implement such a system, which would put a constraint on the overall complexity of the model. Production systems, while they are able to take into account the biological constraints, are very procedural – much like a computer system – and it is unclear how such a system can be implemented within the brain. Neural network based models are an improvement over the latter two, being both constrained by the neurophysiology and easily mapped on to the networks of neurons within the brain. However, learning rules are often used to train the network configuration of the model to produce the desired behaviour, and while this demonstrates that a certain configuration of the network is able to reproduce serial recall data, it does not further our understanding on what mechanisms within the network are

responsible for reproducing the data. The primary objective of our research is to bridge the gap between theoretical constructs and neurobiology, which has culminated in the Ordinal Serial Encoding model of serial memory – a mathematically-based model of serial memory implemented in a neurobiologically plausible manner – and the topic of this thesis.

1.1 Motivation

Two major behavioural effects are often observed when analyzing recall performance data. The first is known as the primacy effect, which is generally seen in most serial recall tasks. The primacy effect describes the tendency for items near the start of a sequence to be more accurately recalled than other items in the sequence. The second effects is the recency effect, which describes the opposite of the primacy effect. Essentially, it is the tendency for items near the end of the sequence to have a higher recall probability than items in the middle of the sequence. Interestingly, the recency effect is only observed in some recall tasks, such as the backwards recall, and free recall tasks, and is often diminished when delayed recall is involved. This effect is not only seen in human recall data, but in animal recall data as well – for example, in the rhesus macaques for the task of recalling serial movements [Averbeck et al., 2002] and in rats in the task of recalling locations in a maze [Kesner and Novak, 1982].

Figure 1.1 compares the primacy and recency effects for the task of serial recall and free recall. In the serial recall task, subjects are presented a sequence of items, and then instructed to recall the items in the order in which they were presented. In free recall, the item presentation is the same, but on recall, subjects are allowed to recall items in any order they choose. As evidenced by the figure, these two seemingly similar tasks produce very different behaviour effects. In (forward) serial recall, there is a strong primacy effect, and a weaker recency effect; whereas in free recall, the primacy effect is relatively similar, but the recency effect is much more pronounced. Adding a delay period between the sequence presentation and item recall causes yet another effect to occur. In this scenario the recency effect almost entirely disappears, leaving only the primacy effect behind. Identifying the mechanisms in serial memory that are responsible for this behaviour and developing a model that is able to capture this behaviour while maintaining biological plausibility is the primary motivation of the research behind this thesis.

1.2 Thesis Format

This thesis is divided into seven chapters. Chapter 2 discusses some of the existing models of serial memory that have been implemented using a vector-based symbolic architecture. Chapter 3 provides a description of the holographic reduced representation, which is the architecture chosen for the implementation of the Ordinal Serial Encoding (OSE) model. This chapter also introduces the encoding and decoding schemes used in the OSE model. Chapter 4 details the

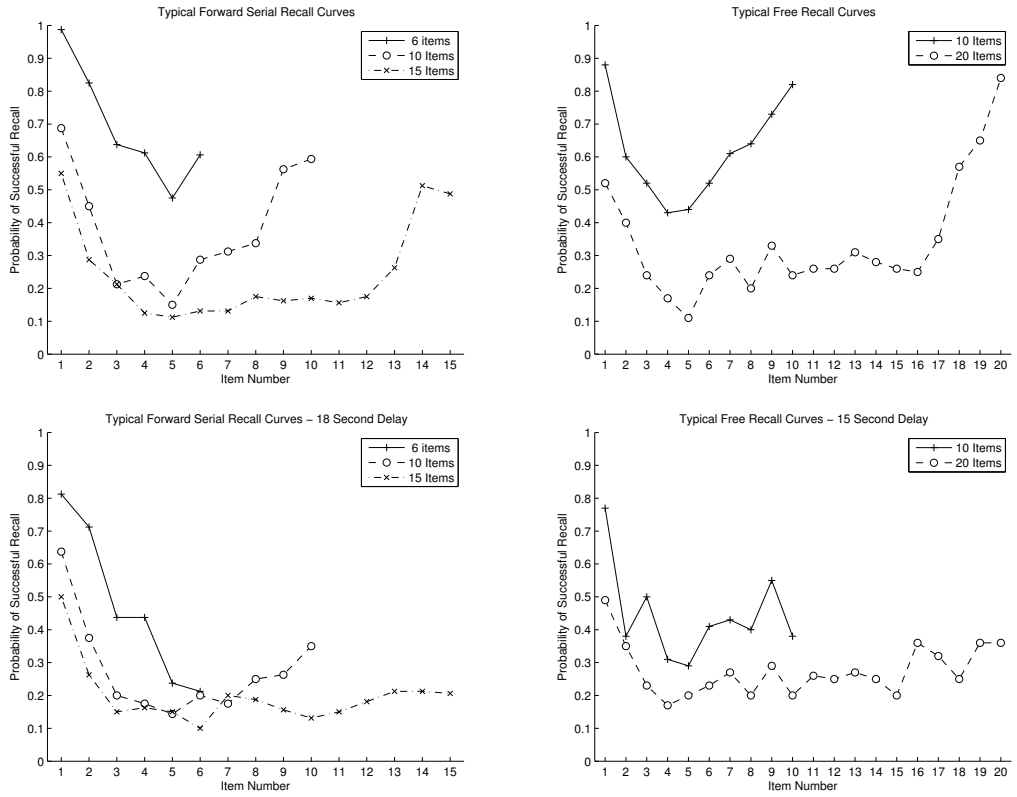

Figure 1.1: Comparison of the behavioural effects observed in different recall tasks. (Top Left) Typical recall performance curves for the forward serial recall task, exhibiting a strong primacy and a slightly weaker recency effect. (Top Right) Typical recall performance curves for the free recall task, demonstrating a more pronounced recency effect and an equivalent or weaker primacy effect. (Bottom Left) Recall curves after introducing an 18 second delay period for the forward recall task. (Bottom Right) Recall curves after introducing a 15 second delay period for the free recall task. After the introduction of the delay period, the recency effect disappears for both recall tasks. The serial recall task data is taken from [Jahnke, 1968], and the free recall data is taken from [Postman and Phillips, 1965].

neural engineering framework (NEF), and how it is used to design a neural network capable of performing the encoding and decoding schemes laid out in the previous chapter. Analysis of the performance of the OSE model is provided in Chapter 5, and several extensions to the OSE model are presented in Chapter 6. Lastly, Chapter 7 concludes the thesis with a discussion on future work that can be undertaken with the OSE model.

Chapter 2

Existing Vector-Based Models of Memory for Serial Order

Between 1977 and 1993, three influential models of serial recall were developed: the CADAM model [Liepa, 1977], the TODAM model [Murdock, 1983], and the TODAM2 model [Murdock, 1993]. Each of these models shared a similar architecture. The models encoded items in terms of vectors, and defined various mathematical operations that are used to manipulate these vectors algebraically. Using the convolution operation, vectors are associated (or bound) together to form new vectors on which other operations can then be performed. Vectors can also be "unbound" from each other using the correlation operation. The process of binding and unbinding vectors can be thought of as packing objects into a bag. Each object represents a vector, and the act of packing represents the binding operation. Additional operations can then be performed on this bag, for example, it can be moved to a new location, or perhaps put into a bigger bag, all the while maintaining the integrity of the objects inside. After these operations have been performed, the bag can then be unpacked – analogous to the unbinding operation – to retrieve the objects that were contained within the bag.

Because the convolution operation was used to perform the binding process, this architecture came to be known as a "convolution-based" architecture. More generally, this architecture is an example of a vector symbolic architecture (VSA), which share some of the same concepts as the convolution-based architectures. In VSAs, concepts are also represented using vectors, and they also have some notion of binding and unbinding operations. This chapter begins with a description of these fundamental concepts, followed by an examination and critique of the three aforementioned models.

2.1 Vector Symbolic Architectures

Vector symbolic architectures are a class of models that are well-suited for use in cognitive modelling [Gayler, 2003]. As the name suggests, VSAs use vectors to represent symbols or concepts (or in the case of this thesis, items from a list) within the system. The use of vector representation offers VSAs and advantage over other architectures in terms of efficiency in scalability, continuity, and analogical representation [Plate, 2003]. Scalability refers to the amount of additional resources required when enlarging the representation to accommodate a bigger system. Continuity refers to the ability to represent values on a continuous scale, as opposed to a set of specific values on a discrete scale. Analogical representation refers to the ability to represent similar concepts with vector representations that are themselves similar.

As previously mentioned, a VSA contains a set of algebraic operations that can be used to manipulate the vector representations. More precisely, three types of algebraic operations are necessary in a VSA: a merging operation, a binding operation, and an unbinding operation. Using the bag-and-object example from before, the binding and unbinding operations can be thought of as packing and unpacking balls of malleable clay into a bag. Without applying additional force, these balls of clay will not stick to each other and can be individually packed and removed from the bag without much damage. The merging operation, however, is like mashing all the balls of clay into one giant ball. Although the giant ball of clay contains all of the "information" of the constituent balls, extracting the individual balls of clay will be more difficult. This section describes each of these operations with respect to VSAs and clarifies the need for each operation.

2.1.1 Merging

The merging operation takes two vectors and combines them, generating a third vector. The resulting vector has the property that it is similar to both of its constituent vectors. With vectors, this operation is simply a vector superposition, as demonstrated by Figure 2.1. Vector superposition is also known as vector addition, and is analogous to calculating the addition of two numbers in the scalar domain. The merging operation performs the role similar to that of addition in algebraic mathematics. Just like scalar addition, the merging operation is associative, commutative, and distributive. In this paper, the addition symbol (+) is used to represent the merging operation.

2.1.2 Binding

The binding operation is similar to the merging operation in the way that it takes two vectors and produces a third result vector. In addition to that, the binding operation is also associative, commutative, and distributive. However, unlike the merging operation, where the resulting vector is similar to its constituent vectors, the binding operation must produce a resulting vector that is

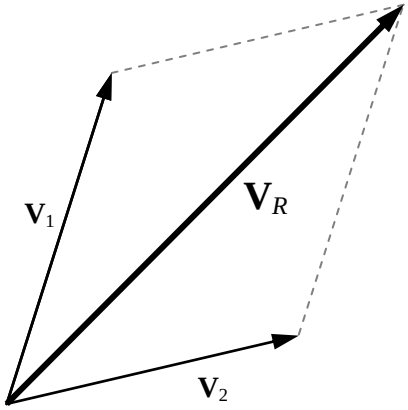

Figure 2.1: Illustration of the method of vector superposition, or vector addition. In the figure, two 2-dimensional vectors, $\mathbf{V}_1$ and $\mathbf{V}_2$, are added together, resulting in a third 2-dimensional vector $\mathbf{V}_R$ that is similar to both $\mathbf{V}_1$ and $\mathbf{V}_2$.

highly dissimilar from the original vectors. Vector similarity can be calculated using any measure in vector space, but is usually determined using the normalized vector dot product. Many mathematical operations can be used to implement the binding operation, convolution [Murdock, 1979] and tensor products [Smolensky, 1990] are two such examples. Just as the merging operation is analogous to addition, the binding operation is analogous to multiplication in algebraic mathematics. In this thesis, it is represented using the symbol $(*)$.

2.1.3 The Identity Vector

In scalar multiplication, there is a special rule that any number multiplied by 1 results in the original number. This special case is also required for the binding operation in a VSA. A vector with this property is called the identity vector, and is given the symbol (I). In the case where convolution is used as the binding operation, the identity vector is a vector in which the middle element is 1, and the rest of the elements are 0 ($I = [\ldots, 0, 0, 0, 1, 0, 0, 0, \ldots]$). Equation (2.1) summarizes the effect binding an arbitrary vector $\mathbf{A}$ with the identity vector.

$$\mathbf{A} * I = \mathbf{A} \tag{2.1}$$

2.1.4 Unbinding

Since the binding operation results in a vector that is highly dissimilar from the original vectors, a method of unbinding the vectors must be provided. As with the binding operation, the unbinding operation can also be implemented using a mathematical operation, for example the correlation

function used by Murdock [Murdock, 1979]. It is more common, however, to define an operation called the inverse operation; with the property that binding a vector with its inverse results in the identity vector. The inverse operation is denoted by a superscript $^{-1}$ in this thesis.

$$\mathbf{A} * \mathbf{A}^{-1} = I \tag{2.2}$$

It is then possible to unbind previously bound vectors by applying the binding operation to the bound vector and the inverse of the vector that is associated with the vector to be unbound. The following example illustrates this more clearly. Using $\mathbf{V}_{AB}$ to denote the result of applying the binding operation to vectors $\mathbf{A}$ and $\mathbf{B}$, extracting the vector $\mathbf{A}$ from $\mathbf{V}_{AB}$ is as simple as binding $\mathbf{V}_{AB}$ to the inverse of $\mathbf{B}$.

$$\begin{aligned}
\mathbf{V}_{AB} &= \mathbf{A} * \mathbf{B} \\
\mathbf{V}_{AB} * \mathbf{B}^{-1} &= (\mathbf{A} * \mathbf{B}) * \mathbf{B}^{-1} \\
&= \mathbf{A} * (\mathbf{B} * \mathbf{B}^{-1}) \\
&= \mathbf{A} * I \\
&= \mathbf{A}
\end{aligned}$$

It is trivial to see that this unbinding process can also be used with the inverse of $\mathbf{A}$ to retrieve the vector $\mathbf{B}$. If a vector needs to be extracted from the bound result of two or more vectors, then the unbinding process needs to be applied with the inverse of every undesired vector. For example, if the vector $\mathbf{B}$ is desired from the bound result of $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$, then the unbinding operation needs to be applied with $\mathbf{A}^{-1}$, $\mathbf{C}^{-1}$, and $\mathbf{D}^{-1}$.

$$\begin{aligned}
\mathbf{V}_{ABCD} &= \mathbf{A} * \mathbf{B} * \mathbf{C} * \mathbf{D} \\
\mathbf{V}_{ABCD} * \mathbf{A}^{-1} * \mathbf{C}^{-1} * \mathbf{D}^{-1} &= (\mathbf{A} * \mathbf{B} * \mathbf{C} * \mathbf{D}) * \mathbf{A}^{-1} * \mathbf{C}^{-1} * \mathbf{D}^{-1} \\
&= (\mathbf{A} * \mathbf{A}^{-1}) * \mathbf{B} * (\mathbf{C} * \mathbf{C}^{-1}) * (\mathbf{D} * \mathbf{D}^{-1}) \\
&= I * \mathbf{B} * I * I \\
&= \mathbf{B}
\end{aligned}$$

Having described the fundamental concepts behind VSAs, the following sections will examine how a specific type of VSA, mentioned earlier as a convolution-based architecture, is used to model serial memory.

2.2 CADAM

The CADAM (Content Addressable Distributed Associative Memory) model was developed by Peter Liepa in 1977, and it is one of the first VSA-based memory models that could be used to encode a sequence of items into one single memory trace. A "memory trace" is a stand-alone

vector that is the result of applying the encoding process to a sequence of items, where each item is also represented using a vector (this vector is referred to as the "item vector"). Note that since memory traces are stand-alone vectors – just like the item vectors – additional encoding operations can be applied to collections of memory traces to form bigger memory traces.

The CADAM model has a distinct encoding scheme for each of the three types of information: item information, associative information, and serial-order information. The following sections briefly discuss each one of these encoding schemes.

2.2.1 Item Information Encoding

A memory trace of a list with purely item information (henceforth referred to as an "item list") is encoded as the superposition of all the vectors representing the list items. For an example list containing three items: **A**, **B**, and **C**, the resulting memory trace would be: $\mathbf{A} + \mathbf{B} + \mathbf{C}$. More generally, for a list of n items, the encoded memory trace would simply be the superposition of all the item vectors.

$$\mathbf{M} = \sum_{i=1}^{n} \mathbf{I}_i, \tag{2.3}$$

where, $\mathbf{M}$ is the vector representing the encoded memory, and $\mathbf{I}_i$ denotes the item vector for the i^{th} item in the list.

The item retrieval process for such an encoding is simple. To retrieve items from the memory trace, it must be probed by the item in question. This would be akin to an experimenter asking the subject "Was item **A** in the list?" To probe the memory trace for an item, dot product is used to calculate the similarity of the memory trace with the probe vector. If the probe vector exists within the memory trace, then the result of the dot product would return a strength s indicating the strength of the retrieval. If all of the item vectors used to encode the memory trace were orthogonal to each other, then this strength would be approximately 1. If the probe vector is not contained within the memory trace, then the value of s would be less than or equal to 0.

$$\mathbf{M} \bullet \mathbf{P} = \begin{cases} s & \text{if probed item exists} \\ \leq 0 & \text{if probed item does not exist} \end{cases} \tag{2.4}$$

In the equation above $\mathbf{P}$ denotes the item vector of the probed item, and $(\bullet)$ represents the dot product (or inner product) between the memory trace vector and the probed item vector. It is notable to mention that the CADAM encoding scheme for item lists requires a probe vector to perform item retrieval. That being the case, this encoding scheme is thus unable to handle free recall, a task in which the subject is instructed to recall the list items without being provided with a probed item.

2.2.2 Associative Information Encoding

A more complex encoding mechanism is needed for lists with associative information. For this type of list, the items are typically presented in pairs (and thus referred to as a "paired item list" from this point forward), which allows associations to be formed between each item. The CADAM model does not address lists for which more than two items are presented at the same time. To encode paired item lists, each pair is bound together using the binding operation, and then added together using the merging operation. This is illustrated in Equation (2.5). In the equation below, **I** and **J** are used to represent the items of each pair in the paired item list.

$$\mathbf{M} = \sum_{i=1}^{n} \mathbf{I}_i * \mathbf{J}_i \tag{2.5}$$

Items in a paired item list are retrieved in a similar fashion to the item lists. In this case, however, rather than probing the memory trace with the desired item vector, the memory trace is probed with the paired associate of the item in question. This would be similar to an experimenter asking a subject "What was the item paired with **A**?". The decoding equation is thus of the form:

$$\mathbf{I} = \mathbf{M} * \mathbf{P}^{-1} \tag{2.6}$$

This concept is better illustrated with an example. Consider the list [**A**, **X**], [**B**, **Y**], [**C**, **Z**]. Using Equation (2.5), the resulting encoded memory trace would be: $\mathbf{M} = \mathbf{A} * \mathbf{X} + \mathbf{B} * \mathbf{Y} + \mathbf{C} * \mathbf{Z}$. Retrieving one item from the first pair of items, for example, would then involve applying the binding operation to the inverse of either **A** or **X**.

To find what was paired with **A**:

$$\begin{aligned}
\mathbf{I} &= \mathbf{M} * \mathbf{A}^{-1} \\
&= (\mathbf{A} * \mathbf{X} + \mathbf{B} * \mathbf{Y} + \mathbf{C} * \mathbf{Z}) * \mathbf{A}^{-1} \\
&= \mathbf{A} * \mathbf{X} * \mathbf{A}^{-1} + \mathbf{B} * \mathbf{Y} * \mathbf{A}^{-1} + \mathbf{C} * \mathbf{Z} * \mathbf{A}^{-1} \\
&= (\mathbf{A} * \mathbf{X} * \mathbf{A}^{-1} + \mathbf{C} * \mathbf{Z} * \mathbf{A}^{-1}) + \mathbf{B} * \mathbf{Y} * \mathbf{A}^{-1} \\
&= noise + \mathbf{X} \approx \mathbf{X}
\end{aligned}$$

likewise, to find what was paired with **X**:

$$\begin{aligned}
\mathbf{I} &= \mathbf{M} * \mathbf{X}^{-1} \\
&= (\mathbf{A} * \mathbf{X} + \mathbf{B} * \mathbf{Y} + \mathbf{C} * \mathbf{Z}) * \mathbf{X}^{-1} \\
&= \mathbf{A} * \mathbf{X} * \mathbf{X}^{-1} + \mathbf{B} * \mathbf{Y} * \mathbf{X}^{-1} + \mathbf{C} * \mathbf{Z} * \mathbf{X}^{-1} \\
&= (\mathbf{A} * \mathbf{X} * \mathbf{X}^{-1} + \mathbf{C} * \mathbf{Z} * \mathbf{X}^{-1}) + \mathbf{B} * \mathbf{Y} * \mathbf{X}^{-1} \\
&= noise + \mathbf{A} \approx \mathbf{A}
\end{aligned}$$

It is trivial to see that items from the second pair can be retrieved by binding the memory trace to the inverse of either **B** or **Y**, and that items from the third pair can be retrieved by binding the memory trace to the inverse of either **C** or **Z**. As with the item list encoding, there is no mechanism in the paired list encoding that allows for the extraction of item vectors without using a probed recall. Interestingly, there is also no mechanism in the paired list encoding that allows for questions like "Was item **A** in the list?" to be successfully answered, even though this was trivial in the non-paired item list encoding scheme.

2.2.3 Serial-order Information Encoding

For items with serial-order information, the encoding becomes even more complex. Before deriving the general encoding formulae however, it is best to illustrate the basic encoding concept of this encoding scheme with an example. For a list with just one item, **A**, the resulting memory trace $\mathbf{M}_A$ would just be **A**:

$$\mathbf{M}_A = \mathbf{A} \tag{2.7}$$

Adding one more item to the list, **B**, would result in:

$$\mathbf{M}_{AB} = \mathbf{A} + \mathbf{A} * \mathbf{B} \tag{2.8}$$

Inserting yet another item, **C**, to the list results in the following:

$$\mathbf{M}_{ABC} = \mathbf{A} + \mathbf{A} * \mathbf{B} + \mathbf{A} * \mathbf{B} * \mathbf{C} \tag{2.9}$$

From here, a pattern emerges. As subsequent items are added to the list, the resulting memory trace is formed by binding the "newest" items to all the previous items and then adding the bound result to the memory trace for a list excluding the "newest" item.

From the example lists, a more generic encoding equation can be derived. Using the variable **C** to represent the running result of binding each item as they are presented (i.e. in the example above, $\mathbf{C}_A = \mathbf{A}$, $\mathbf{C}_{AB} = \mathbf{A} * \mathbf{B}$, $\mathbf{C}_{ABC} = \mathbf{A} * \mathbf{B} * \mathbf{C}$, and so forth), the encoding equations are formulated below.

$$\begin{aligned} \mathbf{C}_i &= \mathbf{C}_{i-1} * \mathbf{I}_i \\ \mathbf{M}_i &= \mathbf{M}_{i-1} + \mathbf{C}_i, \\ \text{where } \mathbf{C}_0 &= I \text{ and } \mathbf{M}_0 = 0 \end{aligned} \tag{2.10}$$

The subscript for the variables **C** and **M** indicate which item the encoding equations are for. For example, $\mathbf{M}_i$ represents the memory trace after encoding the i^{th} item, and likewise, $\mathbf{C}_i$ represents the result of binding all the list items up to the i^{th} item. The identity vector is used for the initial value of $\mathbf{C}_0$ because of the property $I * \mathbf{A} = \mathbf{A}$. Substituting these initial values into the

serial-order encoding equations, it can then be shown that for the first item,

$$\begin{aligned}\mathbf{C}_1 &= \mathbf{C}_0 * \mathbf{I}_1 = I * \mathbf{I}_1 = \mathbf{I}_1 \text{ , and} \\ \mathbf{M}_1 &= \mathbf{M}_0 + \mathbf{C}_1 = 0 + \mathbf{I}_1 = \mathbf{I}_1\end{aligned}$$

which satisfies Equation (2.7). A simple analysis of the encoding equation indicates that as more items are added to the list, the contribution from that item to the resulting memory trace gets less and less significant because of the binding operation with the rest of the items in the list. As will be discussed shortly, this means that items at the start of the list will have a better recall probability, resulting in a primacy effect in the recall probabilities. There is no obvious mechanism that would generate a recency effect though.

The item retrieval process for lists with serial-order information is slightly more involved than the retrieval scheme used for the previous two types of list. As with the encoding process, the item retrieval process is better illustrated with an example. Consider the three-itemed list, $[\mathbf{A}, \mathbf{B}, \mathbf{C}]$. From Equation (2.10), $\mathbf{M} = \mathbf{A} + \mathbf{A} * \mathbf{B} + \mathbf{A} * \mathbf{B} * \mathbf{C}$ would be the resulting encoded memory trace of the three itemed list. The retrieval procedure is then as follows:

$$\begin{aligned}\mathbf{I}_1 &= \mathbf{M} * I^{-1} \\ &= \mathbf{A} + (\mathbf{A} * \mathbf{B} + \mathbf{A} * \mathbf{B} * \mathbf{C}) \\ &= \mathbf{A} + noise \approx \mathbf{A} \\ \mathbf{I}_2 &= (\mathbf{M} - \mathbf{A}) * \mathbf{A}^{-1} \\ &= (\mathbf{A} - \mathbf{A} + \mathbf{A} * \mathbf{B} + \mathbf{A} * \mathbf{B} * \mathbf{C}) * \mathbf{A}^{-1} \\ &= \mathbf{A} * \mathbf{B} * \mathbf{A}^{-1} + \mathbf{A} * \mathbf{B} * \mathbf{C} * \mathbf{A}^{-1} \\ &= \mathbf{B} + noise \approx \mathbf{B} \\ \mathbf{I}_3 &= (\mathbf{M} - \mathbf{A} - \mathbf{A} * \mathbf{B}) * (\mathbf{A} * \mathbf{B})^{-1} \\ &= (\mathbf{A} - \mathbf{A} + \mathbf{A} * \mathbf{B} - \mathbf{A} * \mathbf{B} + \mathbf{A} * \mathbf{B} * \mathbf{C}) * \mathbf{B}^{-1} \\ &= \mathbf{A} * \mathbf{B} * \mathbf{C} * (\mathbf{A} * \mathbf{B})^{-1} \\ &= \mathbf{C}\end{aligned}$$

Analysis of the retrieval process reveals that three variables are needed for the generic retrieval (decoding) equations – $\mathbf{I}$ for the item being retrieved, $\mathbf{MI}$ for the intermediate memory trace used at each step of the decoding process, and $\mathbf{C}$ for the running result of the binding operation applied to each item retrieved. With these variables, it is then possible to set up a series of equations to describe the decoding process.

$$\begin{aligned}\mathbf{I}_i &= \mathbf{MI}_{i-1} * \mathbf{C}_{i-1}^{-1} \\ \mathbf{C}_i &= \mathbf{C}_{i-1} * \mathbf{I}_i \\ \mathbf{MI}_i &= \mathbf{MI}_{i-1} - \mathbf{C}_i\end{aligned} \tag{2.11}$$

In Equation (2.11), the initial value for $\mathbf{MI}_0$ is the encoded memory trace obtained using Equation (2.10), and the initial value for $\mathbf{C}_0$ is the identity vector, I.

2.2.4 Results and Discussion

Liepa did not provide any formal simulations for his CADAM encoding schemes. However, I have implemented the model in Matlab®as well as in a network of spiking neurons to compare the behaviour of an ideal mathematical implementation to the implementation in a spiking neuron network. The results obtained are included below. The spiking neuron network was constructed by applying the Neural Engineering Framework to the equations laid out in (2.10) and (2.11). This framework will be discussed in further detail in Section 4.2. Figure 2.2 compares the results of the simulation for an ideal (purely mathematical) implementation and the spiking neural network implementation for lists varying in length from 3 to 7 items. 30-dimensional item vectors were used in the simulation.

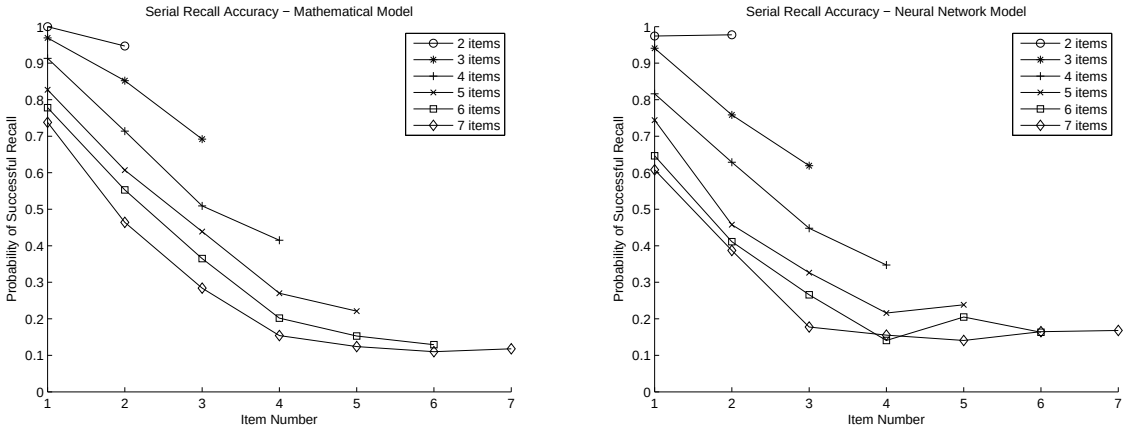

Figure 2.2: Simulation results of the CADAM model implemented in spiking neuron network. Also illustrated is the simulation results of an ideal (Matlab®) implementation of the CADAM model

As the simulation shows, and as it was predicted, the CADAM encoding scheme for serially ordered lists is able to produce a pronounced primacy effect in the recall data, but is unable to replicate the recency effect that is seen in humans (see Figure 1.1). Additionally, because subsequent list items are bound to previous list items, it is not possible to perform probed recall on a memory trace encoded with the CADAM serial-order encoding scheme. The simulation does, however, show that the neural implementation is able to match the results of the ideal implementation. As a result, it is at least possible, in principle, to implement these kinds of models in a biologically realistic manner.

Several failings of the CADAM encoding schemes have already been highlighted, namely that in the item list encoding scheme, items are only retrievable via probed recall; in the paired-

association list encoding scheme, items cannot be retrieved via probed recall; and in the serial-order list encoding scheme, there are no mechanisms by which to reproduce the recency effect. The CADAM encoding schemes do however, provide a solid foundation from which other models can be constructed, and they are demonstrably neurally plausible.

2.3 TODAM

The TODAM (Theory of Distributed Associative Memory) model was developed by Murdock in 1982 and extended to encode lists with serial-order information in 1983. The TODAM model combines the CADAM encoding scheme for item lists and paired associations in what Murdock calls the "chaining" method. As each item is presented, the item vector and the paired associate combining the item vector with the preceding item vector are added to the memory trace. With this encoding style, the list $[\mathbf{A}, \mathbf{B}, \mathbf{C}]$ would be encoded as: $\mathbf{M}_{ABC} = \mathbf{A}+\mathbf{B}+\mathbf{A}*\mathbf{B}+\mathbf{C}+\mathbf{B}*\mathbf{C}$.[1] To decode the list, the memory trace is convolved with the inverse of the first item, which results in the second item. The process is then repeated with the second item to obtain the third item.

$$\begin{aligned}
\mathbf{M}_{ABC} * \mathbf{A}^{-1} &= (\mathbf{A} + \mathbf{B} + \mathbf{A} * \mathbf{B} + \mathbf{C} + \mathbf{B} * \mathbf{C}) * \mathbf{A}^{-1} \\
&= \mathbf{A} * \mathbf{A}^{-1} + \mathbf{B} * \mathbf{A}^{-1} + \mathbf{A} * \mathbf{B} * \mathbf{A}^{-1} + \mathbf{C} * \mathbf{A}^{-1} + \mathbf{B} * \mathbf{C} * \mathbf{A}^{-1} \\
&= I + \mathbf{B} + noise \approx \mathbf{B} \\
\mathbf{M}_{ABC} * \mathbf{B}^{-1} &= (\mathbf{A} + \mathbf{B} + \mathbf{A} * \mathbf{B} + \mathbf{C} + \mathbf{B} * \mathbf{C}) * \mathbf{B}^{-1} \\
&= \mathbf{A} * \mathbf{B}^{-1} + \mathbf{B} * \mathbf{B}^{-1} + \mathbf{A} * \mathbf{B} * \mathbf{B}^{-1} + \mathbf{C} * \mathbf{B}^{-1} + \mathbf{B} * \mathbf{C} * \mathbf{B}^{-1} \\
&= I + \mathbf{A} + \mathbf{C} + noise \approx \mathbf{C} + \mathbf{A} \Rightarrow \mathbf{C}
\end{aligned} \tag{2.12}$$

This encoding scheme poses some problems. Firstly, the first item needs to be known to initiate the recall procedure. In a task where the goal is to correctly recall the all of the list items, this seems hopelessly recursive. The TODAM model solves this problem by using a predefined vector to encode the start of the list. Thus, $\mathbf{M}_{ABC} = \mathbf{X} + \mathbf{A} + \mathbf{X} * \mathbf{A} + \mathbf{B} + \mathbf{A} * \mathbf{B} + \mathbf{C} + \mathbf{B} * \mathbf{C}$, where $\mathbf{X}$ is the predefined start vector.

As the last line of Equation (2.12) illustrates, the second problem is that convolving the memory trace with the inverse of an item vector produces two vector results: the item vector that precedes the item vector being used ($\mathbf{A}$ in the example), and the item vector that succeeds the item vector being used ($\mathbf{C}$ in the example). Murdock refers to this problem as the "deblurring" problem. The original TODAM model does not specify how the preceding item vector is eliminated during the cleanup process, but Lewandowksy and Li (1994) suggest that a Hebbian learning mechanism could be trained to remove previously recalled items.

Although it was neither proposed by the original authors nor by Lewandowsky and Li, it should also be possible to solve this problem by subtracting the preceding item from the result

[1] Note that the first item is not preceded by any other items, so there is no paired associate term.

after the item probe is applied.

$$\begin{aligned} &\mathbf{I}_i = \mathbf{M} * \mathbf{I}_{i-1}^{-1} - \mathbf{I}_{i-2}, \text{ where} \\ &\mathbf{I}_0 = \mathbf{X}, \text{ and } \mathbf{I}_{-1} = 0 \end{aligned} \tag{2.13}$$

To illustrate this with an example, the memory trace for $\mathbf{M}_{ABC}$ derived above is used.

$$\mathbf{M}_{ABC} \quad = \quad \mathbf{X} + \mathbf{A} + \mathbf{X} * \mathbf{A} + \mathbf{B} + \mathbf{A} * \mathbf{B} + \mathbf{C} + \mathbf{B} * \mathbf{C}$$

Recalling the first item:

$$\begin{aligned} \mathbf{I}_1 &= \mathbf{M}_{ABC} * \mathbf{I}_0^{-1} - \mathbf{I}_{-1} \\ &= \mathbf{M}_{ABC} * \mathbf{X}^{-1} - 0 \\ &= (\mathbf{A} + \mathbf{X} * \mathbf{A} + \mathbf{B} + \mathbf{A} * \mathbf{B} + \mathbf{C} + \mathbf{B} * \mathbf{C}) * \mathbf{X}^{-1} \\ &= \mathbf{A} * \mathbf{X}^{-1} + \mathbf{X} * \mathbf{A} * \mathbf{X}^{-1} + \mathbf{B} * \mathbf{X}^{-1} + \mathbf{A} * \mathbf{B} * \mathbf{X}^{-1} + \mathbf{C} * \mathbf{X}^{-1} + \mathbf{B} * \mathbf{C} * \mathbf{X}^{-1} \\ &= noise + \mathbf{A} \approx \mathbf{A} \end{aligned}$$

Recalling the second item:

$$\begin{aligned} \mathbf{I}_2 &= \mathbf{M}_{ABC} * \mathbf{I}_1^{-1} - \mathbf{I}_0 \\ &= \mathbf{M}_{ABC} * \mathbf{A}^{-1} - \mathbf{X} \\ &= (\mathbf{A} + \mathbf{X} * \mathbf{A} + \mathbf{B} + \mathbf{A} * \mathbf{B} + \mathbf{C} + \mathbf{B} * \mathbf{C}) * \mathbf{A}^{-1} - \mathbf{X} \\ &= (\mathbf{A} * \mathbf{A}^{-1} + \mathbf{X} * \mathbf{A} * \mathbf{A}^{-1} + \mathbf{B} * \mathbf{A}^{-1} + \mathbf{A} * \mathbf{B} * \mathbf{A}^{-1} + \mathbf{C} * \mathbf{A}^{-1} + \mathbf{B} * \mathbf{C} * \mathbf{A}^{-1}) - \mathbf{X} \\ &= (noise + \mathbf{X} + \mathbf{B}) - \mathbf{X} \approx \mathbf{B} \end{aligned}$$

And finally, recalling the third item:

$$\begin{aligned} \mathbf{I}_3 &= \mathbf{M}_{ABC} * \mathbf{I}_2^{-1} - \mathbf{I}_1 \\ &= \mathbf{M}_{ABC} * \mathbf{B}^{-1} - \mathbf{A} \\ &= (\mathbf{A} + \mathbf{X} * \mathbf{A} + \mathbf{B} + \mathbf{A} * \mathbf{B} + \mathbf{C} + \mathbf{B} * \mathbf{C}) * \mathbf{B}^{-1} - \mathbf{A} \\ &= (\mathbf{A} * \mathbf{B}^{-1} + \mathbf{X} * \mathbf{A} * \mathbf{B}^{-1} + \mathbf{B} * \mathbf{B}^{-1} + \mathbf{A} * \mathbf{B} * \mathbf{B}^{-1} + \mathbf{C} * \mathbf{B}^{-1} + \mathbf{B} * \mathbf{C} * \mathbf{B}^{-1}) - \mathbf{A} \\ &= (noise + \mathbf{A} + \mathbf{C}) - \mathbf{A} \approx \mathbf{C} \end{aligned}$$

The accuracy of this modified decoding scheme is uncertain, and further investigation would be needed to verify its feasibility. It is also unclear how incorrectly recalled items, or items below the recall threshold will affect this modified decoding scheme.

The complete TODAM model also includes additional parameters to better fit the human

data, resulting in the encoding formula below:

$$\mathbf{M}_i = \alpha \mathbf{M}_{i-1} + \gamma_i \mathbf{I}_i + \Omega_i (\mathbf{I}_i * \mathbf{I}_{i-1}), \tag{2.14}$$

where α is a forgetting factor that is used to simulate interference as more items are added to the encoded list, γ is an attentional parameter used to determine the contribution that item information has on the memory trace, and Ω is an attentional parameter used to determine the contribution that order information has on the memory trace. Additionally, γ and Ω sums to 1, and Ω changes with respect to the position of the item in the list.

$$\Omega_i = \begin{cases} 1, & i = 1 \\ \Omega_0 e^{\lambda(i-2)}, & i > 1, \end{cases} \tag{2.15}$$

where Ω_0 is the initial value of Ω and the parameter λ determines the rate of change of Ω. Note that in the equations above, the start vector is indicated by an index (i) value of 1. As before, the initial values of $\mathbf{M}_0$ and $\mathbf{I}_0$ are 0 and I respectively.

2.3.1 Results and Discussion

In 1989, Lewandowsky and Murdock published a paper that compared the TODAM model to human data. The results of the simulations and comparisons are briefly discussed here. The TODAM model was benchmarked against several experimental tasks. The standard serial recall task and the backwards recall task are two examples of the experimental tasks used for the benchmarking. As Figure 2.3 illustrates, the TODAM model performs well on both of these tasks. The TODAM model is also able to match human data on the other experimental tasks that are not detailed in this thesis.

There are, however, several flaws with the model. Firstly, subsequent human studies involving serial recall with different list configurations provided evidence against a chaining-based model of serial memory [Henson et al., 1996]. In this study, the authors presented subjects with sequences of items that alternate between confusable and non-confusable letters. "B", "D", and "G" are examples of confusable letters whereas "Q", "R", and "Y" are examples of non-confusable letters. By presenting their subjects with these types of lists, the authors found dips in the recall accuracy for confusable items, whereas non-confusable items suffered no loss in recall accuracy (see Figure 2.4), contrary to what the TODAM model would predict. Because of the chaining mechanism employed by the TODAM model, errors in recall of one item would result in an increase in the recall error of the subsequent items. This means that if given a sequence with alternating confusable and non-confusable items, the recall accuracy would decrease for all the items, not just for the confusable items.

In addition to the previously mentioned problems, the results presented in the 1989 paper adjust the model's parameters to fit the human data from each of the experimental benchmarks. Although the changes in the model parameters might have been required because the experiments

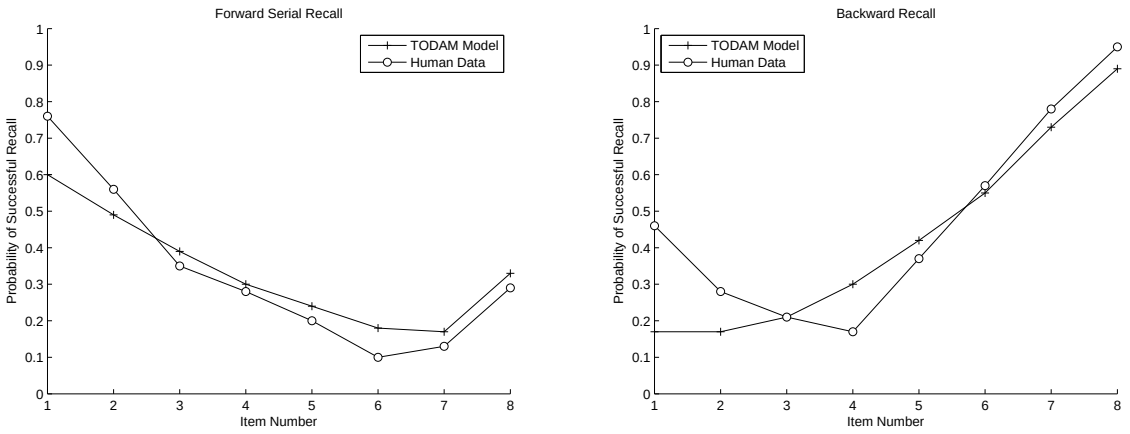

Figure 2.3: A comparison of the recall performance of the TODAM model versus data collected from human subjects for the tasks of forward and backward serial recall. From [Lewandowsky and Murdock, 1989, Figure 23 – visual modality].

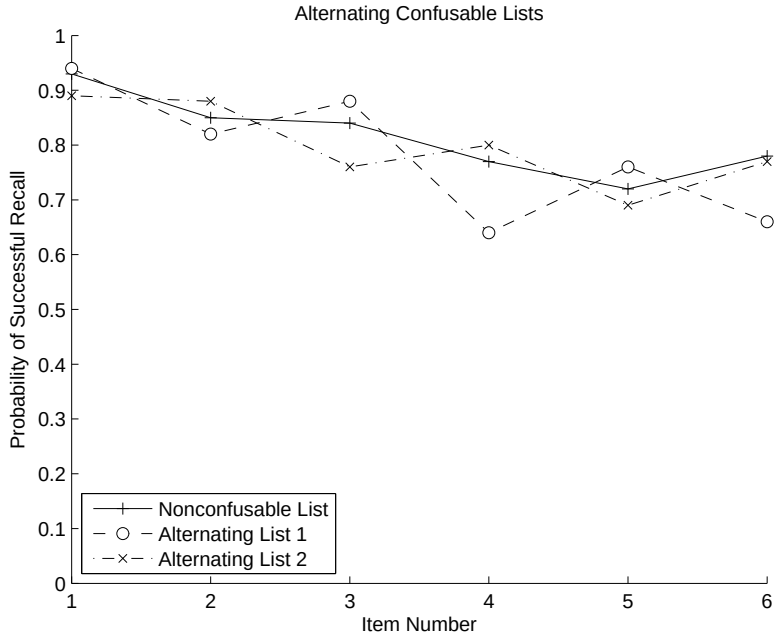

Figure 2.4: Plot of the human recall performance for lists containing confusable items. Alternating list 1 contained confusable items at every even-numbered position, while alternating list 2 contained confusable items at every odd numbered position. The non-confusable list contained no confusable items and served as the control for this experiment. From [Henson et al., 1996, Figure 1].

involved a different set of subjects, it could also be argued that all the authors have successfully accomplished is to create a model with enough adjustable parameters to fit all the benchmark human data. There is also little explanation to why each parameter is set to a certain value, apart from the fact that those values provide the best fits to the human data. In a 1995 paper, Murdock also admits that the chaining model had difficulty performing tasks such as the recall of missing items, and sequential and positional probes. It is easy to see why the chaining model would be unable to perform the positional probe task. Since no positional information is encoded within the memory trace, trying to identify the position of a given item, or trying to identify the item in a given would require the model to unpack the memory trace to produce any usable results. This means that the TODAM model is unable to reproduce all of the effects seen in human positional probe data. The TODAM model also fails at the sequential probe because it relies on the previously recalled items to determine the direction of recall (see the "deblurring" problem: Equation 2.12), the model would be unable to determine the correct direction of recall if it was initiated in the middle of the list.

2.4 TODAM2

In 1993, in response to the criticisms of the TODAM chaining model, Murdock proposed an improved model called TODAM2, which is actually an extension of another model that Murdock developed that he refers to as the "chunking" model. Murdock developed the chunking model as a non-chaining method for encoding lists with serial-order information, and then extended it to encompass the encoding of item and paired-item lists. Because the complete TODAM2 model is highly complex, this thesis will concentrate mainly on the chunking model and only briefly describe the workings of the full TODAM2 model.

The Chunking Model

The chunking model employs three new concepts: convolution powers, n-grams, and chunks. Convolution powers are analogous to exponents in the scalar domain. In this thesis, the notation $\mathbf{A}^{*n}$ will be used to denote the n^{th} convolution power of the vector $\mathbf{A}$, similar to the way a^n is used to denote the n^{th} exponential power of the scalar value a. $\mathbf{A}^{*n}$ also represents the convolution of the vector $\mathbf{A}$ with itself (i.e. an auto-convolution) $n-1$ times, the same way the a^n represents the number a multiplied with itself $n-1$ times.

$$\mathbf{A}^{*n} = \mathbf{A} * \mathbf{A} * \mathbf{A} * \ldots * \mathbf{A} \qquad (n \text{ times}) \tag{2.16}$$

An "n-gram"; as Murdock defines it; is "the n-way auto-convolution of the sum of n item vectors" [Murdock, 1995]. Using this definition, an n-gram of one item would just be the item itself; an n-gram of two items, $\mathbf{A}$ and $\mathbf{B}$, would be $(\mathbf{A}+\mathbf{B})^{*2}$; an n-gram of three items, $\mathbf{A}$, $\mathbf{B}$

and $\mathbf{C}$, would be $(\mathbf{A}+\mathbf{B}+\mathbf{C})^{*3}$; and so forth. In more general terms:

$$\mathrm{G}(n) = \left(\sum_{i=1}^{n} \mathbf{I}_i\right)^{*n}, \tag{2.17}$$

where $\mathrm{G}(n)$ is used to represent the n-gram of n items.

From n-grams, chunks can then be constructed. A chunk for m items is defined as the sum of the n-grams for 1 to m items. To illustrate this more clearly,

$$\mathrm{C}(n) = \sum_{i=1}^{n} \mathrm{G}(i), \tag{2.18}$$

where $\mathrm{C}(n)$ denotes the chunk for a list of n items. This chunking equation is the core of the chunking model, and a memory trace for an n-itemed list in the chunking model is really just the chunk created from said list. Essentially,

$$\mathbf{M}_n = \mathrm{C}(n). \tag{2.19}$$

To demonstrate how the chunking model encodes a list, we use the standard 3-list example $[\mathbf{A}, \mathbf{B}, \mathbf{C}]$. With the chunking model, the memory trace would be as such:

$$\begin{aligned} \mathbf{M}_{ABC} &= \mathrm{C}(3) \\ &= \mathrm{G}(1) + \mathrm{G}(2) + \mathrm{G}(3) \\ &= \mathbf{A} + (\mathbf{A}+\mathbf{B})^{*2} + (\mathbf{A}+\mathbf{B}+\mathbf{C})^{*3} \end{aligned} \tag{2.20}$$

Decoding a memory trace that has been encoded using the chunking model is similar to the decoding scheme proposed by Liepa for the CADAM model. In the CADAM model (see Section 2.11), an item is retrieved by convolving an intermediate memory trace with the inverse of the running convolution of all the items already retrieved. In the chunking model, the running convolution is not convolved with the intermediate memory trace; rather, it is convolved with the list's encoded memory trace instead. Using the memory trace encoded in Equation (2.20), an example unpacking of the list items follows. To simplify the decoding example, the fully expanded results of each n-gram is listed below.

$$\mathrm{G}(1) = \mathbf{A} \tag{2.21}$$

$$\begin{aligned} \mathrm{G}(2) &= (\mathbf{A}+\mathbf{B})^{*2} \\ &= \mathbf{A}^{*2} + 2(\mathbf{A}*\mathbf{B}) + \mathbf{B}^{*2} \end{aligned} \tag{2.22}$$

$$\begin{aligned} \mathrm{G}(3) &= (\mathbf{A}+\mathbf{B}+\mathbf{C})^{*3} \\ &= \mathbf{A}^{*3} + \mathbf{B}^{*3} + \mathbf{C}^{*3} + 3[(\mathbf{A}^{*2}*\mathbf{B}) + (\mathbf{A}^{*2}*\mathbf{C}) + (\mathbf{B}^{*2}*\mathbf{A}) \\ &\quad + (\mathbf{B}^{*2}*\mathbf{C}) + (\mathbf{C}^{*2}*\mathbf{A}) + (\mathbf{C}^{*2}*\mathbf{B})] + 6(\mathbf{A}*\mathbf{B}*\mathbf{C}) \end{aligned} \tag{2.23}$$

To recall the first item, the memory trace is simply passed through a cleanup memory module to isolate the item vector with the highest contribution in the memory trace. By expanding the result of the chunking formula, we can see that this is always the first item.

$$\begin{aligned}\mathbf{M}_{ABC} &= \mathbf{A} + (\mathbf{A}+\mathbf{B})^{*2} + (\mathbf{A}+\mathbf{B}+\mathbf{C})^{*3} \\ &= \mathbf{A} + [\mathbf{A}^{*2} + 2(\mathbf{A}*\mathbf{B}) + \mathbf{B}^{*2}] + \mathrm{G}(3) \\ &= \mathbf{A} + noise \end{aligned} \tag{2.24}$$

Analysis of the Equations (2.23) and (2.24) show that since the vector matching item **A** is the only vector that is contained within the cleanup memory's vocabulary, it would be the only vector that the cleanup memory module would pick out. Thus, passing the encoded memory trace through a cleanup memory module will always result in the retrieval of the first item.[2]

Similar to the CADAM model, retrieving the second item of the sequence would then involve convolving the memory trace with the inverse of the first item retrieved.

$$\begin{aligned}\mathbf{M}_{ABC} * \mathbf{A}^{-1} &= [\mathbf{A} + (\mathbf{A}+\mathbf{B})^{*2} + (\mathbf{A}+\mathbf{B}+\mathbf{C})^{*3}] * \mathbf{A}^{-1} \\ &= (\mathbf{A} * \mathbf{A}^{-1}) + [\mathbf{A}^{*2} + 2(\mathbf{A}*\mathbf{B}) + \mathbf{B}^{*2}] * \mathbf{A}^{-1} + (\mathrm{G}(3) * \mathbf{A}^{-1}) \\ &= I + [\mathbf{A}^{*2} * \mathbf{A}^{-1} + 2(\mathbf{A}*\mathbf{B}) * \mathbf{A}^{-1} + \mathbf{B}^{*2} * \mathbf{A}^{-1}] + noise \\ &= I + [\mathbf{A} + 2(I * \mathbf{B}) + noise] + noise \\ &= I + \mathbf{A} + 2\mathbf{B} + noise \approx \mathbf{B} \end{aligned} \tag{2.25}$$

Note that although the final line in the Equation (2.25) contains the item vectors for both **A** and **B**, the item vector for **B** has twice the strength as the former. Because the cleanup memory module picks the item vector with the best similarity measure, the double strength of the **B** vector would then cause the cleanup memory module to output the second item, **B**, rather than the first item, **A**.

To retrieve the last item from the list, the memory trace is convolved with the inverse of the convolution of both the first and second retrieved item. Because the third n-gram, G(3) needs to be convolved with the inverse term as well, for clarity, the retrieval process for the third item will be broken into two parts – one part to process the first two n-grams, and another part to

[2] This will be the case if the number of dimensions used to represent each item vector is high enough that the vectors comprising the memory trace (e.g. $[\mathbf{A} * \mathbf{B}]$ or $[\mathbf{B}^{*2} * \mathbf{C}]$) are not similar to any of the original item vectors used.

process the third n-gram.

$$\begin{aligned}
\mathbf{M}_{ABC} * (\mathbf{A} * \mathbf{B})^{-1} &= [\mathrm{G}(1) + \mathrm{G}(2) + \mathrm{G}(3)] * (\mathbf{A} * \mathbf{B})^{-1} \\
&= [\mathrm{G}(1) + \mathrm{G}(2)] * (\mathbf{A} * \mathbf{B})^{-1} + [\mathrm{G}(3)] * (\mathbf{A} * \mathbf{B})^{-1} \qquad (2.26) \\
[\mathrm{G}(1) + \mathrm{G}(2)] * (\mathbf{A} * \mathbf{B})^{-1} &= [\mathrm{G}(1) * (\mathbf{A} * \mathbf{B})^{-1}] + [\mathrm{G}(2) * (\mathbf{A} * \mathbf{B})^{-1}] \\
&= [\mathbf{A} * (\mathbf{A} * \mathbf{B})^{-1}] + [(\mathbf{A}^{*2} + 2(\mathbf{A} * \mathbf{B}) + \mathbf{B}^{*2}) * (\mathbf{A} * \mathbf{B})^{-1}] \\
&= noise + [noise + (2(\mathbf{A} * \mathbf{B}) * (\mathbf{A} * \mathbf{B})^{-1}) + noise] \\
&= noise + 2I \approx 2I \qquad (2.27) \\
[\mathrm{G}(3)] * (\mathbf{A} * \mathbf{B})^{-1} &= \{\mathbf{A}^{*3} + \mathbf{B}^{*3} + \mathbf{C}^{*3} + 3[(\mathbf{A}^{*2} * \mathbf{B}) + (\mathbf{A}^{*2} * \mathbf{C}) + (\mathbf{B}^{*2} * \mathbf{A}) \\
&\quad + (\mathbf{B}^{*2} * \mathbf{C}) + (\mathbf{C}^{*2} * \mathbf{A}) + (\mathbf{C}^{*2} * \mathbf{B})] + 6(\mathbf{A} * \mathbf{B} * \mathbf{C})\} * (\mathbf{A} * \mathbf{B})^{-1} \\
&= noise + 3[(\mathbf{A}^{*2} * \mathbf{B}) * (\mathbf{A} * \mathbf{B})^{-1} + (\mathbf{B}^{*2} * \mathbf{A}) * (\mathbf{A} * \mathbf{B})^{-1} + noise] \\
&\quad + 6(\mathbf{A} * \mathbf{B} * \mathbf{C}) * (\mathbf{A} * \mathbf{B})^{-1} \\
&= noise + 3[(I * \mathbf{A}) + (I * \mathbf{B})] + 6(I * \mathbf{C}) \\
&= noise + 3\mathbf{A} + 3\mathbf{B} + 6\mathbf{C} \approx 3\mathbf{A} + 3\mathbf{B} + 6\mathbf{C} \qquad (2.28) \\
\therefore \mathbf{M}_{ABC} * (\mathbf{A} * \mathbf{B})^{-1} &= 2I + 3\mathbf{A} + 3\mathbf{B} + 6\mathbf{C} \\
&\approx \mathbf{C} \qquad (2.29)
\end{aligned}$$

As with the case when retrieving the second item, convolving the inverse item with the memory trace produces additional item vectors that are irrelevant to the final result. However, just as with the second item, the desired item vector has twice the strength of any of the irrelevant vectors. Performing the retrieval exercise for longer lists shows that this will always hold true.

The Complete TODAM2 Model

As mentioned earlier, the complete TODAM2 model is the chunking model extended to enable encoding of both item and paired-item lists. Three more additional concepts need to be introduced to discuss the TODAM2 model, and those are reduced n-grams, labels, and lebals. The "reduced n-gram" is similar to an n-gram, except that it is to the convolution power of $(n-1)$ instead of n. In the following equation, $\mathbf{G}^{-}$ is used to represent the reduced n-gram.

$$\mathrm{G}^{-}(n) = \left(\sum_{i=1}^{n} \mathbf{I}_i \right)^{*(n-1)} \qquad (2.30)$$

A label is a randomly generated vector used to "tag" chunks in the encoded memory trace, and a lebal is the involution of the vector that represents the label.[3] The involution operation

[3] "Lebal" is "label" spelt backwards. This mirrors the property that a lebal is the involution of a label, and the involution operation is a mirror-image operation for vectors. In essence, if the vector $\mathbf{A}$ is $[a_0, a_1, a_2, \ldots, a_n]$, then the involution of $\mathbf{A}$ would be $[a_n, a_{n-1}, a_{n-2}, \ldots, a_0]$.

has the unique property that it approximates the inverse of the label vector. This means that convolving a label with its corresponding lebal results in a vector that is approximately equal to the identity vector. Since the involution operation approximates an inverse operation, in this paper a label is represented with L and a lebal is represented with L^{-1}.

Finally, the encoding equation for the TODAM2 model can finally be put together from all of its components.

$$\mathbf{M}_i = \alpha \mathbf{M}_{i-1} + L_i^{-1} * \mathrm{G}^{-}(i) + L_i * \mathrm{C}(i) \tag{2.31}$$

As with the encoding equation for the original TODAM model, α is used to represent the equation's forgetting factor. The added complication of the TODAM2 model allows it to perform tasks unrelated to serial recall that the original TODAM model is, but the chunking model is not, able to accomplish. Further details on why Murdock chose this form of encoding equation over other forms can be found in his 1993 paper.

2.4.1 Results and Discussion

The chunking model and the TODAM2 model are significantly more complex than the original TODAM model, however it does overcome some of the difficulties seen with the original TODAM model. Notably, it is able to produce grouping effects, where the list is broken down into sublists before memory. This effect is seen in humans [Hitch et al., 1996], and is very easy to implement in the TODAM2 model (each chunk can already be considered a group on its own). The TODAM2 model is also able to perform the missing item recall task, and it does not suffer from the "deblurring" problem that was present in the TODAM model.

There are, however, several drawbacks to the TODAM2 model. The primary issue with the TODAM2 model is the use of the convolution powers as an encoding mechanism. A simple experiment with the convolution powers shows that for every successive convolution, both the magnitude of the vector[4] and the dimensionality of the vector grows exponentially. This has two consequences. First, it means that encoding long lists become prohibitively expensive in terms of the number of dimensions needed for the memory trace vector. For example, with a starting dimensionality of 49 dimensions, after 10 auto-convolutions, the resulting vector has 49153 dimensions. As will be discussed later, the problem of increasing dimensionality can be mitigated using the Holographic Reduced Representation (see Section 3.1), but this still does not solve the second problem, which deals with the magnitude of the vector being represented.

As Figure 2.5 demonstrates, the result of performing 10 auto-convolutions on a vector with an initial magnitude of 1 results in a vector with the final magnitude of 8.46×10^{272}. In terms of biological plausibility, this means that the neurons representing these vectors have to be configured somehow to deal with an enormous range of possible input values. This issue can be resolved by normalizing the result of the auto-convolution after each convolution, but each normalization

[4]The magnitude of a vector is calculated as the square root of the sum of each element squared. Thus, the magnitude of the 2-dimensional vector $[x, y]$ is $\sqrt{x^2 + y^2}$.

operation reduces the strength of each item vector in the chunk [Murdock, 1992], which may have undesirable consequences. Consequently, unlike the simpler CADAM model, TODAM2 does not seem to be biologically plausible.

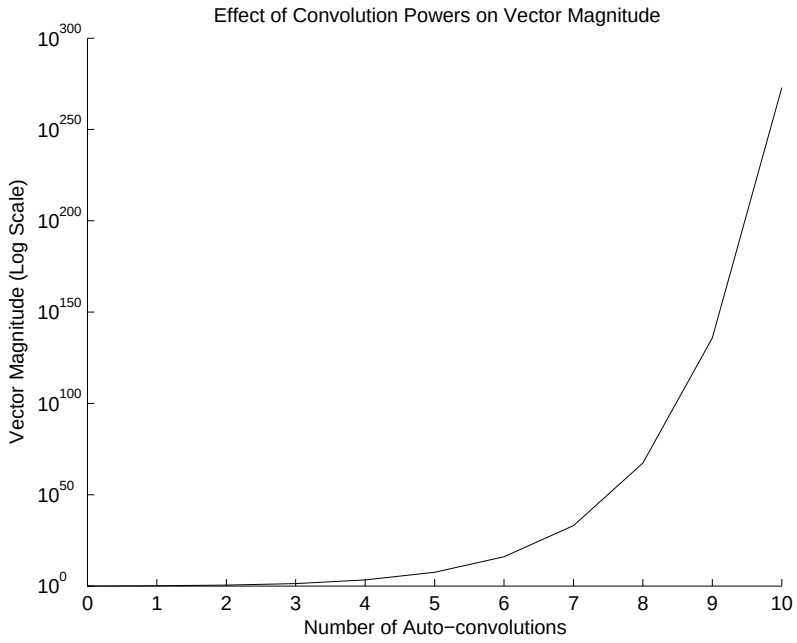

Figure 2.5: An illustration of the effect of computing the auto-convolution on the size of the vector magnitude. In the simulation, the starting vector used was a randomly chosen vector with a magnitude of 1, and the auto-convolution operation was performed a total of 10 times. The magnitude of the vector after 10 auto-convolutions was recorded to be 8.46×10^{272}. Note that the y-axis of the graph is in a logarithmic scale.

The TODAM2 and chunking model also have difficulty performing some tasks that humans are able to do. As with the chaining model, because the chunking model does not explicitly encode the order information, it also has difficulties performing the positional probe task. Traversing the memory trace backwards (i.e. backwards recall) is also an issue with the chunking model. Since the chunking model requires previously recalled items to recall subsequent items, recalling the last item in the list requires the retrieval of every item in the list. Consequently, the chunking model is only able to perform backwards recall by doing multiple forward recall passes to retrieve each item. This does not mean that the chunking model fails completely at backwards recall though, as the method by which humans perform backwards recall is still not entirely known.

Murdock notes in his 1995 article on TODAM2 that the chunking model fails to reliably reproduce the fundamental recency effect seen in human data. Figure 2.6 shows that a weak recency effect is seen for longer lists, but not for the shorter lists, which is contrary to the human data. In that same paper, the author also states that the transpositional errors generated by the

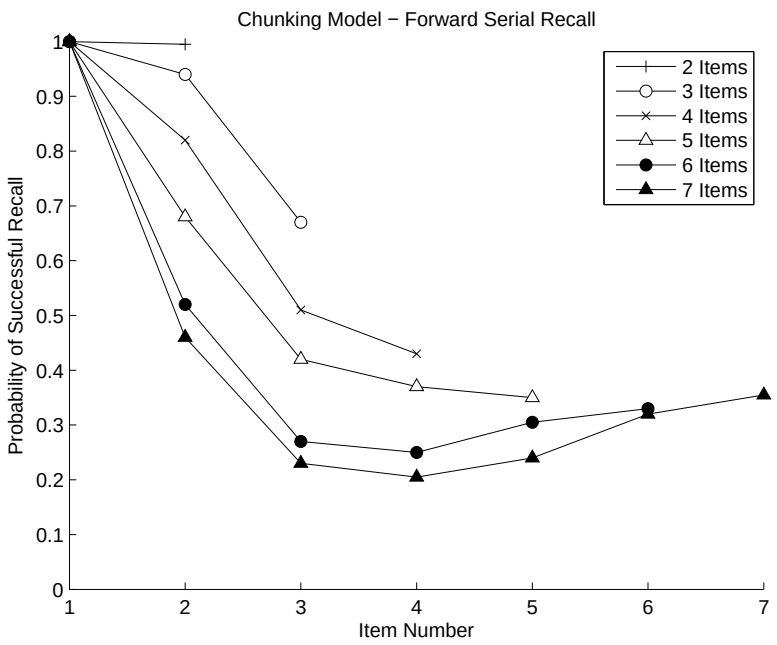

Figure 2.6: Recall accuracies of the chunking model for lists with varying lengths of 2 to 7 items. From [Murdock, 1995, Figure 3].

chunking model was "consistently wrong". Lastly, the model has not been as rigorously tested as the original TODAM model, so claims about the accuracy of the model compared to human data cannot be substantiated.

Chapter 3

The Ordinal Serial Encoding Model

The Ordinal Serial Encoding (OSE) model is designed from the onset with biological plausibility in mind. Consequently, human neuro-imaging data as well as neural recording and behavioural data collected from macaque monkey studies has been used to design the model. The monkey data was used to get insight into how a more primitive brain encodes serial order information, to see if it can provide clues on how the human brain performs the same task. This chapter introduces the vector symbolic architecture used by the OSE model, as well as the basic structure and encoding concepts behind the OSE model.

3.1 The Holographic Reduced Representation

The Holographic Reduced Representation (HRR) is a vector symbolic architecture developed by Plate (2003). It has several unique properties that make it suitable for use in a neurobiologically plausible system. Before discussing its suitability for neural implementation, the three necessary operations required for any vector symbolic architecture – the binding, merging, and unbinding operations – are presented. In addition to these operations, the representational assumptions of the HRR, as well as a fourth operation, called the cleanup operation, are described.

3.1.1 Item Representation

In the HRR, items and concepts are represented using vectors where each vector element is a number chosen from a normal distribution with a mean of 0 and a variance of $1/n$, where n is the dimensionality of the vector. Choosing the vector elements from this distribution ensures that the magnitude of the vector is approximately 1.

3.1.2 Merging

The merging operation used in the HRR is identical to the merging operation discussed in Section 2.1.1, which is the vector superposition operation.

3.1.3 Binding

The previous section revealed one of the major flaws in the VSA employed by Murdock, and that is the fact that the dimensionality of the vectors used in the VSA roughly doubles for each convolution operation made. For a neurobiological system to implement an operation with such a feature, it would either require the system to sacrifice accuracy, by keeping the same number of neurons, but using fewer neurons to represent each dimension; or to sacrifice efficiency, by recruiting more neurons to represent the additional dimensions as the dimensionality grows. The system could also truncate the result of the convolution, but this would mean that information is lost for every convolution operation. All of these options have undesirable outcomes, so another binding method is needed. The HRR circumvents these problems by utilizing circular convolution as its binding operation.

Circular convolution is almost exactly like convolution, except that the circular convolution function considers the vector operands as circular (or wrapped). Figure 3.1 illustrates this concept and the differences between circular convolution and regular convolution. In this paper, the circular convolution operation is represented using this ($\circledast$) symbol. Mathematically, the circular convolution operation is defined by the following summation.

$$\begin{aligned} &\text{If } \mathbf{R} = \mathbf{X} \circledast \mathbf{Y}, \text{ and } \mathbf{R} = [r_0, r_1, \ldots, r_{n-1}],^{5} \text{ then} \\ &r_j = \sum_{k=0}^{n-1} x_k y_{j-k}, \qquad \text{for } j = 0 \text{ to } n-1, \\ &\text{(where the subscripts are modulo-}n\text{)} \end{aligned} \tag{3.1}$$

The circular convolution operation can be implemented using the discrete Fourier Transform (DFT), the inverse discrete Fourier Transform functions (IDFT) and an element-wise multiplication operation. These functions have the advantage of decreasing the computational complexity

[5] The remaining equations in this section use this variable naming convention as well.

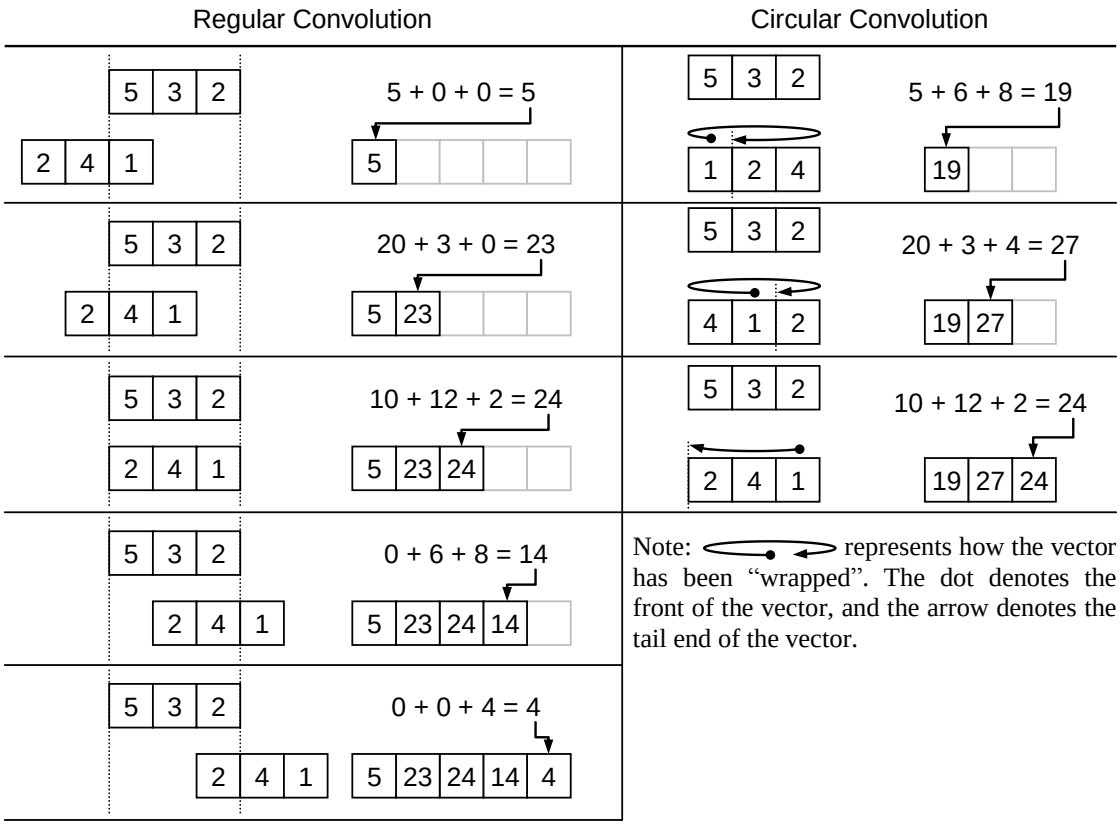

Figure 3.1: A visual illustration of how convolution and circular convolution works. In regular convolution, one vector "slides" past the other and the result of the operation has a $(2n-1)$ dimensions, where n is the dimensionality of the original vectors. In circular convolution however, the vectors are circularized, and then one vector is "rotated" in place. This results in no increase in the dimensionality of the result. In the figure, each line represents a step in the convolution operation. The vectors used in this example are $[5,3,2]$ and $[1,4,2]$. The first step in both convolution operations is to flip the second vector. Next, the element-wise multiplication is performed between the first and second vector (shown for each step in the figure), and this result is used as an element in the convolved vector.

of the circular convolution operation.[6] The DFT ($\mathcal{F}$) and IDFT ($\mathcal{F}^{-1}$) are defined as:

$$\text{If } \mathbf{F} = \mathcal{F}(\mathbf{X}), \qquad f_j = \sum_{k=0}^{n-1} x_k e^{-\frac{(2\pi i)}{n} jk} \tag{3.2}$$

$$\text{If } \mathbf{X} = \mathcal{F}^{-1}(\mathbf{F}), \qquad x_j = \frac{1}{n} \sum_{k=0}^{n-1} f_k e^{\frac{(2\pi i)}{n} jk} \tag{3.3}$$

where $i = \sqrt{-1}$, and n is the dimensionality of the input vector. The circular convolution function can be computed with the DFT and IDFT functions, which essentially transforms the vectors into frequency space, performs an element-wise multiplication, and then transforms the result back into vector space. In other words:

$$\mathbf{X} \circledast \mathbf{Y} = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{X}) \odot \mathcal{F}(\mathbf{Y})), \tag{3.4}$$

where $\odot$ represents the element-wise multiplication operation. It is important to note that the DFT and IDFT function are linear operations, which make them easy to implement neurally. Section 4.3.1 discusses this further.

The use of the Fourier Transform to calculate the circular convolution also demonstrates the advantage of using HRR vectors. Figure 3.2 compares the Fourier coefficients of an HRR vector, and a vector comprised of elements chosen from a random uniform distribution. As the figure shows, over multiple runs the coefficients for the HRR vector are relatively uniform, whereas the coefficients for the random uniform vector consists of a large value for the first coefficient and small values for the rest. From a network implementation standpoint, this means that if a neural network is used to implement the circular convolution operation, for an HRR vector, each node in the network can be configured to operate within a specified range. However, if the random uniform vector is used, the nodes in the network would have to operate within a much larger range, much of which is unused, reducing the efficiency and accuracy of the network.

3.1.4 The Identity Vector

Because circular convolution is used as the binding operation in the HRR, the HRR identity vector must be the circular form of the identity vector used with the regular-convolution binding operation. This vector turns out to be a vector with 1 as the first element, and 0 for the rest of the elements ($I = [1, 0, 0, 0, \ldots]$). Because this vector also represents what is known as a delta vector and to avoid confusion with the identity vector mentioned in the previous section, this thesis will use the symbol ($\boldsymbol{\delta}$) to denote the HRR identity vector instead.

[6] Using Equation 3.1 to compute the circular convolution results in an $O(n^2)$ run-time complexity, whereas using the Fourier transforms result in a run-time complexity of $O(n \log n)$ [Plate, 2003].

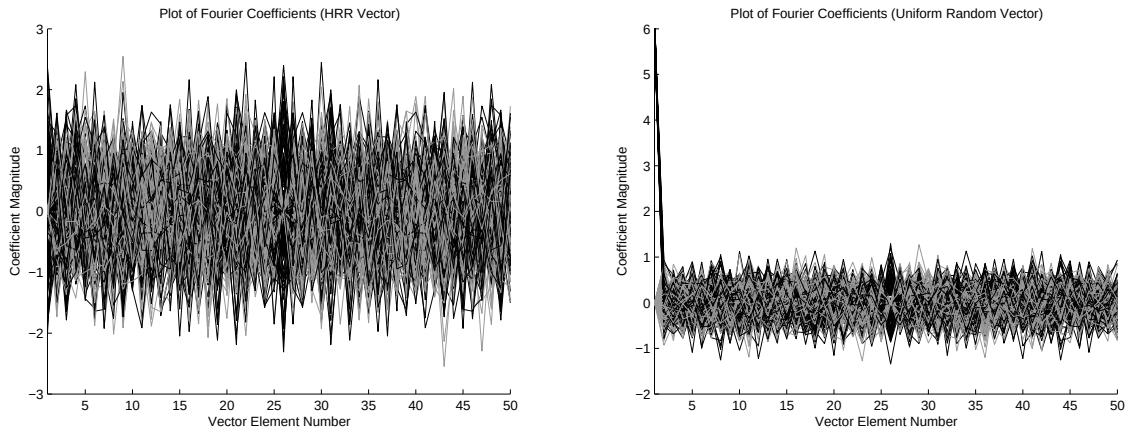

Figure 3.2: Plots of the Fourier coefficients for an HRR vector (left), and a uniform random vector (right). To generate these figures, a 50-dimensional vector was used in both cases, and the simulation was run for 100 iterations. For both graphs, the real-valued coefficients are plotted in black, and the imaginary-valued coefficients are plotted in grey.

3.1.5 Unbinding

There are several methods to perform unbinding in the HRR. One method is to use the circular version of the correlation function (recall that Murdock utilizes the correlation function to perform unbinding in his models). The circular correlation function (represented by the $\circledast$ symbol) is defined as follows.

$$\begin{aligned} &\text{If } \mathbf{X} = \mathbf{R} \circledast \mathbf{Y} \\ &x_j = \sum_{k=0}^{n-1} r_k y_{j+k}, \qquad \text{for } j = 0 \text{ to } n-1, \\ &\text{(where the subscripts are modulo-}n\text{)} \end{aligned} \tag{3.5}$$

However, since the circular correlation function is neither associative nor commutative, it is generally easier to work with the vector inverse instead. It is possible to calculate the inverse of a vector such that binding a vector with its inverse results in the delta vector. Using this property, the inverse of a vector can be calculated as shown below.

$$\begin{aligned} \mathbf{X} \circledast \mathbf{X}^{-1} &= \boldsymbol{\delta} \\ \mathcal{F}^{-1}(\mathcal{F}(\mathbf{X}) \odot \mathcal{F}(\mathbf{X}^{-1})) &= \boldsymbol{\delta} \\ \mathcal{F}(\mathbf{X}) \odot \mathcal{F}(\mathbf{X}^{-1}) &= \mathcal{F}(\boldsymbol{\delta}) \\ \mathcal{F}(\mathbf{X}^{-1}) &= \mathcal{F}(\boldsymbol{\delta}) \oslash \mathcal{F}(\mathbf{X}) \\ \mathbf{X}^{-1} &= \mathcal{F}^{-1}(\mathcal{F}(\boldsymbol{\delta}) \oslash \mathcal{F}(\mathbf{X})), \end{aligned} \tag{3.6}$$

where the $\oslash$ symbol denotes the element-wise division of two vectors. The division operation poses a problem, however, when the Fourier Transform yields vector elements that are close to zero. When this is the case, taking the reciprocal of the vector elements results in very large numbers, which is not ideal for a system with a finite representational capacity. In addition to this, Plate (2003) shows that using the exact inverse on memory traces that are noisy or that are the result of the superposition of other vectors results in more noise in the retrieved vectors.

Rather than using the exact inverse, an approximate inverse function is used. The approximate inverse is calculated using the involution function, which has already been seen before. However, as with all the functions used in the HRR, the involution function used in the HRR has to be modified for use with circular operations. The involution is thus defined as:

$$\text{If } \mathbf{Y} = \mathbf{X}', \text{ then, } y_j = x_{(-j \bmod n)} \tag{3.7}$$

where $\mathbf{X}'$ is used to denote the approximate inverse of the vector $\mathbf{X}$ calculated using the involution function.[7] To visualize what the involution function does, for a given vector $\mathbf{X} = [x_0, x_1, \ldots, x_{n-1}]$, the involution of $\mathbf{X}$ would be $[x_0, x_{n-1}, x_{n-2}, \ldots, x_1]$, which is a linear operation on the original vector $\mathbf{X}$.

3.1.6 Cleanup or Deblurring

As seen in the previous section, the exact inverse operation is expensive to calculate exactly, and is not robust to noisy inputs. As such, the approximate inverse is usually preferred in the unbinding process. Additionally, the circular convolution operation can be thought of as a lossy compression operation and information is typically lost with each application of the binding operation. These factors introduce errors in the results of the unbinding process and consequently, the unbound vector does not always exactly equal the desired vector, but only approximates it. Thus, for the holographic reduced representation, a fourth operation – the cleanup operation – is needed to complete the set of necessary operations. The job of the cleanup operation is then to compare this approximation to a known vocabulary of vectors and to choose the best match from the vocabulary. If the desired vector is contained within the cleanup memory, then the cleanup process will most likely select that vector as the best match to the approximate.

3.2 Memory Trace Encoding in the OSE Model

The OSE model takes inspiration from behavioural data collected from macaque monkeys that indicate that the monkeys categorize items using order information [Orlov et al., 2000]. To perform this type of encoding using the HRR, vectors are randomly-generated to represent the item's

[7] Different notations are used here to differentiate the approximate inverse operation ($\mathbf{X}'$) from the exact inverse operation ($\mathbf{X}^{-1}$).

order information and then bound to the vector representing the item itself. Using such a scheme, the list $[\mathbf{A}, \mathbf{B}, \mathbf{C}]$ would be encoded as:

$$\mathbf{M}_{ABC} = \mathbf{A} \circledast \mathbf{P}_A + \mathbf{B} \circledast \mathbf{P}_B + \mathbf{C} \circledast \mathbf{P}_C, \tag{3.8}$$

where $\mathbf{P}_A$, $\mathbf{P}_A$, and $\mathbf{P}_A$ are the vectors used to indicate the positional information for the items $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ respectively.

While past serial memory models have attempted to address all of the memory phenomena associated with list memory using their encoding and decoding schemes, the approach taken here relies on taking a broader view of the cognitive architecture. As such, it draws on work on general features of memory to account for some of the specific list-related memory effects. To begin, the OSE embraces the strong evidence for a dual-store model of memory. In the general dual-store concept, two memory components – a short term memory (STM), and a longer term memory (LTM) – are used to explain the origin of the primacy and recency effects seen in serial recall data. The dual-store theory was first proposed by Atkinson and Shiffrin (1968), and recent fMRI studies lend support to this theory [Talmi et al., 2005]. The dual-store model proposes a short-term store in which memory traces need to be actively maintained (through rehearsal, for example) in order to be remembered, and a long-term store in which memory traces are stored through more permanent changes in the neural network, reducing the requirement for active maintenance. Because the OSE models components from the human memory system, it utilizes the short-term store concept from the dual-store model, and replaces the long-term store with an medium-term store that acts like an actively maintained "gateway" to longer-term memory, similar to the working memory model proposed by Baddeley and Hitch (1974), and later expanded by Baddeley (2000).

Baddeley's working memory model comprises four main components: a visuo-spatial sketchpad, a phonological loop, an episodic buffer, and a central executive. The short-term memory store in the OSE functions like the phonological loop and visuo-spatial sketchpad, which temporarily store information crucial to whatever task the subject is doing at the time, and decays if not consistently refreshed. Because of the decay, the longer-term memory store is needed to remember information that while relevant to the current task, needs to be held in memory for a longer period of time than the decay allows. In this way, the longer-term memory store functions like the episodic buffer in Baddeley's working memory model [Baddeley, 2000]. Figure 3.3 illustrates the similarities and differences between the dual-store model, OSE model, and Baddeley's working memory model.

Because of the way they function in the OSE model, the short-term memory store is referred to as the "input buffer" component and the longer-term memory store is referred to as the "episodic buffer" component. Both memory components encode lists in the same manner as the ordinal encoding scheme mentioned earlier, however each one is configured slightly differently to mimic the behaviour of a dual-store model. As mentioned earlier, the input buffer component is designed to decay slowly, and this reproduces the recency effect because more recent items will have less

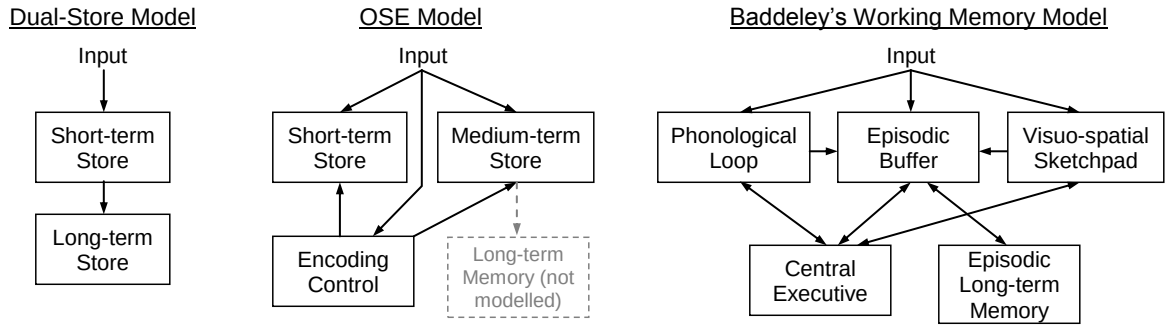

Figure 3.3: A comparison between the dual-store model [Atkinson and Shiffrin, 1968] (left), the Ordinal Serial Encoding model (middle), and Baddeley's working memory model [Baddeley, 2000] (right).

decay and will thus contribute more to the overall memory trace. The encoding equation for the input buffer component is thus:

$$\mathbf{M}_i^{\text{in}} = \gamma \mathbf{M}_{i-1}^{\text{in}} + (P_i \circledast I_i), \tag{3.9}$$

where $\mathbf{M}^{\text{in}}$ represents the memory trace stored by the input buffer and γ is the rate of decay of the old memory trace. To achieve decay, the value assigned to γ must be less than 1. Data fitting for this parameter will be discussed in a later section.

Since the input buffer component of the OSE model only contributes to the recency in the recall data, the episodic buffer component of the OSE model must account for primacy effects. This is done by introducing a primacy gradient in the encoding scheme for the episodic buffer component, that functions to mimic the effect that rehearsal has on the memory system. The primacy gradient puts more emphasis on items that are closer to the start of the list, and this is achieved by applying a "decay" term that is greater than 1 to the old memory trace. Mathematically, the encoding equation for the episodic buffer component is as follows:

$$\mathbf{M}_i^{\text{epis}} = \rho \mathbf{M}_{i-1}^{\text{epis}} + (P_i \circledast I_i), \tag{3.10}$$

where $\mathbf{M}^{\text{epis}}$ represents the memory trace stored by the episodic buffer component, and ρ is the scaling factor used to achieve the primacy gradient. Comparing the cumulative effect of ρ on a single item in the memory trace to the number of rehearsals performed by humans in a list memory task [Rundus, 1971], as illustrated in Figure 3.4, shows a similar exponential curve.

The complete encoded memory trace produced by OSE model is then the combination of the memory trace generated by the input buffer component and the memory trace generated by the episodic buffer component.

$$\mathbf{M}^{\text{OSE}} = \mathbf{M}^{\text{epis}} + \mathbf{M}^{\text{in}} \tag{3.11}$$

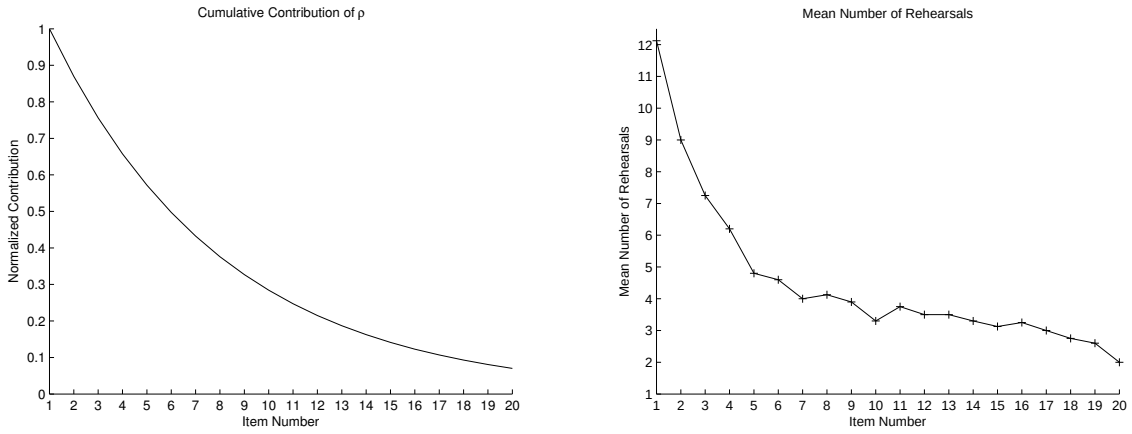

Figure 3.4: (Left) A plot of the cumulative contribution of ρ to an item in an OSE-encoded memory trace. The graph was computed using a value of $\rho = 1.2$. (Right) A plot of the mean number of rehearsals for human subjects in a free recall experiment. From [Rundus, 1971].

3.3 Recall in the OSE Model

Recall in the OSE model is very straightforward. Because each item is bound to its corresponding position vector, to retrieve an item in a specific position just requires convolving the memory trace with the inverse of the vector representing the desired item position in the list. In other words,

$$\mathbf{I}_i \approx \mathbf{M}^{\text{OSE}} \circledast \mathbf{P}'_i \tag{3.12}$$

The result of the unbinding operation is then fed through a cleanup memory module to choose the "best" item from a dictionary of possible item vectors. In the OSE model, the desirability of an item vector is calculated using the dot product operation, and the item vector that yields the highest dot product is chosen as the output of the cleanup memory module. A cutoff value of 0.3 is also applied to the cleanup memory to handle the situation in which none of the item vectors produce desirable results. In this scenario, none of the item vectors are chosen, and the retrieval is recorded as an omission.

Chapter 4

Implementing the OSE Model in Spiking Neurons

4.1 Neuron Basics

The human brain consists of roughly 10 to 100 billion neurons, all of which are connected to each other either directly or indirectly. The human brain can thus be considered a giant network, with each neuron representing a node within the network. Before we can design a network of neurons to perform some specified function, the basic component of the network (i.e. the neuron) must be understood.

The neurons used in the OSE model are known as large pyramidal neurons, and are typically found in the prefrontal cortex of human brains. This area of the brain associated with the working memory system [Goldman-rakic, 1995], and in particular, fMRI studies have shown the dorso-lateral prefrontal to be active during serial and free recall tasks [Henson et al., 2000]. Neurons can be considered like a self-enclosed processing unit with multiple inputs and in the case of large pyramidal neurons, one output. The inputs to a neuron are called dendrites, and the output of the neuron is called the axon. Neurons communicate with each other in the form of spikes, which travel down the axon of one neuron and arrives at the dendrites of the neuron which it is connected to. The point where the axon of one neuron meets the dendrite of another is known as the synapse. The arrival of a spike at a synapse causes a flow of electric charge (also called current) down the dendrite and into the neuron's cell body (also known as the soma). The soma sums all of the input current coming from all of the dendrites, and if it exceeds a certain threshold, causes a spike to be sent down the axon of the neuron. Figure 4.1 shows what a typical large pyramidal cell looks like and also illustrates the process of spike generation. A more detailed description of the cellular mechanisms that generate a spike is provided in Section 4.1.1.

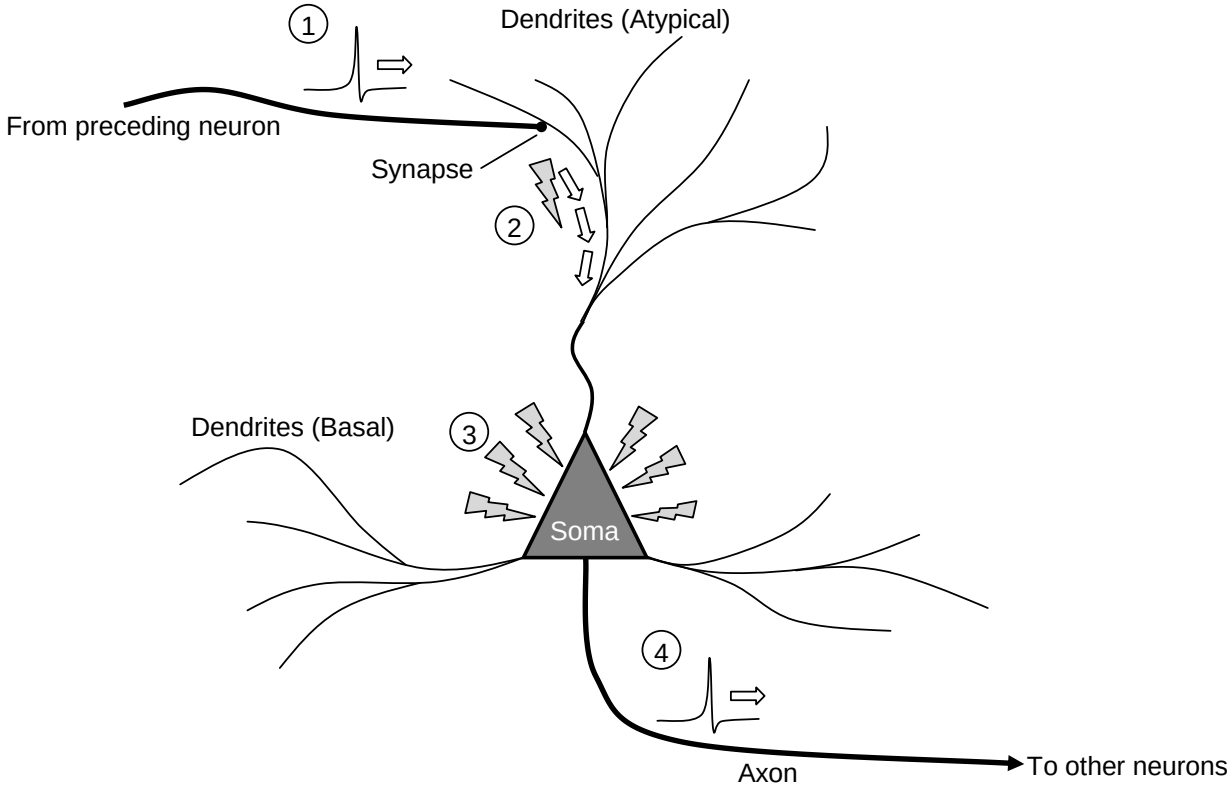

Figure 4.1: An simple schematic of a large pyramidal neuron detailing the dendrites and axon. The illustration also shows the effects of receiving an incoming spike at the neuron's dendrites.
① Spike from preceding neuron travels down its axon and arrives at the synapse.
② Dendrites to send current to the soma.
③ Membrane potential increases in the soma.
④ A spike is sent down the axon.

4.1.1 Generating a Spike

As mentioned in the previous section, the arrival of a spike at a synapse causes current to flow down the dendrite and into the soma. This current flow is called the postsynaptic current or PSC. When a spike arrives at a synapse, the PSC peaks, and then exponentially decays back to zero, with a time constant of τ^{PSC}. The shape and duration of the PSC depends on the neurotransmitters being utilized at the synapse.

The influx of current into the soma results in a buildup of electric charge within the cell body, increasing the difference in electric charge between the interior and exterior of the neuron. The difference in electric charge across the cell membrane, which is known as the membrane potential; is prevented from immediately equalizing by the cell membrane, the wall between the interior and

exterior of the cell. Instead, electric charge slowly leaks across the cell membrane, out of the cell. If the rate of current flow into the soma exceeds the rate at which current leaks from the cell, then current will build up in the cell, increasing the membrane potential (this is called depolarization). When the membrane potential exceeds a certain level; called the spiking threshold; gates in the cell membrane open, first causing more electric charge to quickly flow into the cell, and then a very short time after that, causes the electric charge to quickly exit the cell. Measuring the membrane potential when this process is happening produces a graph similar to the one shown in Figure 4.2. This rapid up and down movement in the membrane potential is called an action potential, or "spike" because of its characteristic shape. After the spike is generated, the gates in the membrane remain open for some time, called the refractory period, to allow the membrane potential to equalize. During this time, no spikes will be generated regardless of the magnitude of the input current provided to the neuron. After the refractory period however, if the input current is sufficient to drive the membrane potential past the threshold level, another spike will be generated, repeating the cycle of rapid depolarization and subsequent return to equilibrium of the membrane potential.

One must note that the initiation of a spike happens at what is known as the trigger zone, which is at the base of the cell body, where the axon originates. The electric charge that flows into this area of the cell during the rapid upswing of the action potential will disperse within the cell body, causing the membrane potential in the area slightly downstream of the trigger area to rise and then exceed the spiking threshold. Essentially, a spike originating in the trigger area will cause another spike slightly downstream of the trigger area, and like a row of dominoes, the spike will continue to propagate in this manner, travelling like a wave down the length of the axon.

4.1.2 Simulating Neurons

In the world of electronic circuitry, there is a simple circuit that is able to closely mimic the way the large pyramidal neurons collect and leak electric charge. This circuit is known as an "RC" or resistor-capacitor circuit in electronic circuits. If an RC circuit is provided with a constant input current, the capacitor in the circuit will slowly accumulate charge, and when the input current is taken away, the resistor will cause the accumulated charge in the capacitor to slowly leak away. The combination of the resistor and capacitor values determine the "RC time constant" (τ^{RC}), which is the rate at which electric charge is accumulated and dispersed.

The RC circuit forms the foundation of the "leaky-integrate-and-fire" (LIF) neuron model which has been demonstrated to simulate the spiking behaviour of the pyramidal neurons found in the human neocortex[8] [Rauch et al., 2003]. The RC circuit equations shown in Equation (4.1) are used in the LIF neuron model to determine the rate of accumulation and decay of electric

[8]Neocortex refers to the "new" parts of the brain. The term "neocortex" is used interchangeably with the term "cerebral cortex" which is on the "surface" of the brain. Older cortical structures are generally found deeper within the brain.

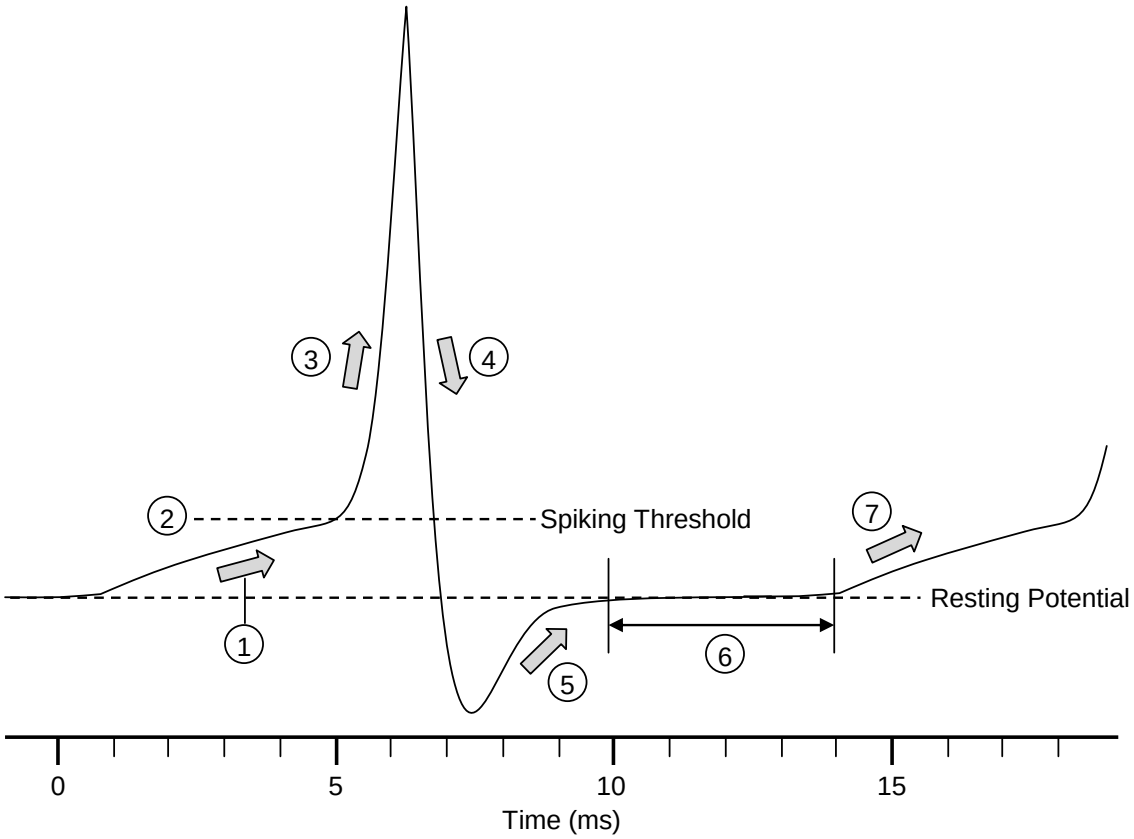

Figure 4.2: Illustration of the components of an action potential (spike) generated by a typical large pyramidal neuron. The graph shown is a plot of the membrane potential measured at the trigger zone of the neuron.
① Current from dendrites cause increase in membrane potential.
② Membrane potential exceeds spiking threshold.
③ Gates in the cell membrane open, causing a large influx of electric charge into the cell.
④ Followed by a large outflow of electric charge out of the cell.
⑤ Membrane potential returns to resting conditions.
⑥ Refractory period.
⑦ Cycle continues if neuron is being provided more input current.

charge in the neuron.

$$\begin{aligned} \tau^{RC} &= RC, \\ \frac{dV}{dt} &= -\frac{1}{\tau^{RC}}(V - J_{in}R), \end{aligned} \tag{4.1}$$

where R and C are the resistor and capacitor values of the RC circuit, $\frac{dV}{dt}$ is the rate of change of electric charge in the neuron, V is the membrane potential of the neuron at a moment in time, and J_{in} is the total amount of current flowing into the soma from the dendrites. When the membrane potential of the simulated neuron reaches the threshold level, a spike is "pasted" in, and the membrane potential is reset to its resting level. The model then fixes the membrane potential at the resting level for the duration of the refractory period (τ^{ref}), after which it is allowed to accumulate incoming electric charge again. Figure 4.3 illustrates the simulation of the LIF neuron when provided with a constant input current. The LIF neuron model is explored in greater detail in Eliasmith and Anderson (2003), where they describe the full RC circuit used, as well as discuss the advantages and disadvantages of the LIF neuron model.

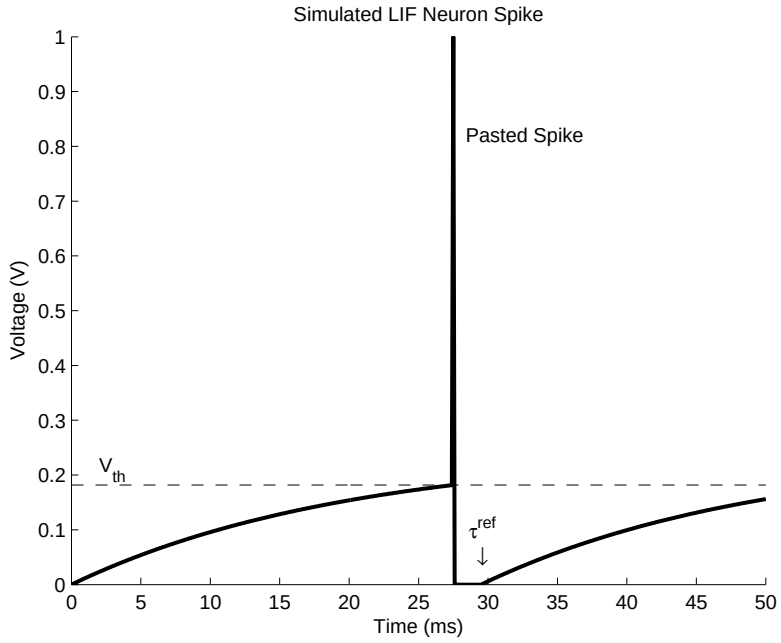

Figure 4.3: A simulation of the response of a leaky-integrate-and-fire (LIF) neuron given a constant input. The sub-threshold soma voltage curve was calculated using Equation 4.1. When the voltage exceeds the spiking threshold, V_{th}, a spike is inserted, and the voltage is returned to 0. The voltage is kept at 0 until a predefined period, τ^{ref}, has passed; after which, the voltage is allow to increase again.

4.2 The Neural Engineering Framework

The Neural Engineering Framework (NEF) is a systematic approach to designing neural networks and constrains any model built in the NEF to the underlying neurobiology. Unlike many other spiking neuron models which use neurons with identical responses, models built with the NEF intrinsically incorporate non-identical neuron properties as well as noise into the model. A spiking neuron network also allows for comparisons to be made to real-world spiking data in order to test the plausibility of the model.

Starting from the level of a single neuron, and working its way to the level of networks of populations of neurons, this section will describe how the NEF can be used to create networks to perform any arbitrary function needed.

4.2.1 Representation with Single Neurons

From the previous section, we have seen that neurons operate in terms of currents and electrical charges. In order to use neurons to represent physical values, the units in which the neurons operate must somehow be linked with the values we wish to represent. We start by assuming that there is a directly proportional relationship between the physical value x, and the total input current J. This relationship can be written as:

$$J(x) = \alpha x + J^{bias}, \tag{4.2}$$

where $J(x)$ is the input current as a function of the variable x, α is a scaling factor that converts x into the appropriate units used by the input current, and J^{bias} is a background current that results from the background firing of all the neurons connected to the neuron we are modelling.

By measuring the fire rate of the neuron while varying this input current, the response curve of the neuron can then be plotted. Figure 4.4 compares the response curves of neocortical neurons and the response curves generated using the LIF neuron model. For the LIF model, the neuron response curve can be calculated using the following equations:

$$a(x) = G[J(x)] = \begin{cases} \frac{1}{\tau^{ref} - \tau^{RC} \ln\left(1 - \frac{J^{th}}{J(x)}\right)}, & \text{if } J(x) > J^{th} \\ 0, & \text{otherwise} \end{cases} \tag{4.3}$$

where $a(x)$ is the activity of the neuron – reported in spikes per second – for the input value x; $G[\ldots]$ is the function that defines the non-linear response curve of the neuron to the input current $J(x)$; τ^{RC} and τ^{ref} are the RC and refractory time constants mentioned before; J^{th} is the threshold current above which the neuron will begin spiking, and is directly proportional to the spiking threshold level mentioned in the previous sections; and $J(x)$ is the total input current that was derived before.

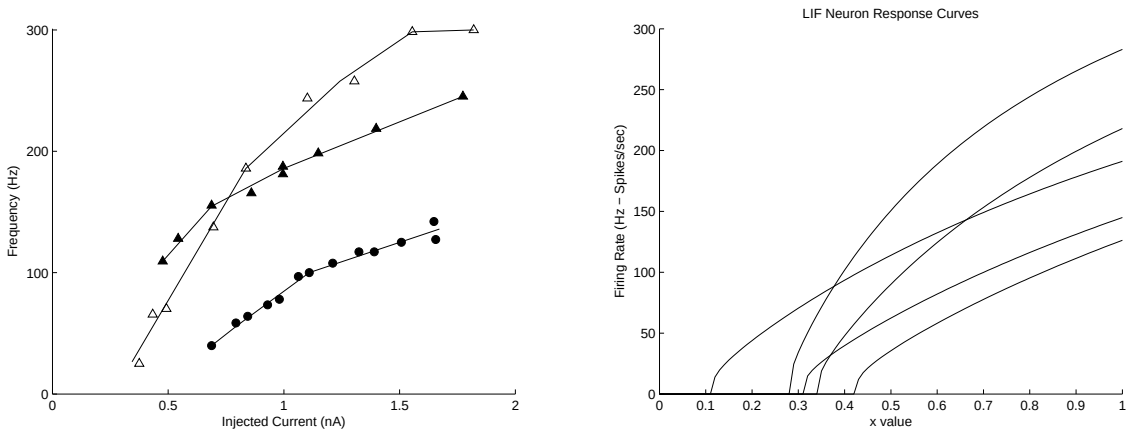

Figure 4.4: Example neuron response curves: (Left) Response curves plotted for regular-spiking neurons found in the guinea pig neocortex, from [McCormick et al., 1985]. (Right) Response curves generated by the LIF neuron model using Equation (4.3). Note that for this graph, the x-axis does not indicate the input current, but rather a value proportional to the input (injected) current.

One will notice that the units of $a(x)$ are given in spikes per second (Hz), which is not ideal. A scaling factor is thus needed to "decode" the activity information back into the appropriate units for x. We can then write the estimated representation of x as:

$$\hat{x} = a(x)\phi, \tag{4.4}$$

where $\hat{x}$ represents the estimate of the value of x, and ϕ is the scaling factor – known as the "optimal decoder" in the NEF – used to convert the activity of the neuron back into the units and scale of x.

Even with this decoder, a single neuron still does a horrible job at representing some given value x. As we have seen, a linear change in x results in a non-linear change in $a(x)$, and no amount of scaling will reverse this non-linearity. In order to compensate for the non-linearity, additional neurons need to be recruited.

4.2.2 Representation using Populations of Neurons

The reason that a better reconstruction of a physical value is achieved by employing more neurons can be understood using the following analogy. Suppose we wanted to represent an arbitrary number, for example 3.5763. If we had only one "neuron", it is equivalent to being able to use only multiples of 1 to reconstruct the number; and the best job we can do is either 3 or 4. If we had two "neurons", we could use not only multiples of 1, but multiples of 0.1 to reconstruct the number. With this, we can get 3.5, which is much better than either 3 or 4, but still not the

exact number. Continuing this pattern for multiples of 0.01, 0.001, and 0.0001, we see that for this example we would need a total of five or more "neurons" to accurately represent the number. This concept can also be applied to the neuron response curves. Now suppose we wanted to have the population of neurons output a value of x for any given value of x. Plotted on a graph, this is just the straight line with a slope of 1. Obviously, one neuron response curve is unable to fit that line very well, however, by using the weighted summation of multiple response curves, we can get a better fit to the line. This is illustrated in Figure 4.5. The reconstructed estimate of the input x using a population of n neurons can then be re-written as:

$$\hat{x} = \sum_{i=1}^{n} a_i(x)\phi_i, \tag{4.5}$$

where a_i and ϕ_i are the response curve and the decoder for the i^{th} neuron, respectively. The decoders needed to accurately reconstruct the input value can be calculated by:

$$\phi = \mathbf{\Gamma}^{-1}\mathbf{\Upsilon}, \qquad \text{where} \qquad \mathbf{\Gamma} = \mathbf{A}\mathbf{A}^T \qquad \text{and} \qquad \mathbf{\Upsilon} = \mathbf{A}\mathbf{X}^T \tag{4.6}$$

In the equation above, $\mathbf{A}$ is the population's activity matrix[9], and $\mathbf{X}$ is a vector of x values that the population needs to represent. A detailed derivation of the equations used to compute the decoders can be found in [Eliasmith and Anderson, 2003, Appendix A].

4.2.3 Representation in Higher Dimensions

In 1986, Georgopoulos et al. observed that neurons from the motor cortex of rhesus monkeys appeared to fire maximally when the monkey was commanded to move its arm in one of 8 spatial directions. He also observed that the direction the neurons preferred differed for different neurons. In spatial coordinates, each one of these preferred directions can be thought of as a three dimensional vector originating from the initial arm position and pointing toward the final arm position. Given its preferred direction vector, and the commanded direction vector, we can then deduce the activity of the neuron by calculating the similarity between the commanded direction vector and the preferred direction vector; and by assuming a linear relationship between the similarity measure and the resulting activity of the neuron. Knowing this, we can then modify the input current J to be proportional to this similarity measure instead of to some arbitrary value.

$$J(\bar{\mathbf{x}}) = \alpha(\boldsymbol{\varphi} \bullet \bar{\mathbf{x}}) + J^{bias} \tag{4.7}$$

In the equation above, the input current J is now a function of $\bar{\mathbf{x}}$, the commanded (input) direction vector. Instead of being directly proportional to the input vector $\bar{\mathbf{x}}$, the current is now proportional to the dot product ($\bullet$) of the preferred direction vector ($\boldsymbol{\varphi}$) and $\bar{\mathbf{x}}$. Note that the

[9]The activity matrix is a matrix that is constructed from the neuron response curves from all of the neurons in the neuronal population. Each row in this matrix corresponds to the neuron response curve of one neuron in the population of neurons.

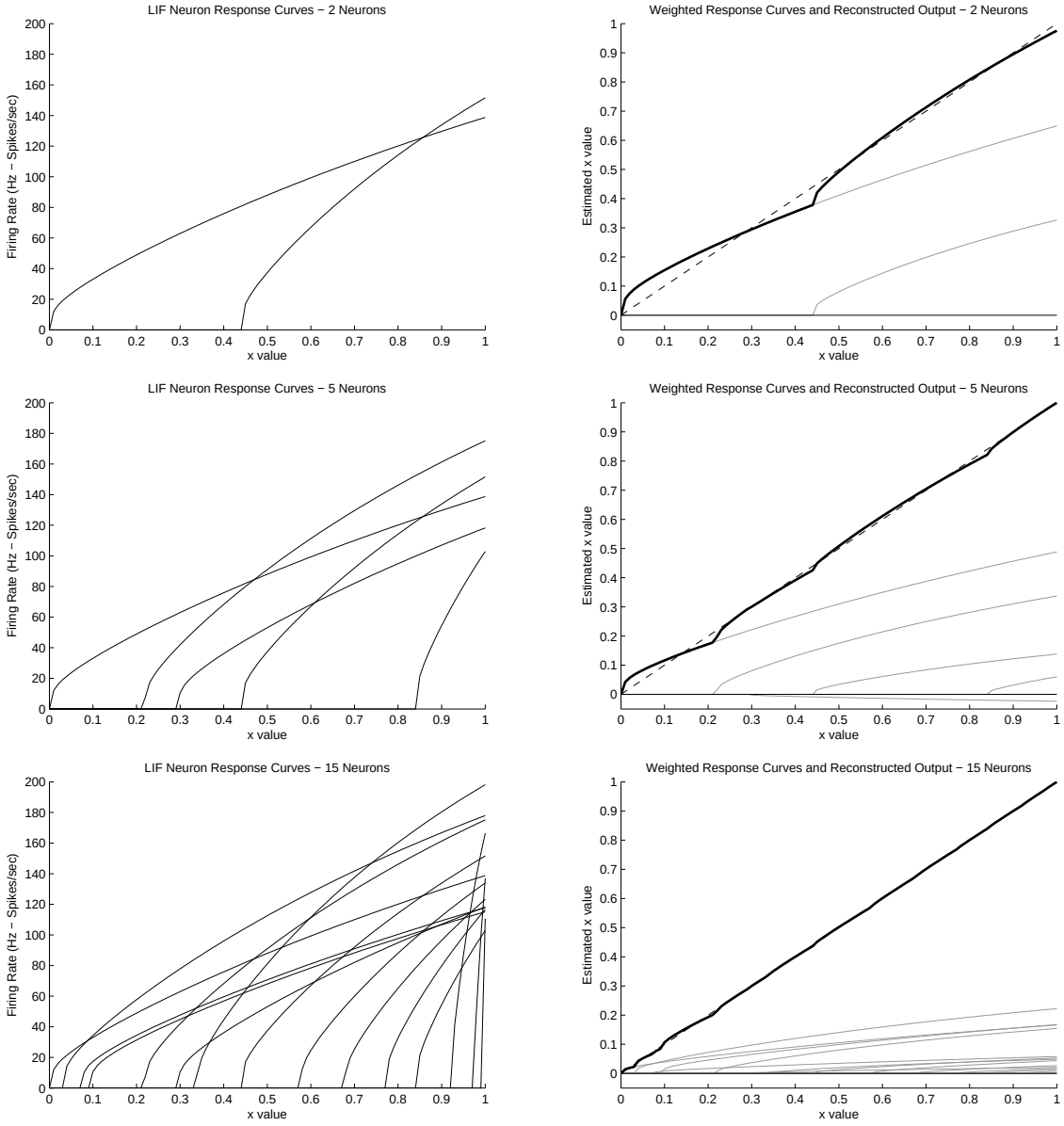

Figure 4.5: An illustration of how the response curves from multiple neurons can be appropriately weighted to estimate the input x value. In the left column, the original response curves for the neurons are shown. In the right column, the rescaled version of each response curve is shown (solid grey lines), in addition to the reconstructed x values (solid black line), and the ideal reconstruction of the x values (dashed line). The reconstructed x values are calculated by adding together the weighted response curves. Note that the response curves that lie below the zero line are response curves that have been scaled by a negative amount. Going from top to bottom, it is observed that as the number of neurons increases (from 2 to 5 to 15), the reconstructed estimate x improves.

original input current $J(x)$ was a scalar value. Additionally, because of the dot product, which produces a scalar value as well, the modified input current $J(\bar{\mathbf{x}})$ is also a scalar value. This means that it can be directly substituted into the neuron response function without needing to make any additional changes:

$$a(\bar{\mathbf{x}}) = G[J(\bar{\mathbf{x}})] = \begin{cases} \frac{1}{\tau^{ref} - \tau^{RC} \ln\left(1 - \frac{J^{threshold}}{J(\bar{\mathbf{x}})}\right)}, & \text{if } J(\bar{\mathbf{x}}) > J^{threshold} \\ 0, & \text{otherwise} \end{cases} \tag{4.8}$$

Reconstructing an estimate of the original input vector becomes slightly more complex because the result of the reconstruction should match the dimensionality of the input vector. Since the response curves of the neurons produce scalar values, the only way to accomplish this is by increasing the dimensionality of the decoders. Calculating the decoders remains the same as in the scalar case,

$$\boldsymbol{\phi} = \boldsymbol{\Gamma}^{-1}\boldsymbol{\Upsilon}, \qquad \text{where} \qquad \boldsymbol{\Gamma} = \mathbf{A}\mathbf{A}^T \qquad \text{and} \qquad \boldsymbol{\Upsilon} = \mathbf{A}\mathbf{X}^T, \tag{4.9}$$

with the exception that $\mathbf{X}$ is no longer a vector of x values that the population needs to represent, but rather a matrix of vectors that the population needs to represent. Performing the matrix calculation reveals that the dimensionality of the decoders do indeed match the dimensionality of the input vector. The reconstruction of the original input vector can then be computed as:

$$\hat{\mathbf{x}} = \sum_{i=1}^{n} a_i(\bar{\mathbf{x}})\boldsymbol{\phi}_i \tag{4.10}$$

As in the scalar case, $\hat{\mathbf{x}}$ is the reconstruction of the input vector, a_i is the i^{th} neuron's response to the $\bar{\mathbf{x}}$, and $\boldsymbol{\phi}_i$ is the i^{th} neuron's now multi-dimensional decoding vector.

4.2.4 Representation Over Time

The discussion thus far has been concerned about time-invariant values, be it scalar or multi-dimensional. However, we know that neurons do not operate solely with static values. Quite the contrary, neurons operate with signals that can vary over time. This implies that the framework that has been discussed so far must be extended into the temporal domain. The task of doing so is as simple as the scalar-to-multi-dimensional case and requires minimal changes.

To perform an analysis in the temporal domain, rather than using the static input $\bar{\mathbf{x}}$, the time-varying input $\bar{\mathbf{x}}(t)$ is used. This signal can be thought of as vector that changes its direction over time. Now, as in the vector case, the input current is:

$$J(\bar{\mathbf{x}}(t)) = \alpha(\boldsymbol{\varphi} \bullet \bar{\mathbf{x}}(t)) + J^{bias} \tag{4.11}$$

Note that in this case, $J(\bar{\mathbf{x}(t)})$ changes with time because $\bar{\mathbf{x}}(t)$ is a time-varying signal. Calculating

the response a of the neuron population becomes more complex because ideally, the response of the neurons should be in terms of spikes and the previous method of calculating the response only provided a static rate response rather than a response that changes over time. Using the mechanism described in Section 4.1.2 to generate the spiking behaviour, the individual spike response for each neuron in the population can be rewritten as:

$$a(\bar{\mathbf{x}}(t)) = \sum_s \delta(t - t_s), \tag{4.12}$$

where s is the number of spikes there are in the spike train, and $\delta(t - t_s)$ denotes the time of each spike. Note that the function $\delta(t)$ is similar to the HRR identity vector ($\boldsymbol{\delta}$) seen in Section 3.1.4. The function $\delta(t)$ is equal to 1 at $t = 0$ and 0 everywhere else, whereas in the $\boldsymbol{\delta}$ vector, the first vector element is 1 and the remaining vector elements are 0.

In order to reconstruct the input signal $\bar{\mathbf{x}}(t)$, or in this case, to determine how the downstream neuron is interpreting the outgoing spike trains of the entire population, the following equation is used.

$$\hat{\mathbf{x}}(t) = \left(\sum_{i=1}^{n} a_i(\bar{\mathbf{x}}(t)) \boldsymbol{\phi}_i \right) * h(t) \tag{4.13}$$

The equation above is very similar to the one seen in the multi-dimensional case, with a few modifications. Instead of a static response value, the spike trains for all the neurons in the population is used. The decoders $\boldsymbol{\phi}_i$, however, are still calculated in the same manner as before. The reason why it is possible to use decoders derived using the rate response values for a time-varying signal is explained further in [Eliasmith and Anderson, 2003]. Because the response of the neuron a_i is now in terms of spikes, multiplying the spike train by the decoders will only produce a scaled version of the spike train. In order to reconstruct the input signal, the scaled spike train is convolved with the post-synaptic current mentioned in Section 4.1.2. This is illustrated in the Figure 4.6. As a side note, the convolution with the PSC signal $h(t)$ can also be performed in the calculation of the neuron response function a. Conceptually, this is more truthful of the neurobiology because the current flowing to the soma of the downstream neuron is the sum of all the PSC's from the dendrites, and not the sum of all the spike trains. The neuron responses can then be rewritten as:

$$a(\bar{\mathbf{x}}(t)) = \sum_s \delta(t - t_s) * h(t) = \sum_s h(t - t_s), \tag{4.14}$$

and the reconstructed estimate as:

$$\hat{\mathbf{x}}(t) = \sum_{i=1}^{n} a_i(\bar{\mathbf{x}}(t)) \boldsymbol{\phi}_i \tag{4.15}$$

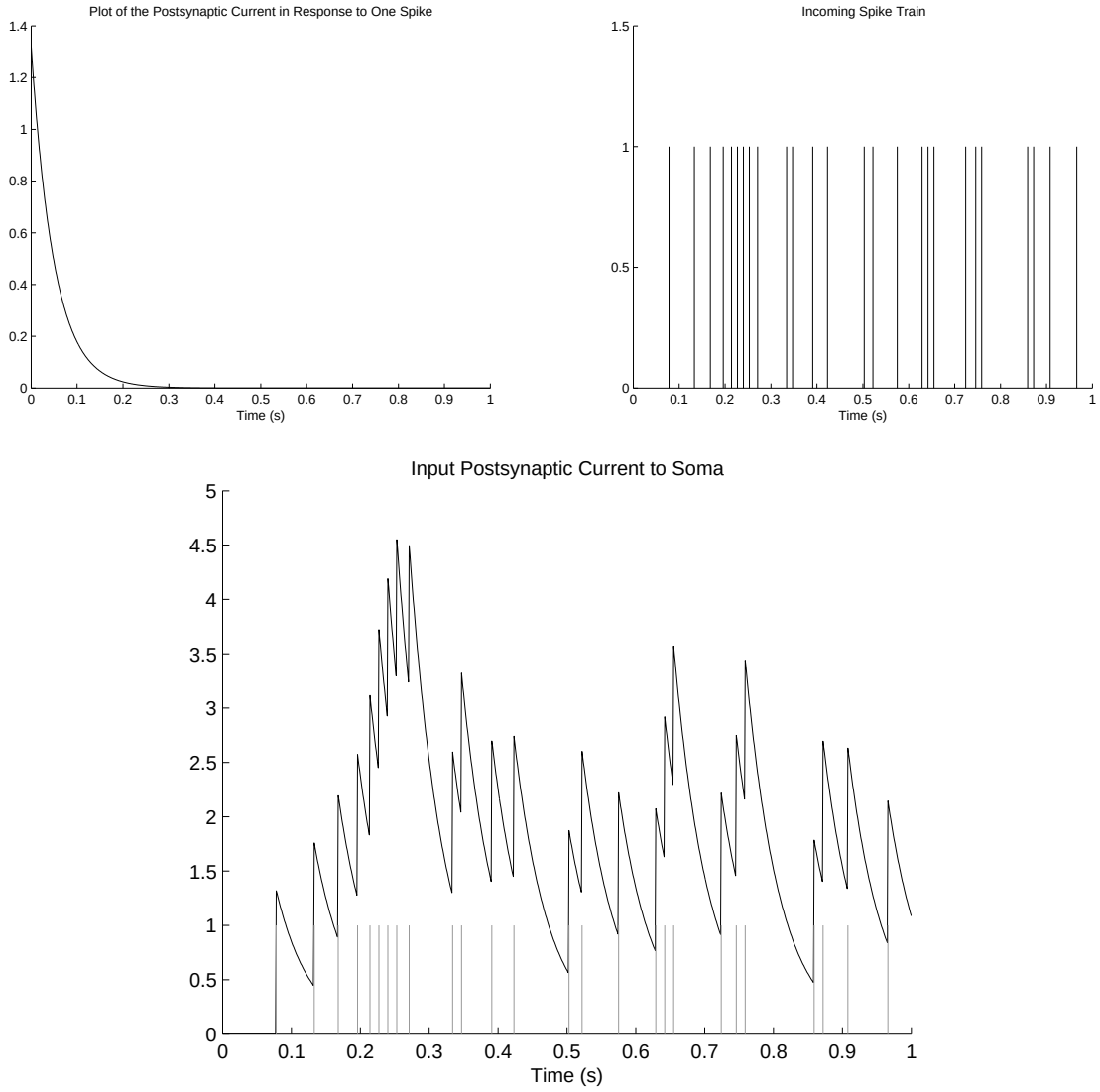

Figure 4.6: (Top Left) Plot of the shape of the PSC signal, $h(t)$, in response to one spike arriving at the synapse. (Top Right) The input spike train, $\sum_s \delta(t - t_s)$, used in this example. (Bottom) Plot of the total input current going into the soma, calculated using Equation (4.14). The input spike train is overlaid on this plot in grey.

4.2.5 Representation of Arbitrary Transformations

Thus far, the discussion has only considered reconstructing the original input signal (i.e. $\hat{x} \approx x$). While this has been valuable in figuring out the formulae necessary to perform representations in neurons, it does not provide many interesting applications. This section therefore discusses how the framework can be easily extended to perform any arbitrary transformation needed.

Consider a weighted linear combination of two signals: $\bar{\mathbf{z}} = C_1\bar{\mathbf{x}} + C_2\bar{\mathbf{y}}$.[10] To perform this transformation, three populations are needed, one for each variable, to be connected as shown in Figure 4.7.

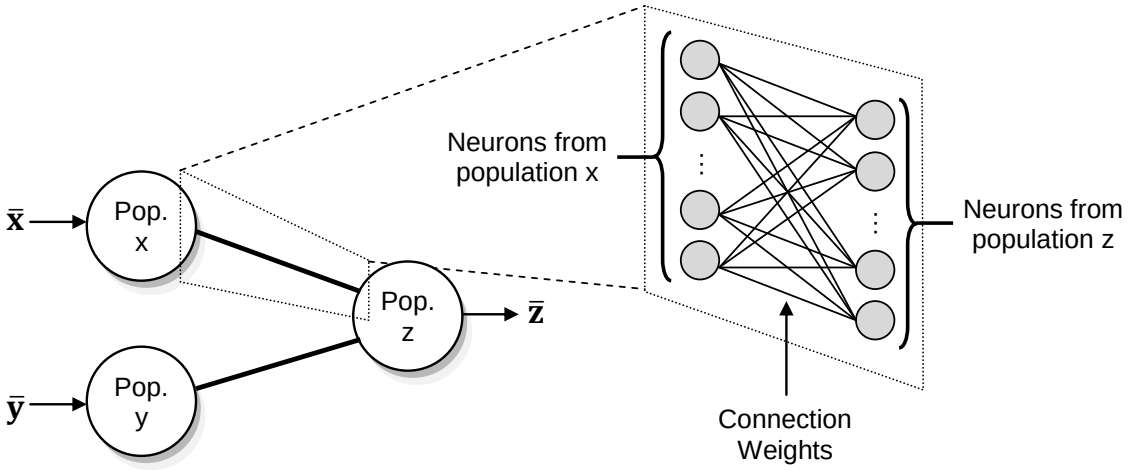

Figure 4.7: A connection diagram of the neural network needed to compute the transformation $\bar{\mathbf{z}} = C_1\bar{\mathbf{x}} + C_2\bar{\mathbf{y}}$. Each neuron in one population is fully connected with all of the neurons in the other population (shown in pop-up).

In the following analysis, a superscript letter will be used to denote these populations. For example, $a^{\mathbf{x}}$ will be used to denote the neuronal response of the population responsible for the signal $\bar{\mathbf{x}}$. As with the analysis in the previous sections, the input current for one neuron in the output population – in this case the "z" population – is first derived.

$$J^{\mathbf{z}}(C_1\bar{\mathbf{x}} + C_2\bar{\mathbf{y}}) = \alpha(\boldsymbol{\varphi}^{\mathbf{z}} \bullet (C_1\bar{\mathbf{x}} + C_2\bar{\mathbf{y}})) + J^{bias} \tag{4.16}$$

Since the outputs of the "x" and "y" populations would be the reconstructions of the inputs, $\hat{\mathbf{x}}$

[10] Note that while the variables $\bar{\mathbf{x}}$, $\bar{\mathbf{y}}$, and $\bar{\mathbf{z}}$ are listed as time-invariant values here, the same analysis can be easily performed with time-varying signals. See Section 4.2.4

and $\hat{\mathbf{y}}$ are substituted for $\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$, respectively. The equation for the input current then becomes:

$$\begin{aligned} J^{\mathbf{z}}(C_1\bar{\mathbf{x}} + C_2\bar{\mathbf{y}}) &= \alpha(\boldsymbol{\varphi}^{\mathbf{z}} \bullet (C_1\hat{\mathbf{x}} + C_2\hat{\mathbf{y}})) + J^{bias} \\ &= \alpha\left[\boldsymbol{\varphi}^{\mathbf{z}} \bullet \left(C_1 \sum_i a_i^{\mathbf{x}}(\bar{\mathbf{x}})\boldsymbol{\phi}_i^{\mathbf{x}} + C_2 \sum_j a_j^{\mathbf{y}}(\bar{\mathbf{y}})\boldsymbol{\phi}_j^{\mathbf{y}}\right)\right] + J^{bias} \\ &= \sum_i w_i^{\mathbf{x}} a_i^{\mathbf{x}}(\bar{\mathbf{x}}) + \sum_j w_j^{\mathbf{y}} a_j^{\mathbf{y}}(\bar{\mathbf{y}}) + J^{bias}, \end{aligned} \tag{4.17}$$

where $w_i^{\mathbf{x}} = \alpha C_1 (\boldsymbol{\varphi}^{\mathbf{z}} \bullet \boldsymbol{\phi}_i^{\mathbf{x}})$, and $w_j^{\mathbf{y}} = \alpha C_2 \left(\boldsymbol{\varphi}^{\mathbf{z}} \bullet \boldsymbol{\phi}_j^{\mathbf{y}}\right)$. In the general case, the scalar variables C_1 and C_2 may be substituted with the matrices $\mathbf{C}_1$ and $\mathbf{C}_2$. The weights $w_i^{\mathbf{x}}$ and $w_j^{\mathbf{y}}$ then become of the form $w_i^{\mathbf{x}} = \alpha\left((\boldsymbol{\varphi}^{\mathbf{z}}\mathbf{C}_1) \bullet \boldsymbol{\phi}_i^{\mathbf{x}}\right)$.

With the input current equation determined, it is then trivial to compute the estimate of the output $\bar{\mathbf{z}}$.

$$a^{\mathbf{z}}(C_1\bar{\mathbf{x}} + C_2\bar{\mathbf{y}}) = G\left[J^{\mathbf{z}}(C_1\bar{\mathbf{x}} + C_2\bar{\mathbf{y}})\right] \tag{4.18}$$

$$\hat{\mathbf{z}} = \sum_{k=1}^{n} a_k^{\mathbf{z}}(C_1\bar{\mathbf{x}} + C_2\bar{\mathbf{y}})\boldsymbol{\phi}_k^{\mathbf{z}} \tag{4.19}$$

Note that in Equation (4.18), the function $G[\ldots]$ is the non-linear neuron response as detailed in Equation (4.8), or for time-varying signals, a spike train, as in Equation (4.14).

4.2.6 Representation of Arbitrary Functions

Extending the derivations above for non-linear functions is also a trivial matter. In the next example, the neural implementation for the scalar non-linear function $z = x \cdot y$ is derived.[11] In previous examples, the decoders used to reconstruct the input signal has been derived to minimize the error between an input signal x and its reconstruction $\hat{x}$. Conceptually, with a sufficient quantity of neurons in a population, these decoders can also be derived to minimize the error between an input signal x and a function $f(x)$, x^2, for example. Figure 4.8 illustrates this concept. To calculate the appropriate decoders to estimate the function $f(x)$, Equation (4.6) is modified such that in the calculation of $\boldsymbol{\Upsilon}$, the values of the desired function applied to $\mathbf{X}$, $f(\mathbf{X})$, are used in place of $\mathbf{X}$:

$$\phi = \boldsymbol{\Gamma}^{-1}\boldsymbol{\Upsilon}, \qquad \text{where} \qquad \boldsymbol{\Gamma} = \mathbf{A}\mathbf{A}^T \qquad \text{and} \qquad \boldsymbol{\Upsilon} = \mathbf{A}f(\mathbf{X})^T \tag{4.20}$$

In order to use these new decoders, the inputs x and y need to be combined in such a way that one neuronal population has access to both at the same time. This is done by configuring the

[11] In this example, a scalar function is used only because of its simplicity. The same analysis, albeit more involved, can be performed for multi-dimensional functions.

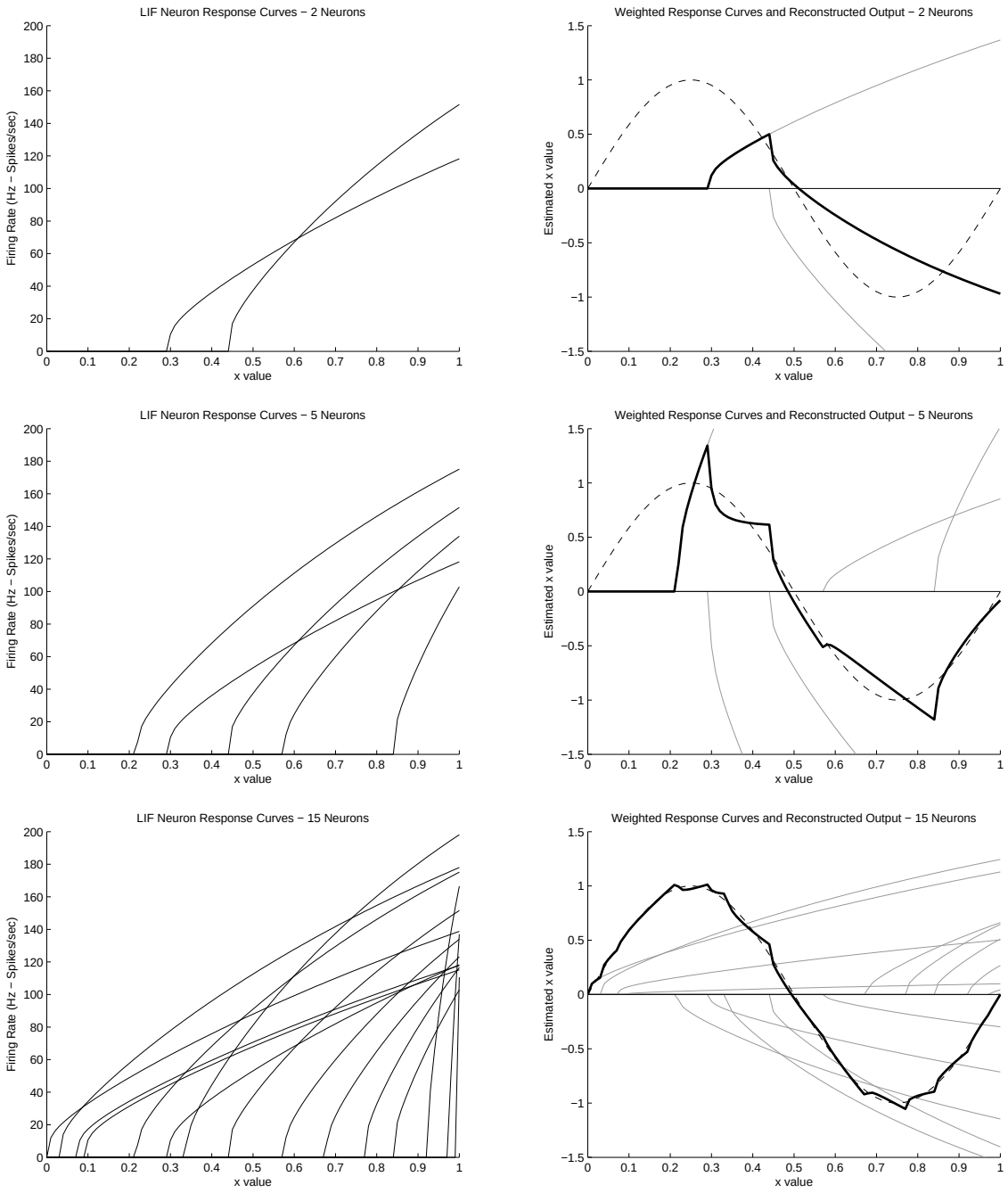

Figure 4.8: An illustration of how decoders can be used to weight the neuron response curves to reconstruct any desired function. The function used in this example is $\sin(2\pi x)$, and the ideal reconstruction of this function is denoted by the dotted line. The layout of this figure and the number of neurons used for each row are identical to Figure 4.5.

"z" population such that it takes the inputs x and y and converts them into a two dimensional vector:

$$\begin{aligned}\bar{\mathbf{m}} &= [x\ y] = [1\ 0]x + [0\ 1]y \\ &= C_1 x + C_2 y,\end{aligned} \tag{4.21}$$

where $C_1 = [1\ 0]$ and $C_2 = [0\ 1]$.

To implement the multiplication function $z = xy$, the output of the "z" population is then set to compute the function $z = (m_1 \cdot m_2)$, which is the multiplication of the first element of the vector $\bar{\mathbf{m}}$, m_1 (which has the value of x) with the second element of the vector $\bar{\mathbf{m}}$, m_2 (which has the value of y). This is done, as mentioned previously, by computing the decoders for the "z" population such that it minimizes the error for the desired function rather than just minimizing the error for the identity transformation (i.e. $x = x$) – See Equation (4.20).

Implementing the first transformation, $\bar{\mathbf{m}} = C_1 x + C_2 y$, has already been discussed, and using the formulas derived in Section 4.2.5, the estimated value of $f(\bar{\mathbf{m}}) = (m_1 \cdot m_2) = z$ can be calculated as follows:

$$\begin{aligned}a^{\mathbf{z}}(\bar{\mathbf{m}} = C_1 x + C_2 y) &= G\left[\sum_i w_i^{\mathbf{x}} a_i^{\mathbf{x}}(x) + \sum_j w_j^{\mathbf{y}} a_j^{\mathbf{y}}(y) + J^{bias}\right], \\ \widehat{f(\bar{\mathbf{m}})} &= \sum_{k=1}^{n} a_k^{\mathbf{z}}(\bar{\mathbf{m}} = C_1 x + C_2 y)\boldsymbol{\phi}_k^{f,\mathbf{z}},\end{aligned} \tag{4.22}$$

where $w_i^{\mathbf{x}} = \alpha\left(\boldsymbol{\varphi}^{\mathbf{z}}\mathbf{C}_1\phi_i^{\mathbf{x}}\right)$, $w_j^{\mathbf{y}} = \alpha\left(\boldsymbol{\varphi}^{\mathbf{z}}\mathbf{C}_2\phi_j^{\mathbf{x}}\right)$, $\widehat{f(\bar{\mathbf{m}})}$ is the estimate of the function $f(\bar{\mathbf{m}})$, and $\boldsymbol{\phi}_k^{f,\mathbf{z}}$ are the decoders that have been computed to minimize the representation error of the function $f(\bar{\mathbf{m}})$.

4.2.7 Relating the NEF to Biology

The discussion about the NEF has introduced abstract concepts such as encoders (φ) and decoders (ϕ), and one might get the misconception that these properties are intrinsic to each neuron. Looking at the neurobiology, neurons receive inputs in terms of current, and produce outputs in terms of spikes, both of which are scalar values. This makes it difficult to reconcile how the neurons would represent high dimensional vectors, or non-linear functions. It must be noted that individual neurons in isolation will not have the ability to represent such objects. Rather, it is the configuration of the connection weight matrix between neurons that enables the neurons to represent such objects. The NEF merely presents a simple way to calculate such a weight matrix to enable the neural population to perform a certain function, as has already been mentioned, albeit in passing only. Re-examining the input current derived for a linear transformation of inputs to a neural population, we see that the derivation itself gives rise to the connection weight

matrices, $w_i^{\mathbf{x}}$ and $w_j^{\mathbf{y}}$, needed to perform that linear transformation:

$$J^{\mathbf{z}}(C_1\bar{\mathbf{x}} + C_2\bar{\mathbf{y}}) = \sum_i w_i^{\mathbf{x}} a_i^{\mathbf{x}}(\bar{\mathbf{x}}) + \sum_j w_j^{\mathbf{y}} a_j^{\mathbf{y}}(\bar{\mathbf{y}}) + J^{bias},$$

$$\text{where} \qquad w_i^{\mathbf{x}} = \alpha C_1 \left(\boldsymbol{\varphi}^{\mathbf{z}} \bullet \boldsymbol{\phi}_i^{\mathbf{x}}\right), \text{ and } w_j^{\mathbf{y}} = \alpha C_2 \left(\boldsymbol{\varphi}^{\mathbf{z}} \bullet \boldsymbol{\phi}_j^{\mathbf{y}}\right)$$

In neurobiological terms, this connection weight matrix determines the strength of the connection – or how much effect a spike arriving at a synapse will have on the probability that the neuron will fire – between the neurons from both populations.

4.3 The OSE Model Implemented

The principles of the NEF make it possible to test the implementation of our mathematically defined system with a spiking neuron network. Using one neuronal population to implement the entire system is possible, but inefficient, and precludes the possibility of modularity. To increase the representational accuracy and computational efficiency of the system, it is broken down into several subsystems. The process of subsystem identification is performed for the OSE model below.

To start, the encoding and decoding (recall) equations for the OSE model are analyzed. As a reminder, the equations derived in Section 3.2 and Section 3.3 are re-stated below:

Encoding:

$$\mathbf{M}_i^{\text{in}} = \gamma \mathbf{M}_{i-1}^{\text{in}} + (P_i \circledast I_i),$$

$$\mathbf{M}_i^{\text{epis}} = \rho \mathbf{M}_{i-1}^{\text{epis}} + (P_i \circledast I_i),$$

$$\mathbf{M}^{\text{OSE}} = \mathbf{M}^{\text{epis}} + \mathbf{M}^{\text{in}}$$

Decoding:

$$\mathbf{I}_i = cleanup(\mathbf{M}^{\text{OSE}} \circledast \mathbf{P}_i')$$

Looking at the equations above, it is immediately apparent a subsystem will be needed to perform the circular convolution binding operation. Since the convolution operation is used in both the encoding and decoding equations, the implementation of the circular convolution subsystem can be reused in both the encoding and recall systems of the OSE model. The implementation of the involution function for the decoding system will be discussed with the implementation of the circular convolution subsystem. Additionally, because the equations require the use of the memory trace from the previous item encoding iteration, some sort of memory subsystem is needed. Lastly, a cleanup memory subsystem will be needed for the recall mechanism of the OSE model. The addition and scaling operations do not require their own subsystems because, as described in the previous sections, they can be easily implemented within the connection weights between

two neuron populations. To summarize the analysis of the encoding and decoding equations, the subsystems needed to implement the OSE model in a network of spiking neurons are:

- A subsystem to perform the binding operation.
- A subsystem to store the memory traces so that they can be utilized to encode subsequent list items.
- A subsystem to clean up the output of the recall mechanism of the OSE model.

With these subsystems defined, it is then possible to design a network that will implement the entire OSE model. The high-level network design of the OSE model is displayed in Figure 4.9.

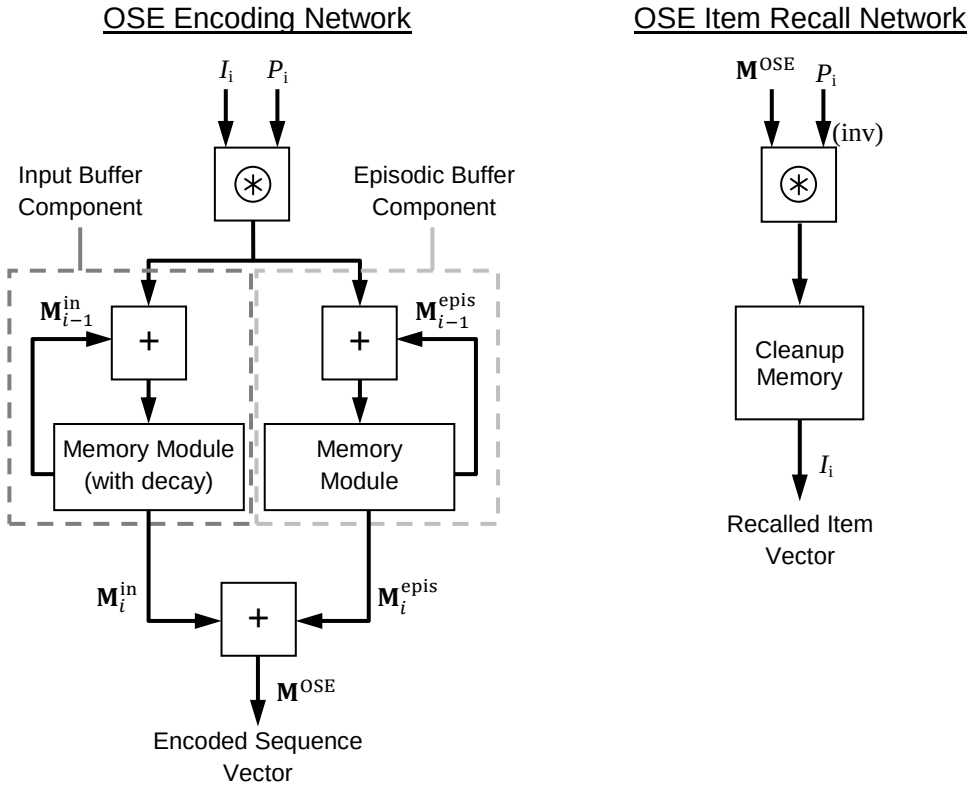

Figure 4.9: Network level diagrams of the OSE encoding network (left) and the OSE item recall network (right).

4.3.1 Binding and Unbinding

Performing the Binding Operation

As mentioned in Section 3.1.3, there are two methods by which the circular convolution can be computed. The first and most straightforward method is to use Equation (3.1). To implement this equation in a network of neurons, n^2 multiplication operations would need to be computed, where n is the dimensionality of the input vectors. This in turn implies that using this method, a total of n^2 interconnected populations of neurons will be needed to compute the circular convolution. However, as will be seen shortly, using the Fourier Transform method of calculating the circular convolution results in only an $O(n)$ increase in the number of multiplication operations needed as opposed to the $O(n^2)$ increase seen previously.

Computing the circular convolution using the Fourier Transform requires the implementation of the DFT and IDFT operations in neurons. Careful analysis of the DFT and IDFT operations (see Equations (3.2) and (3.3)), show that the transforms can be rewritten as the matrix multiplication of the input vector with an n-by-n matrix. For the DFT, this matrix is computed as follows:

$$\mathbf{F} = \mathcal{F}(\mathbf{X}) = \mathbf{X}\mathbf{W}, \text{ where } w_{jk} = e^{-\frac{(2\pi i)}{n}jk} \tag{4.23}$$

Likewise, the matrix for the IDFT is constructed as such:

$$\mathbf{X} = \mathcal{F}^{-1}(\mathbf{F}) = \mathbf{F}\mathbf{V}, \text{ where } v_{jk} = \frac{1}{n}e^{\frac{(2\pi i)}{n}jk} \tag{4.24}$$

Because the neurons can only represent physical values, the real and imaginary components of the DFT operation must be considered separately. In order to do this, the DFT and IDFT matrices are slightly modified:

$$\mathbf{W} = \mathbf{W}^{\mathbb{R}} + \mathbf{W}^{\mathbb{I}}i, \text{ where } w^{\mathbb{R}}_{jk} = \cos\left(-\frac{2\pi}{n}jk\right) \text{ and } w^{\mathbb{I}}_{jk} = \sin\left(-\frac{2\pi}{n}jk\right) \tag{4.25}$$

and

$$\mathbf{V} = \mathbf{V}^{\mathbb{R}} + \mathbf{V}^{\mathbb{I}}i, \text{ where } v^{\mathbb{R}}_{jk} = \frac{1}{n}\cos\left(\frac{2\pi}{n}jk\right) \text{ and } v^{\mathbb{I}}_{jk} = \frac{1}{n}\sin\left(\frac{2\pi}{n}jk\right) \tag{4.26}$$

Note that in the equation above, the superscript $\mathbb{R}$ denotes the real-valued component of the matrices, while the superscript $\mathbb{I}$ denotes the imaginary component of the matrices.

Using the DFT and IDFT matrices, the circular convolution operation can then be re-written

as follows:

$$\begin{aligned}
&\mathbf{A} \circledast \mathbf{B} = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{A}) \odot \mathcal{F}(\mathbf{B})) \\
&= \mathcal{F}^{-1}(\mathbf{A}(\mathbf{W}^{\mathbb{R}} + \mathbf{W}^{\mathbb{I}} i) \odot \mathbf{B}(\mathbf{W}^{\mathbb{R}} + \mathbf{W}^{\mathbb{I}} i)) && (4.27) \\
&= \mathcal{F}^{-1}(\mathbf{A}\mathbf{W}^{\mathbb{R}} \odot \mathbf{B}\mathbf{W}^{\mathbb{R}} + \mathbf{A}\mathbf{W}^{\mathbb{R}} \odot \mathbf{B}\mathbf{W}^{\mathbb{I}} i + \mathbf{A}\mathbf{W}^{\mathbb{I}} i \odot \mathbf{B}\mathbf{W}^{\mathbb{R}} + \mathbf{A}\mathbf{W}^{\mathbb{I}} i \odot \mathbf{B}\mathbf{W}^{\mathbb{I}} i) && (4.28) \\
&= (\mathbf{A}\mathbf{W}^{\mathbb{R}} \odot \mathbf{B}\mathbf{W}^{\mathbb{R}} + \mathbf{A}\mathbf{W}^{\mathbb{R}} \odot \mathbf{B}\mathbf{W}^{\mathbb{I}} i + \mathbf{A}\mathbf{W}^{\mathbb{I}} i \odot \mathbf{B}\mathbf{W}^{\mathbb{R}} - \mathbf{A}\mathbf{W}^{\mathbb{I}} \odot \mathbf{B}\mathbf{W}^{\mathbb{I}})(\mathbf{V}^{\mathbb{R}} + \mathbf{V}^{\mathbb{I}} i) && (4.29) \\
&= (\mathbf{A}\mathbf{W}^{\mathbb{R}} \odot \mathbf{B}\mathbf{W}^{\mathbb{R}})\mathbf{V}^{\mathbb{R}} + (\mathbf{A}\mathbf{W}^{\mathbb{R}} \odot \mathbf{B}\mathbf{W}^{\mathbb{I}})\mathbf{V}^{\mathbb{R}} i + (\mathbf{A}\mathbf{W}^{\mathbb{I}} \odot \mathbf{B}\mathbf{W}^{\mathbb{R}})\mathbf{V}^{\mathbb{R}} i - (\mathbf{A}\mathbf{W}^{\mathbb{I}} \odot \mathbf{B}\mathbf{W}^{\mathbb{I}})\mathbf{V}^{\mathbb{R}} + \\
&\quad (\mathbf{A}\mathbf{W}^{\mathbb{R}} \odot \mathbf{B}\mathbf{W}^{\mathbb{R}})\mathbf{V}^{\mathbb{I}} i - (\mathbf{A}\mathbf{W}^{\mathbb{R}} \odot \mathbf{B}\mathbf{W}^{\mathbb{I}})\mathbf{V}^{\mathbb{I}} - (\mathbf{A}\mathbf{W}^{\mathbb{I}} \odot \mathbf{B}\mathbf{W}^{\mathbb{R}})\mathbf{V}^{\mathbb{I}} - (\mathbf{A}\mathbf{W}^{\mathbb{I}} \odot \mathbf{B}\mathbf{W}^{\mathbb{I}})\mathbf{V}^{\mathbb{I}} i && (4.30)
\end{aligned}$$

Note that in the equations above, the property that $i \times i = -1$ has been used to simplify the equations. Although the final equation looks ungainly, the fact that the circular convolution always produces real-valued vectors can be used to further simplify the equation: because any term in the equation that produces an imaginary value can be removed. The final equation for the circular convolution operation after this simplification is then:

$$\mathbf{A} \circledast \mathbf{B} = (\mathbf{A}\mathbf{W}^{\mathbb{R}} \odot \mathbf{B}\mathbf{W}^{\mathbb{R}})\mathbf{V}^{\mathbb{R}} - (\mathbf{A}\mathbf{W}^{\mathbb{R}} \odot \mathbf{B}\mathbf{W}^{\mathbb{I}})\mathbf{V}^{\mathbb{I}} - (\mathbf{A}\mathbf{W}^{\mathbb{I}} \odot \mathbf{B}\mathbf{W}^{\mathbb{R}})\mathbf{V}^{\mathbb{I}} - (\mathbf{A}\mathbf{W}^{\mathbb{I}} \odot \mathbf{B}\mathbf{W}^{\mathbb{I}})\mathbf{V}^{\mathbb{R}} \quad (4.31)$$

Going back to the analysis of the number of multiplication operations needed, if each element-wise multiplication requires n multiplication operations, then the entire circular convolution operations requires at most $4n$ multiplication operations.

With this form of the circular convolution, the implementation in a neural population is also simplified. A total of 5 neural populations are needed, 4 populations to perform each one of the element-wise multiplications, and a fifth population to perform the IDFT operation. Each element-wise multiplication term is of the form $\mathbf{C}_1\mathbf{X}^T \odot \mathbf{C}_2\mathbf{Y}^T$, which is the vector equivalent of the form $C_1 x \times C_2 y$. The method described in Section 4.2.6 can then be used to determine the appropriate connection weights between the inputs and the element-wise multiplication populations. Likewise, the IDFT operation is a linear combination of the form $\mathbf{C}_1\mathbf{W}^T - \mathbf{C}_2\mathbf{X}^T - \mathbf{C}_3\mathbf{Y}^T - \mathbf{C}_4\mathbf{Z}^T$, and the procedures detailed in Section 4.2.5 are used to determine the connection weights between the multiplication and IDFT populations. Figure 4.10 illustrates how these populations are connected.

Further optimizations can be performed to increase the accuracy of the neural implementation of the circular convolution operation. First, because the vector elements are independent from each other, rather than performing the element-wise multiplication as a single function, each multiplication is performed separately. This eliminates the possibility of cross-talk between each dimension in the input vectors, which means that the result of the element-wise multiplication is more accurate. Second, a scaling factor can be applied to the input vectors to ensure that the result of the DFT operation remains between the optimal representation range of the neurons (which is preset to the range of -1 to 1).

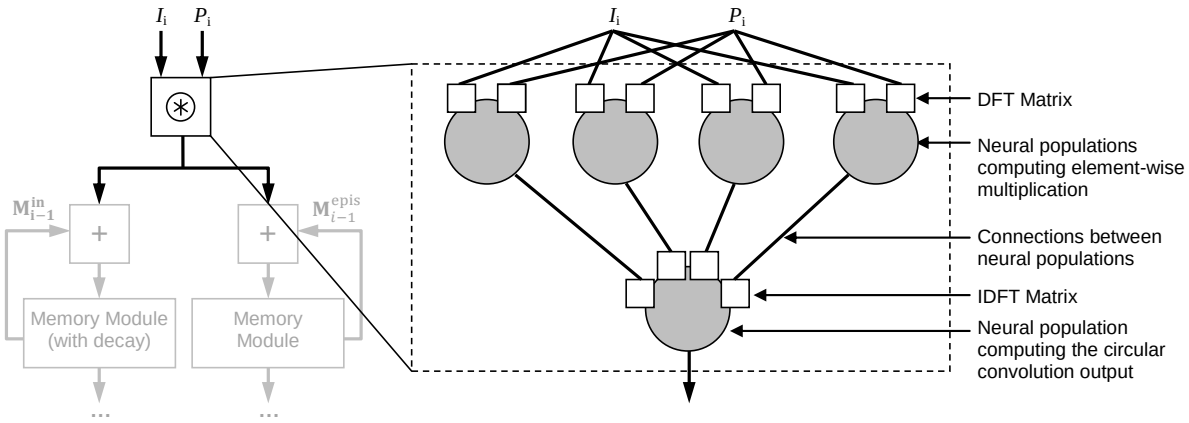

Figure 4.10: An illustration of the circular convolution operation implemented using a network of neural populations. Also shown in the figure is the location of the circular convolution operation within the OSE encoding network. The squares represent the connection matrices required to perform the DFT and IDFT operations, while the grey circles represent the neural populations required to calculate the element-wise multiplication operations, and the final circular convolution output.

Performing the Unbinding Operation

Recall that the unbinding operation is equivalent to convolving the memory trace with the approximate inverse of desired item's paired-associated vector. As mentioned previously, in the HRR, the approximate inverse is calculated using the involution function (refer to Equation (3.7)). Since the involution operation is essentially a function that rearranges the order of the vector elements, it can be computed by multiplying with a predefined matrix $\mathbf{L}$. The involution matrix is an $n \times n$ matrix with a 1 in the top left corner and a reverse diagonal of 1's in the bottom right corner. The involution matrices for 3, 5, and n dimensions are displayed below.

$$\mathbf{L}^3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \mathbf{L}^5 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{L}^n = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & 0 & \dots & 1 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 1 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \end{bmatrix} \tag{4.32}$$

Formally, the involution operation can then be written as:

$$\mathbf{A}' = \mathbf{AL} \tag{4.33}$$

Combining the equation above with the circular convolution equation results in the following equation, which is used to compute the convolution of a vector with the inverse of another vector. Figure 4.11 demonstrates how the unbinding operation is neurally implemented by using an appropriate connection matrix, and the neural implementation of the circular convolution function seen before.

$$\mathbf{A} \circledast \mathbf{B}' = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{A}) \odot \mathcal{F}(\mathbf{BL})) \tag{4.34}$$

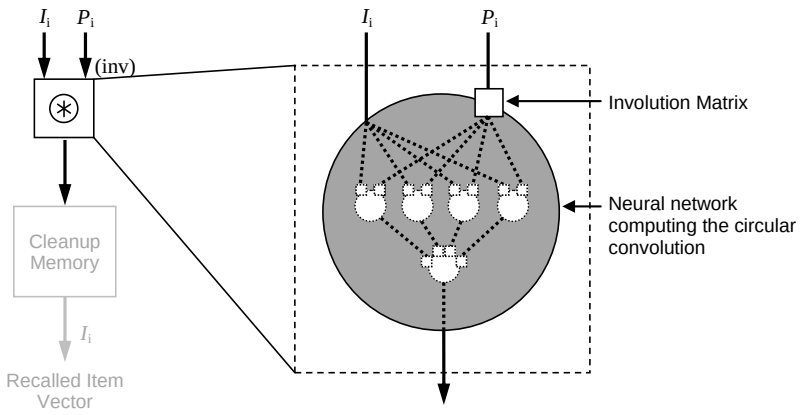

Figure 4.11: An illustration of the unbinding operation implemented using a network of neural populations. Also shown in the figure is the location of the unbinding operation within the OSE recall network. The solid square represent the involution connection matrix required to perform the involution operation operations, while the large grey circle represents the neural network required to calculate circular convolution operation (see Figure 4.10).

4.3.2 Remembering Over Time

The Most Basic Memory Circuit

The most basic memory circuit that can be implemented in neurons is an integrator. Just like an integration in mathematics which sums a signal over a given time period, an integrator circuit will continuously add a given input signal to its internal state; in essence, summing the input signal over time. When the input signal is removed, the integrator maintains the value of its internal state (because it is adding 0 to its internal state), thereby acting as a rudimentary form of memory. The equation for a simple integrator is:

$$x(t) = Ax(t) + Bu(t), \tag{4.35}$$

where $u(t)$ is the input signal to the integrator, $x(t)$ is the output of the integrator, and A and B are the integrator constants. Eliasmith and Anderson (2003) describe the process of deriving

these constants for a simple integrator in Chapter 8 of their book. For an ideal integrator, these constants are set to $A = 1$ and $B = \tau^{PSC}$, where τ^{PSC} is the time constant of the PSC (see Section 4.1.1) of the population used to implement the integrator. The figure below shows the high level network diagram of an integrator implemented using a population of neurons.

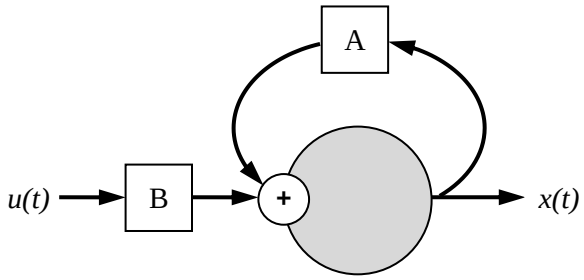

Figure 4.12: Simple schematic of how an integrator is implemented with a population of neurons. The shaded circle represents the population of neurons and the square boxes represents the constant weights A and B applied to each signal. For an "ideal" integrator, $A = 1$ and $B = \tau^{PSC}$.

What is interesting about the integrator is that using an A value of less than 1 results in an integrator whose internal state decays over time. This behaviour is exactly what is required for the input buffer component of the OSE model, and the determination of the A value that produces realistic decay times will be discussed shortly.

A Slightly More Complex Memory Circuit

Although the integrator is capable of acting as a memory circuit, using the integrator is somewhat tedious. The value that the integrator is holding is dependent on both the magnitude of the input signal, and the time for which the input signal is presented to the circuit. For example, presenting a square pulse of height 1 to an integrator for 1 second results in the integrator holding a value of 1. However, if the same signal (a square pulse of height 1) is only presented for half a second, then the resulting value stored in the integrator is 0.5.

This undesirable behaviour of the integrator can be improved by adding a subtraction before the integrator. This alters the behaviour of the integrator by constantly providing input to the integrator until the output of the integrator matches the input signal, at which point, the input to the integrator becomes zero, and the integrator maintains its value. There is one major flaw with this new design however, and that is the fact that it no longer functions as an integrator. Because of the subtraction, if the input is reduced to zero – which in the case of the simple integrator, will cause it to hold its value – the modified design will try to match the integrator with the input, effectively erasing the stored value. This problem can be easily rectified by gating the subtraction operation, so that it can be "turned off" when the integrator needs to hold a value. The process of gating a population of neurons is described in Section 4.3.3. The modified integrator circuit,

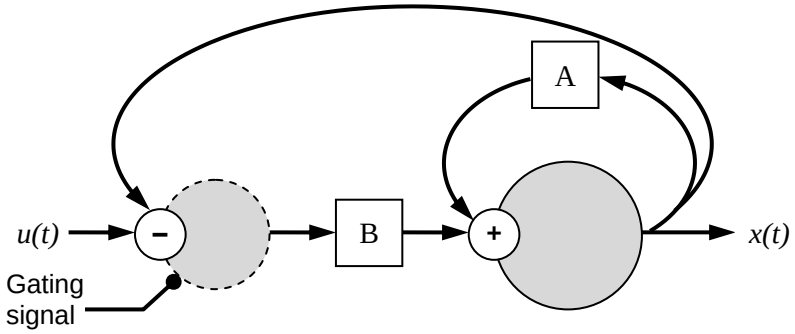

Figure 4.13: Simple schematic of how a gated integrator can be implemented using two neural populations. In addition to the neural population used to implement the integrator, a second population of neurons has been used to implement the subtraction operation. This neural population can be "gated" (indicated by the dashed outline of the population) through the use of an inhibitory gating signal (the inhibitory nature of the gating signal is indicated by the rounded termination).

with the gating signal is illustrated in Figure 4.13. Figure 4.14 also demonstrates the behaviour of the "gated integrator" to varying configurations of input signals.

Outputting and Storing Values Simultaneously

There is one last problem that needs to be reconciled. This is the problem of simultaneous output and storage. Referring back to Figure 4.9, we see that the output of the memory module is fed back to itself. If the gated integrator described above were to be connected in this fashion, this feedback connection will cause whatever changes made to the internal state of the integrator to be fed back into the input of the integrator. This in turn causes the integrator to continue integrating until the neuronal population saturates. To rectify this problem, the integrator needs to be redesigned such that it is able to output the value of M_{i-1} while at the same time, store the value of M_i.

In order to achieve a memory circuit that is able to simultaneously output and to store values, two gated integrators are used in parallel, as illustrated by Figure 4.15 which shows the neural implementation of the memory module, and its location in the OSE encoding network. Appropriate gating signals are then sent to each integrator such that when one integrator is storing the input value, the other integrator is driving the output of the memory circuit, as illustrated in Figure 4.16.

Although the memory circuit above is implemented in this manner, it does not necessarily mean that it is the same way in the brain. The memory circuit designed for the OSE model can be thought of as several dedicated memory slots (the gated integrators), used in conjunction with some sort of control and routing circuitry (the gates and gating signals).

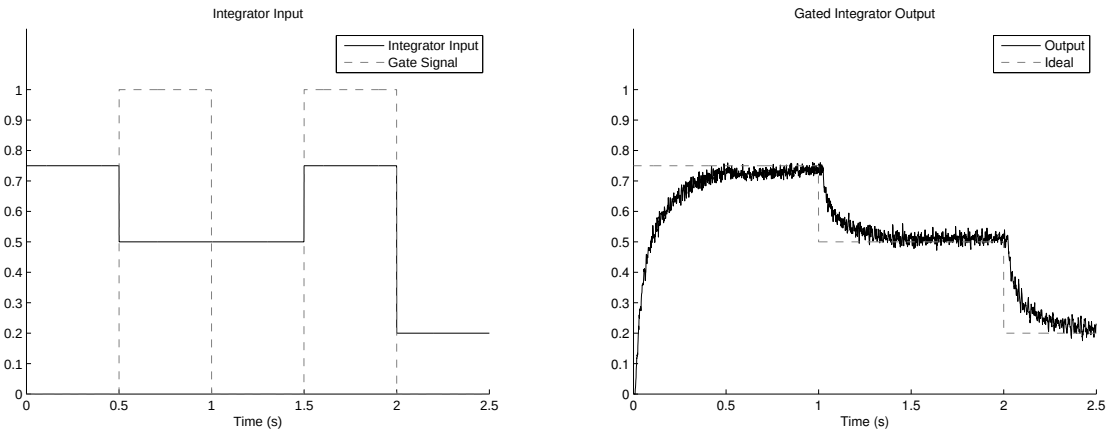

Figure 4.14: Simulation of a gated integrator. (Left) Input signals provided to the gated integrator. When the "Gate" signal is 1, the gated integrator stops integrating and holds the value it has previously stored. (Right) Plot of the output of the gated integrator implemented in spiking neurons. A network of 125 neurons – 75 for the integrator, and 50 for the "gate" – was used. The graph plots both the output of the integrator (solid line) and the ideal output of the integrator (dashed line).

Memory That Decays

The mechanism by which the decay in the OSE input buffer component is implemented is crucial because it has to mimic the short-term retention capacity of the human working memory system accurately. The equation derived for the input buffer memory trace,

$$\mathbf{M}_i^{\text{in}} = \gamma \mathbf{M}_{i-1}^{\text{in}} + (P_i \circledast I_i),$$

is not ideal because it only decays the memory trace every time an item is added, rather than mimicking the continuous decay seen in human delayed recall studies [Peterson and Peterson, 1959, Murdock, 1961]. However, as mentioned previously, using a value of less than 1 for the A constant in the integrator will cause the value stored in the integrator to decay to zero at a constant rate, which is the behaviour desired for this component. Figure 4.17 illustrates the behaviour of the integrator implemented with varying values for A.

In order to determine the appropriate decay value required to match human data, data from Reitman's 1974 paper was used. In this paper, as in the Peterson and Murdock papers mentioned before, subjects were instructed to remember a list of items, and after a predetermined delay, were asked to recall the items; the number of items successfully recalled was then recorded. The subjects in the Peterson and Murdock experiments were instructed to perform a backwards counting task (backwards counting by threes) for the duration of the delay period. The intention of this was to prevent subjects from rehearsing any of the list items during the delay period. There is no way, however, to determine whether the backwards counting was successful at preventing

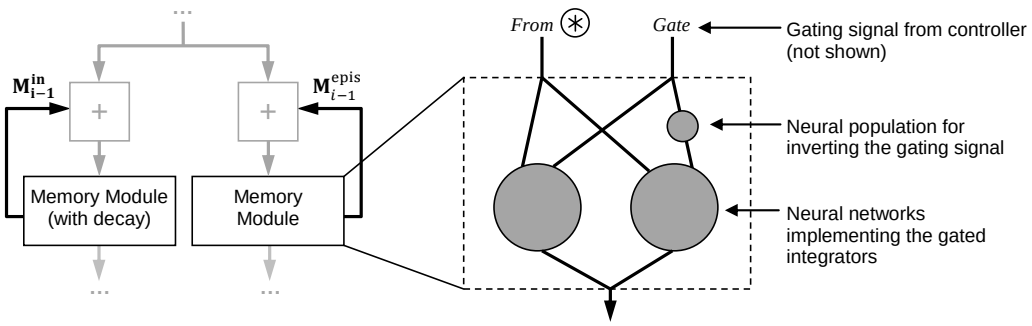

Figure 4.15: An illustration of the memory module implemented using a network of neural populations. Also shown in the figure is the location of the memory module within the OSE recall network.

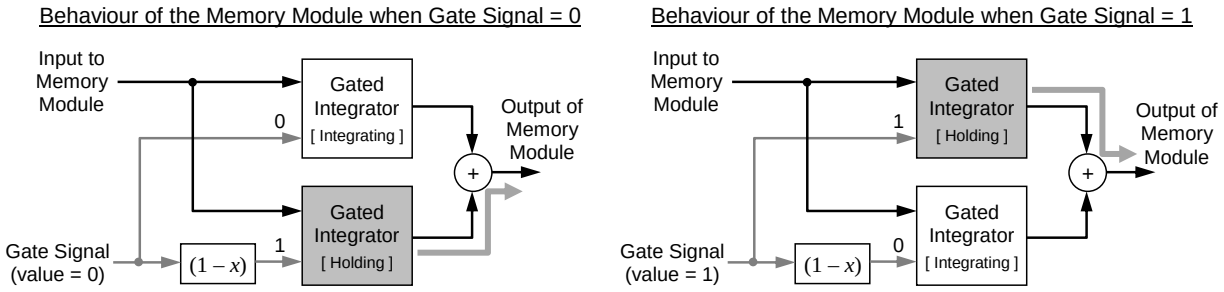

Figure 4.16: Illustration of the two operating states of the OSE model's memory module. (Left) Operating state when the "gate" signal to the memory module is 0. In this state, the top integrator is integrating, and the bottom integrator is holding its value. The thick grey arrow indicates that the bottom integrator is driving the output of the memory module. (Right) Operating state when the "gate" signal is 1. In this case, the roles of the integrators are reversed. Now the top integrator is holding its value and driving the memory module output; while the bottom integrator is integrating.

rehearsal; or if the backwards counting had any effect on the recall accuracy data. After all, the backwards counting task does involve working memory and may interfere with the memory trace of the remembered list.

In Reitman's study, the distraction task used was a signal detection task, where the subjects were instructed to identify a specific tone within an audio stream. Because the audio stream contained no words or numbers, Reitman argues that using this task as a distraction would place no additional load on the working memory system. In addition to the superior distraction task, for each subject, Reitman also compared the recall and tonal detection performance of the experimental condition to the recall and detection performance in a controlled condition where the

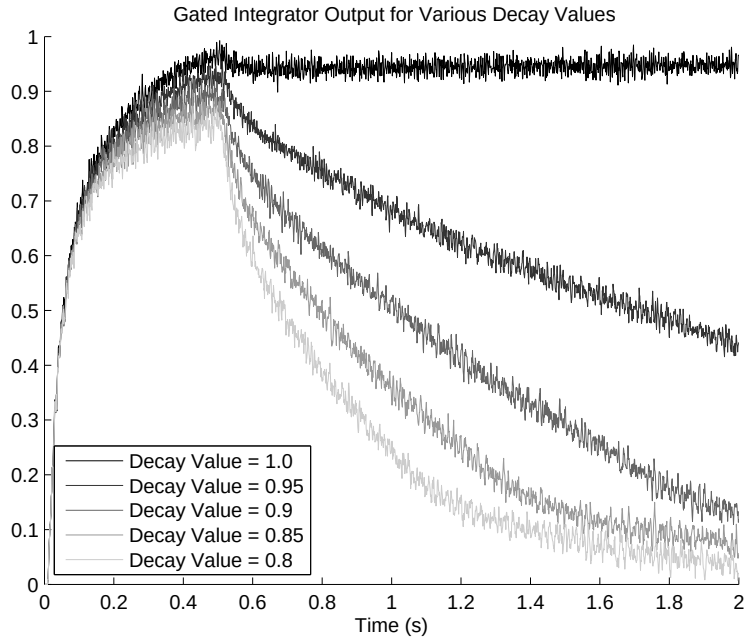

Figure 4.17: Plot of the effect on the integrator output for varying values of the decay parameter, A. For the ideal case, where the integrator maintains its value, $A = 1$. For values of $A < 1$, the rate of decay of the output increases as the value of A decreases.

subjects were explicitly asked to perform rehearsal. By doing this comparison, it can be inferred if subjects did in fact rehearse, either admittedly or unknowingly, during the delay period. The results of the experiment showed that after a 15 second delay period, the subjects were able to recall on average 65% of the items that they recalled immediately after the list presentation.

To tune the OSE's input buffer decay, the recall accuracy of the input buffer component with various decay values was tested against the data in Reitman's paper. Because the OSE's episodic buffer component is meant to model the effects of rehearsal and longer-term memory, it was excluded in the test model used to calibrate the decay value. As per Reitman's study, this test model was presented 5 items for 2 seconds, after which an immediate recall task was performed on the model. Following a 15 second delay period, the recall performance of the model was then tested again. Comparing the items recalled on the immediate recall task with the items recalled after the delay period yields the percentage of items the model was able to recall after the delay period. Figure 4.18 shows the test network used, and the results of the testing procedure. From the results, using a value of 0.9775 for the value of A reproduces the 65% recall performance recorded in Reitman's paper.

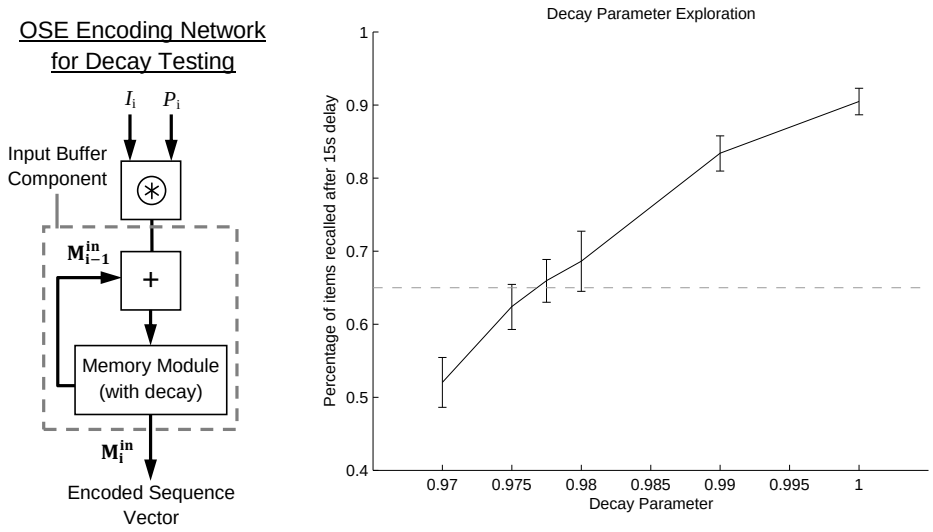

Figure 4.18: (Left) Diagram of the network used to tune the decay parameter for the input buffer component of the OSE model. Since the testing procedure does not include rehearsal, the episodic buffer component of the OSE model was excluded from the testing procedure. (Right) Results of the testing procedure with various values for the decay parameter. The model was simulated for 250 trials for each value of the decay parameter. The error bars on the plot indicate the 95% confidence intervals in the recall data.

4.3.3 Gating Representations

As mentioned before, a method of gating signals is required to implement the memory module for the OSE model. This section presents a simple and biologically plausible mechanism by which this can be achieved. To explain this method, we return to the response curves for each neuron. For one neuron representing a scalar value, it is easy to see how its output can be gated. By applying a strong input that is in the negative direction of the neuron's tuning curve (i.e. along the negative x-axis), any input to the neuron would not be sufficient to make the neuron fire. In a population of neurons, however, the tuning curve for each neuron is aligned in random directions. Applying a strong input in the negative direction of any one neuron in the population, there will be other neurons in the population with a tuning curve in the aforementioned direction, which would not achieve the goal of gating the population. However, if a negative input (an inhibitory input), could be injected directly into the neuron, bypassing the tuning curves, then the output of the entire population can be inhibited, effectively gating its output.

On the population level of neurons, this additional inhibitory input is represented just like any other input to the population, and the output of the population can be represented as the linear combination of the original input x, and an inhibitory input q (see Section 4.2.5 to refer to the equations for a linear combination of inputs). The calculation for the connection weight matrix

between the input population $\mathbf{x}$ and the output population $\mathbf{z}$ is calculated as usual – that is to say, using the decoders and encoders to calculated the weight matrix. However, the connection weight matrix for the inhibitory input population, $\mathbf{q}$, is predefined such that every connection has a weight of -1. The input current to each neuron in the output population is then:

$$J^{\mathbf{z}}(C_1\bar{\mathbf{x}} + C_2 q) = \sum_i w_i^{\mathbf{x}} a_i^{\mathbf{x}}(\bar{\mathbf{x}}) + \sum_j w_j^{\mathbf{q}} a_j^{\mathbf{q}}(q) + J^{bias}, \tag{4.36}$$

where $w_i^{\mathbf{x}} = \alpha C_1 (\boldsymbol{\varphi}^{\mathbf{z}} \bullet \boldsymbol{\phi}_i^{\mathbf{x}})$, and $w_j^{\mathbf{q}} = \alpha C_2(-1)$. Note that the value of the weight C_2 can be set to any arbitrary value that is at least greater than the maximum optimal representational range of the neuronal population. In the neural implementation of the OSE model, this range is typically set from -1 to 1, so the value of C_2 must be at least 2. A value of 5 was chosen, however, to ensure that the population is still inhibited if an input outside of this representational range is provided to the neuronal population.

Figure 4.19 shows the effect of adding an inhibitory input to a population of neurons representing the identity function ($y = x$). The gating of a neural population in this manner is not,

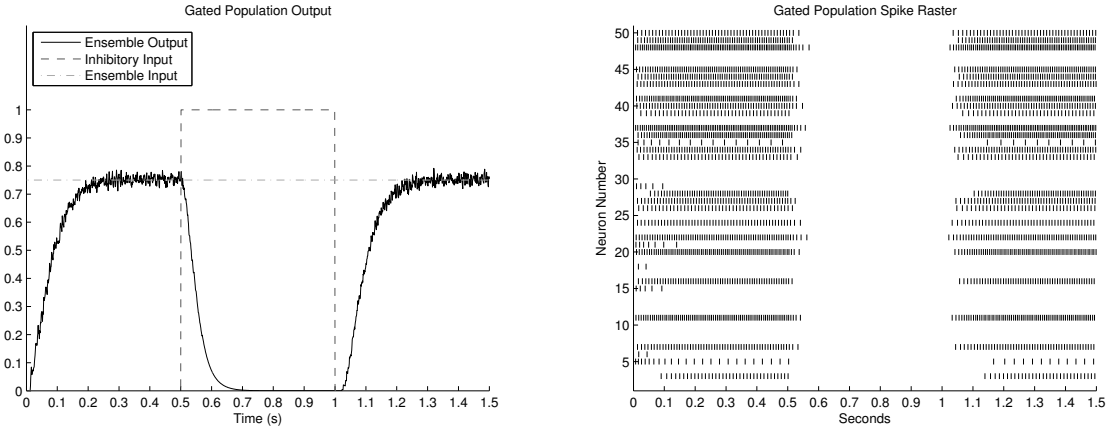

Figure 4.19: Illustration of the effect of inhibiting the output of a 25-neuron population. (Left) Plot of the decoded output of the neural population overlaid with the inhibitory input to the population, and the input signal to the population. Notice that when the inhibitory input is present, the output of the population drops to 0, and all the neurons in the population stop spiking (see right figure). (Right) Spike raster plot for each neuron in the population.

neurobiologically speaking, implausible. There are many neurons in the brain called "inhibitory interneurons" which produce neurotransmitters that decrease the chance of the postsynaptic neuron firing, rather than increasing it as with excitatory connections. Inhibitory connections are also found in major pathways in the brain.

4.3.4 Cleaning Up Representations

The need for a vector cleanup operation has already been discussed in the previous sections. In this section, we explore how this operation can be implemented using a population of neurons. A crucial step to the cleanup operation is to compute the similarity of the input vector to all the vectors contained within a known vocabulary of vectors. The equation for the neuron's input current provides a solution on how this can be achieved. The similarity operation (dot product) is already performed between the input vector and the encoding vector of a neuron, so by predefining the neuron's encoding vector as a vector from the vocabulary, the activity of the neuron becomes proportional to the similarity of the input vector to the vector from the vocabulary. Using a population of neurons, all with the same preferred encoding vector, a linear response to the input vector can be derived.

Configuring the population to output a vector that is identical to the vector from the vocabulary is equally easy. Using appropriately scaled versions of the vocabulary vector as the decoders for the neural population will enable the population to output a vector that is identical to the vocabulary vector, but with a length proportional to the similarity measure mentioned before.

It is also possible to threshold the activity of the population such that only a similarity measure above a certain value will activate the population. By choosing appropriate values of α and J^{bias} for each neuron, the intercepts for each neuron (i.e. the x value at which they start firing) can be set such that they are all above the desired threshold. Figure 4.20 illustrates the neuron response curves when this is done for a threshold value of 0.3, which is the value used in the OSE's spiking neuron network. The method described in this section is outlined elsewhere [Stewart et al., 2009], where it is shown that not only can this method be implemented using realistically noisy spiking neurons, but the cleanup accuracy of this method approaches that of an ideal mathematical cleanup operation.

4.3.5 Data Flow Within the Model

Now that the implementation details of the different components in the OSE model have been presented, the complete OSE model can be constructed. Figure 4.21 shows the complete OSE encoding network, and provides an example of how the two-itemed list $[\mathbf{A}, \mathbf{B}]$ is encoded, while Figure 4.22 illustrates the complete OSE recall network, and provides an example of how items from the list encoded in Figure 4.21 are retrieved.

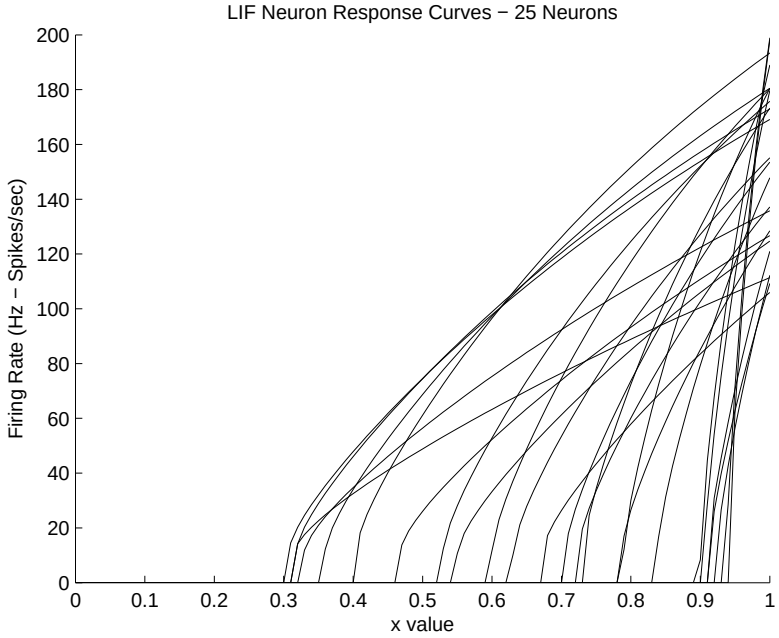

Figure 4.20: Plot of the neuron response curves for a population of 25 neurons with intercepts predefined to be greater than $x = 0.3$. When used in a cleanup memory network, any similarity measures less than 0.3 will not cause the population to spike, thus ignoring that input.

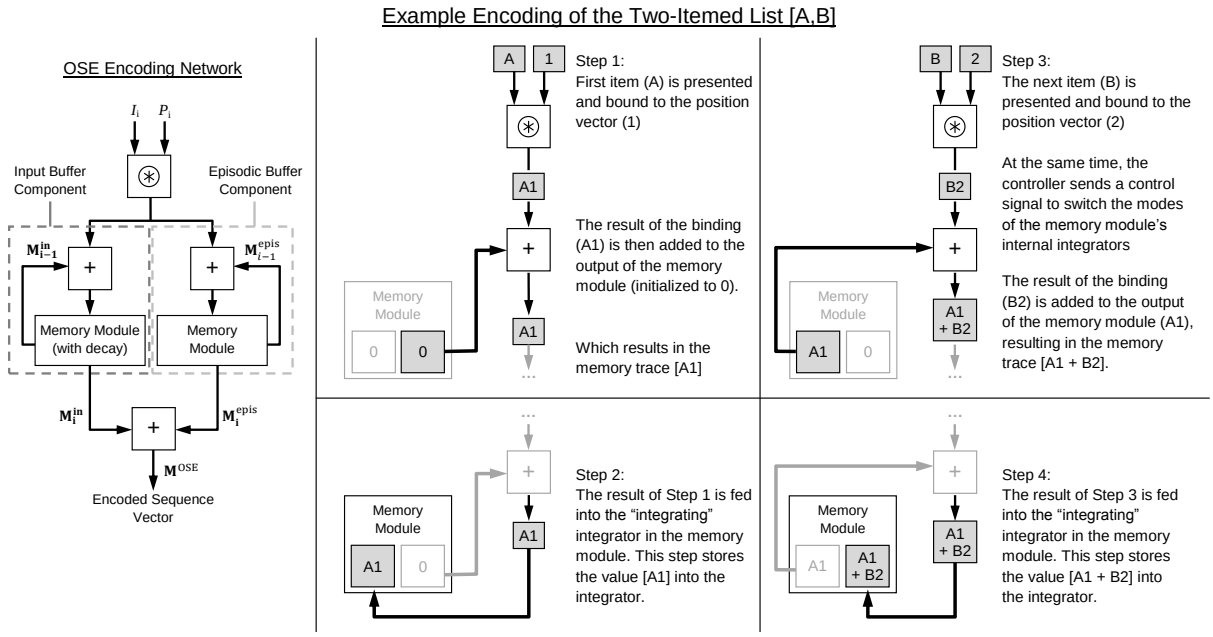

Figure 4.21: An illustration of the flow of information through the OSE encoding network for the example of encoding a simple two-itemed list. The complete encoding network is redisplayed as a quick reference guide to the location of each network component. Note that in the illustration, the flow of information is only shown for half of the network because the flow of information in the other half of the network is exactly the same.

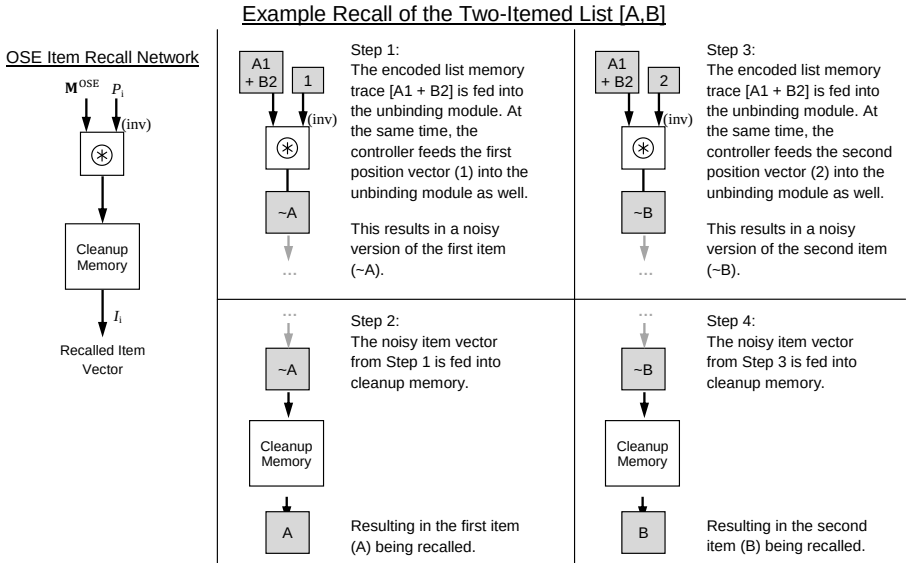

Figure 4.22: An illustration of the flow of information through the OSE recall network for the example of decoding the two-itemed list encoded in Figure 4.21. The complete recall network is redisplayed as a quick reference guide to the location of each network component.

Chapter 5

Results of the OSE Model

This chapter presents the simulation results of the OSE model implemented in a network of spiking neurons. The model parameters used to generate the results in this section are: for the episodic buffer rehearsal parameter, $\rho = 1.6$; and for the input buffer integrator decay parameter, $A = 0.9775$. The value of the episodic buffer rehearsal parameter was chosen because it produced the best match to the wide array of human data presented in this section, while the derivation of the integrator decay parameter has already been discussed in Section 4.3.2.

The recall tasks discussed in this section are: the immediate forward recall task, the delayed forward recall task, the backwards recall task, and forward recall with alternating confusable lists. For each of these tasks, the simulation of the model used 50-dimensional item vectors and was run for an average of 250 trials.

5.1 Immediate Forward Recall

Immediate forward serial recall is the task where a list of items is presented to the subject at a constant rate. Immediately after the item presentation, the subject is instructed to recall the items in the order they were presented. As mentioned in Section 1.1, recall data collected from human studies reveal that the recall performance typically show a strong primacy effect and a weak recency effect. This section compares the recall performance of the OSE model to human forward recall data [Drewnowski and Murdock, 1980]. The results of this comparison are displayed in Figure 5.1.

As the results show, the OSE model is able to reproduce the effects show in the human data. One must keep in mind that the the decay parameter used in the OSE model has not been fitted to the recall data from the aforementioned papers, but to the experiment performed in Reitman's paper.

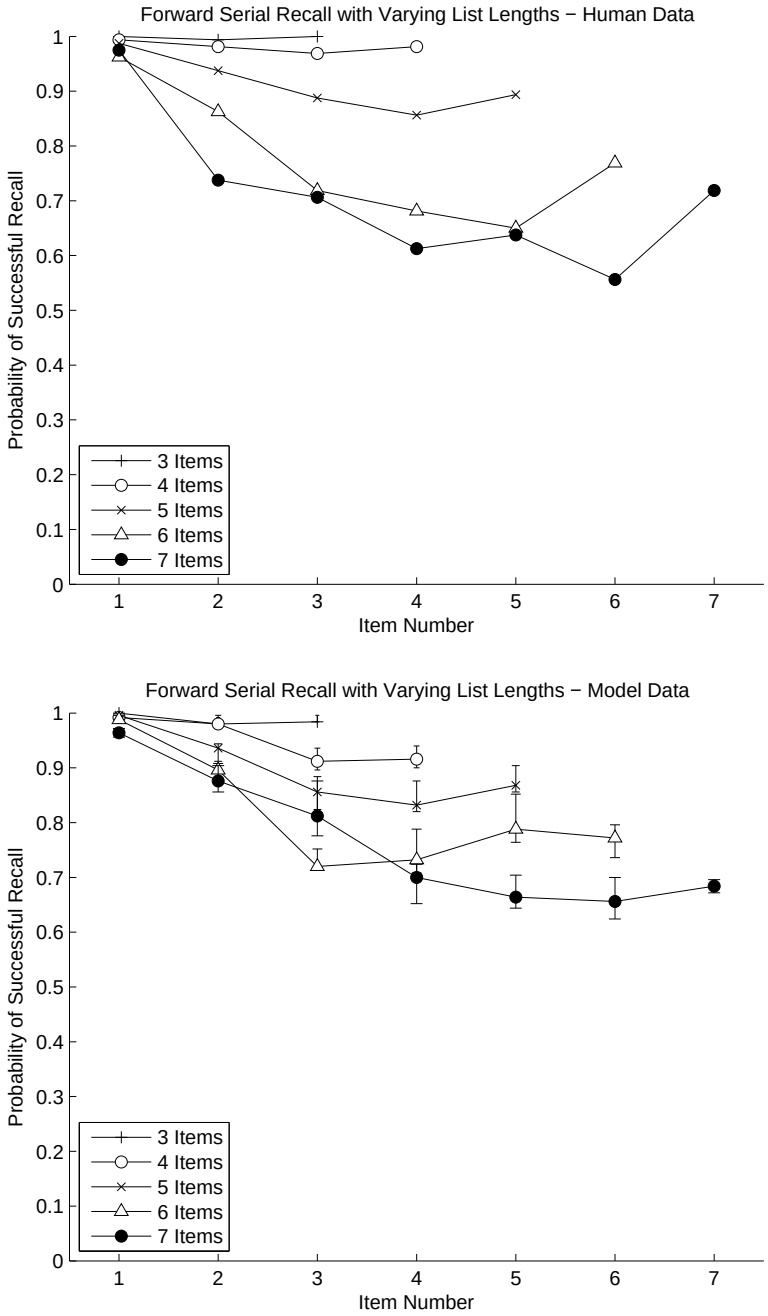

Figure 5.1: Comparison of the results for the immediate forward recall task. (Top) Data collected from [Drewnowski and Murdock, 1980, auditory data] showing the human recall performance for lists of 3 to 7 items. (Bottom) Data generated using the OSE model, demonstrating the model's ability to reproduce the primacy and recency effects seen in the human data. The error bars shown on the plot indicate the 95% confidence intervals.

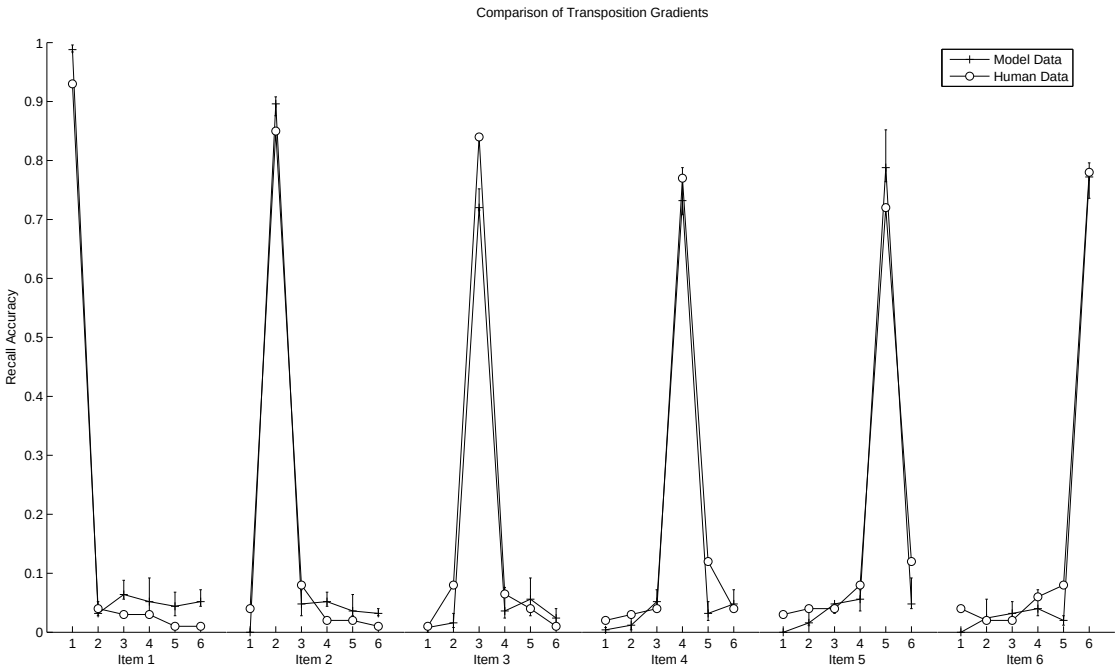

Figure 5.2: Comparison of the transposition gradients obtained from human experiments (taken from [Henson et al., 1996]), and the transposition gradients generated by the OSE model. Analysis of the data indicate that the OSE model is able to reproduce the behaviour observed in the human data.

5.2 Transposition Gradients

The transposition gradient measures the probably of recalling an item outside of its proper position. Many studies have been undertaken to explore this effect; unfortunately, these studies generally involved free recall rather than serial recall [Lee and Estes, 1977, Nairne, 1990]. Transposition error data involving serial data, while lacking, has been reported for a list of 6 items in Henson et al.'s 1996 paper. The data shows that under normal test conditions, the probability of a transposition error decreases as items get further away from their original position. Figure 5.2 compares the results of the simulation of the OSE model with the data presented in [Henson et al., 1996], showing that the model is also able to capture this effect in the transposition gradients.

Analysis of the transposition data indicate that for the first two items in the list, the transposition errors generated by the model are much larger than the errors observed in the human data. A detailed investigation reveals that the most likely cause is the simplistic encoding mechanism used for the episodic buffer component of the OSE model. Since the memory trace in the

episodic buffer does not degrade over time, for items later in the list, it tends to "overpower" the memory trace in the input buffer, causing incorrect recall. This indicates that for future implementations of this model, a more complex encoding mechanism may be required for the episodic buffer component, possibly involving a method by which the memory trace is degraded as items are recalled.

5.3 Delayed Forward Recall

The delayed serial recall task is identical to the forward recall task in terms of the list presentation procedure, however, the subject is instructed to wait a specified amount of time before recall. This waiting time is typically filled with a distractor task, such as backwards counting by threes, to prevent the subject from rehearsing the list. In delayed recall studies (both for free recall and serial recall), a strong primacy effect is observed, but the recency effect diminishes as the length of the delay period increases [Jahnke, 1968].

In this section, the simulation results of the OSE model is compared to human data [Jahnke, 1968]. In the simulation, the model is presented with three lists of 6, 10, and 15 items, respectively, at a rate of 2 items per second. After a list has been presented, the memory trace is allowed to decay for the specified delay periods (0, 3, 9, and 18 seconds), before the recall performance is assessed. To simulate the suppression of rehearsal, no additional information is presented to the model during the delay periods. This comparison is presented in Figure 5.3.

The results show that the model is able to reproduce the diminishing recency effect phenomena for all of the lists lengths. One thing to note in the data generated by the model is the much higher recall accuracies for the first few items in the list. This behaviour is more prominent for the longer 10, and 15 list lengths. Analysis of the data reveals that the cause of this is the naive method of simulating rehearsal that is employed by the OSE model. A more complete method of simulating rehearsal will need to be developed for future versions of this model.

5.4 Immediate Backwards Recall

The task of immediate backward recall involves the recall of the list in the reverse order from which it was presented. Human data [Madign, 1971] reveal that in backward recall, the recency effect is substantially more pronounced than it is in forward recall. Using the concept of the dual-store memory, it is easy to see how this effect can arise. In forward recall, decay in the short-term memory store causes items that are recalled later in the list to have decayed by the time they are recalled, resulting in a smaller recency effect. However, in backwards recall, this decay is less significant because the later items are recalled first, thus resulting in a larger recency effect. As seen in Figure 5.4, this behaviour is also captured by the OSE model.

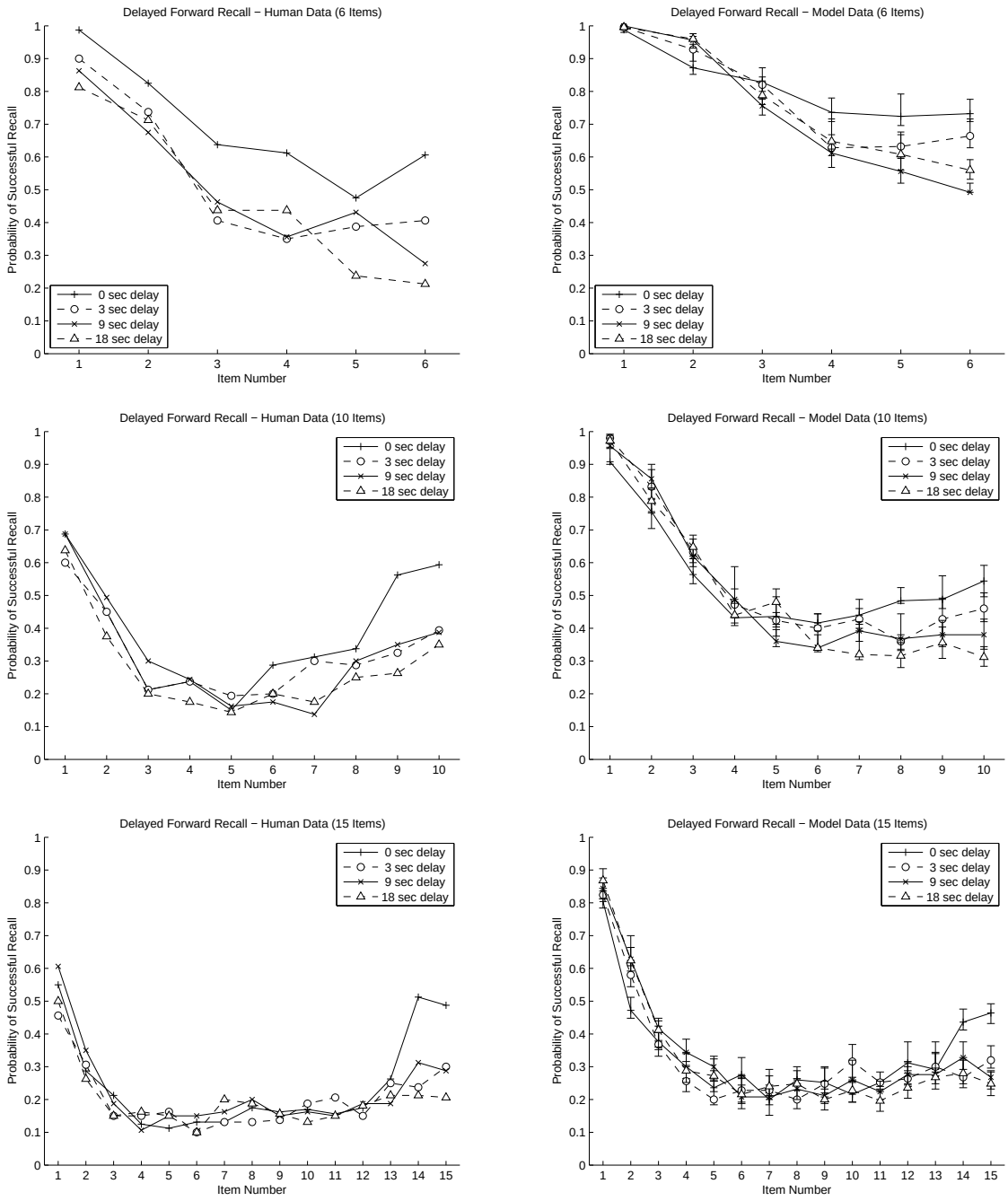

Figure 5.3: Comparison of human and model-generated results for the delayed forward recall task. The left column displays the human data (taken from [Jahnke, 1968]), while the right column displays the serial position curves generated by the model. Each row correspond to a specific list length; starting at 6 items for the top row, 10 items for the middle row, and 15 items for the bottom row. The results shown in this figure demonstrates the model's ability to capture the decreasing recency effect for increasing delay periods.

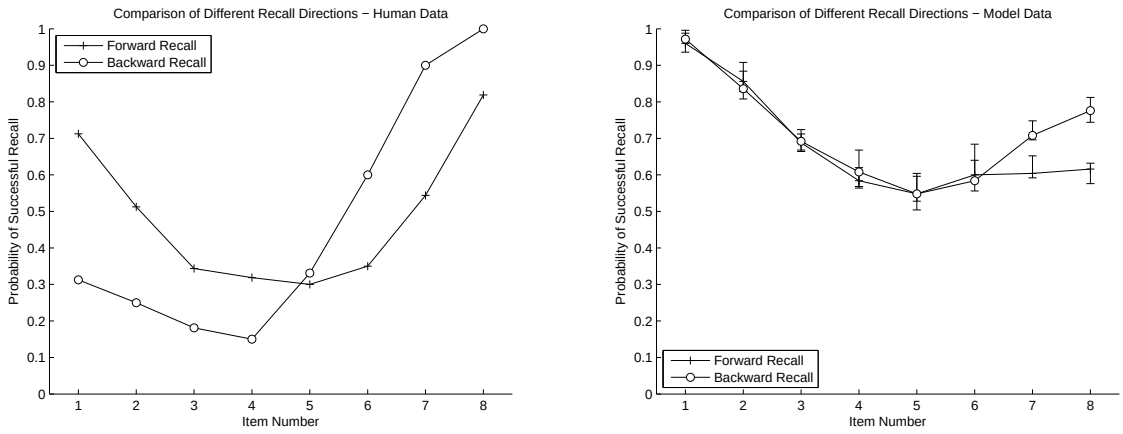

Figure 5.4: Comparison of results obtained from human experimental studies and the OSE model for the immediate backwards recall task. The results obtained for the forward recall task are also plotted to facilitate comparison of the relative primacy and recency amounts in the data. (Left) Human behavioural data from [Madign, 1971, auditory modality]. The human data shows a marked increase in the recency effect, as well as a substantial decrease in the primacy effect. (Right) Results obtained from the simulation of the OSE model. See text for discussion of model results.

Comparing the model to the human data reveals that while the model does capture the behaviour of a general increase in recency, the effect produced by the model is not as large as that seen in the human data. Additionally, there is no decrease in primacy in the model data. The lack of effect on the primacy phenomena is understandable, given that the memory traces stored in the episodic buffer do not degrade over time. A possible solution to this has already been mentioned in Section 5.2.

The effects on recency are more complex to account for, however. First and foremost, in his report, Madigan- did not provide a full account of the experimental details. Since the OSE model implements temporally sensitive components, details like the presentation rate are important to produce accurate simulation results. Secondly, it is not known what kind of information the subjects were given during the experiments. The simulation run on the model assumes that the subjects were informed that they were to perform the standard forward recall task, but at the moment of recall, were asked instead to recall the list in the reverse order. If the subjects were informed before the list is presented that the task is going to be a backwards recall task (and this is most likely the case), they might prioritize more recent items over items at the start of the list. For the OSE model, this would be implemented by designing some method of prioritizing the importance of the two (episodic buffer and input buffer) memory traces. While this ability is currently unavailable in the OSE model, it can be roughly approximated by disabling the episodic buffer component completely. The recall performance of the model when this is done is shown in Figure 5.5, and can be seen that the model is able to match the human data – at least for the

last half of the list – more closely.

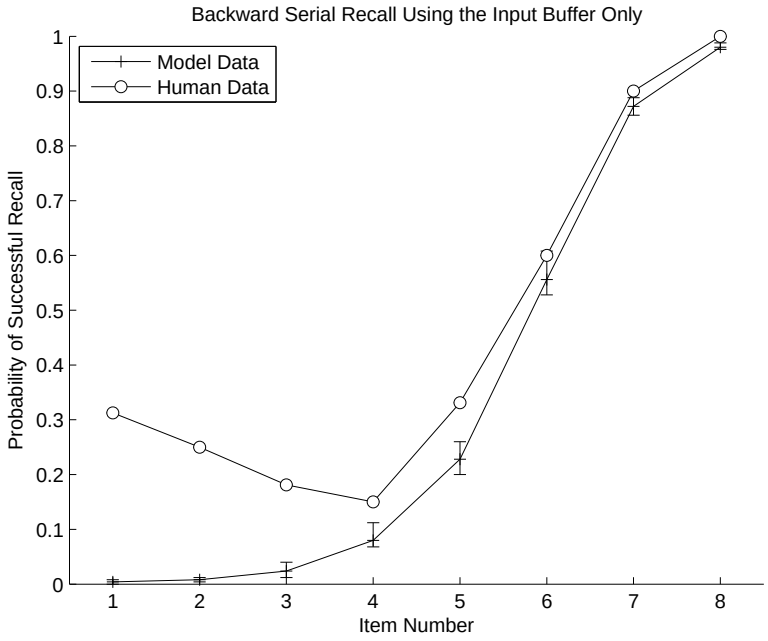

Figure 5.5: Comparison of the human and OSE model backward recall performance for the scenario that the episodic buffer has been ignored during the encoding process. This is done to simulate the subject preferring the input buffer memory trace over the episodic buffer memory trace because they have been informed that the task will be that of backward recall. As with Figure 5.4 the human data is taken from [Madign, 1971].

5.5 Alternating Confusable Lists

This section examines the effect that the use of alternating confusable lists has on the recall performance of the experimental subjects. As discussed in Section 2.3.1, this task was developed by [Henson et al., 1996] and used as a method of disproving the viability of chaining methods as models of serial memory. As a reminder, the experiment involved presenting subjects with lists containing confusable and non-confusable letters. In the experiment, four types of lists were presented to the subject: a list containing all confusable items, a list containing confusable items at every even serial position, a list containing confusable items at every odd serial position, and a list containing no confusable items. Confusability in the experiment was defined on the basis of phonological similarity – or the ability for letters to rhyme – thus, letters like "B", "D", and "G" are examples of confusable letters whereas "H", "K", and "M" are examples of non-confusable letters.

In order to simulate the effect of confusable items on the recall performance of the OSE model, they must be defined. Using the concept of phonological similarity, item confusability for the OSE model was defined using the dot product measure of vector similarity. For the results displayed in Figure 5.6, confusable vectors were chosen such that taking the dot product of two confusable item vectors resulted in a value between 0.25 and 0.5. As demonstrated in the figure, the OSE model is able to reproduce the behavioural data observed in humans. First, a distinctive saw-tooth pattern emerges for lists with alternating confusable items. Second, the recall performance for the list comprised entirely of confusable items is poorer than the recall performance of the other lists. Lastly, there is a tendency for non-confusable items within the alternating lists to be recalled more accurately than in the standard (non-confusable) list. Also displayed in the figure is the plot of the transposition curves for the list with even-positioned confusable items, which shows that the OSE model is also able to capture the effect that confusable items have on the transposition curves.

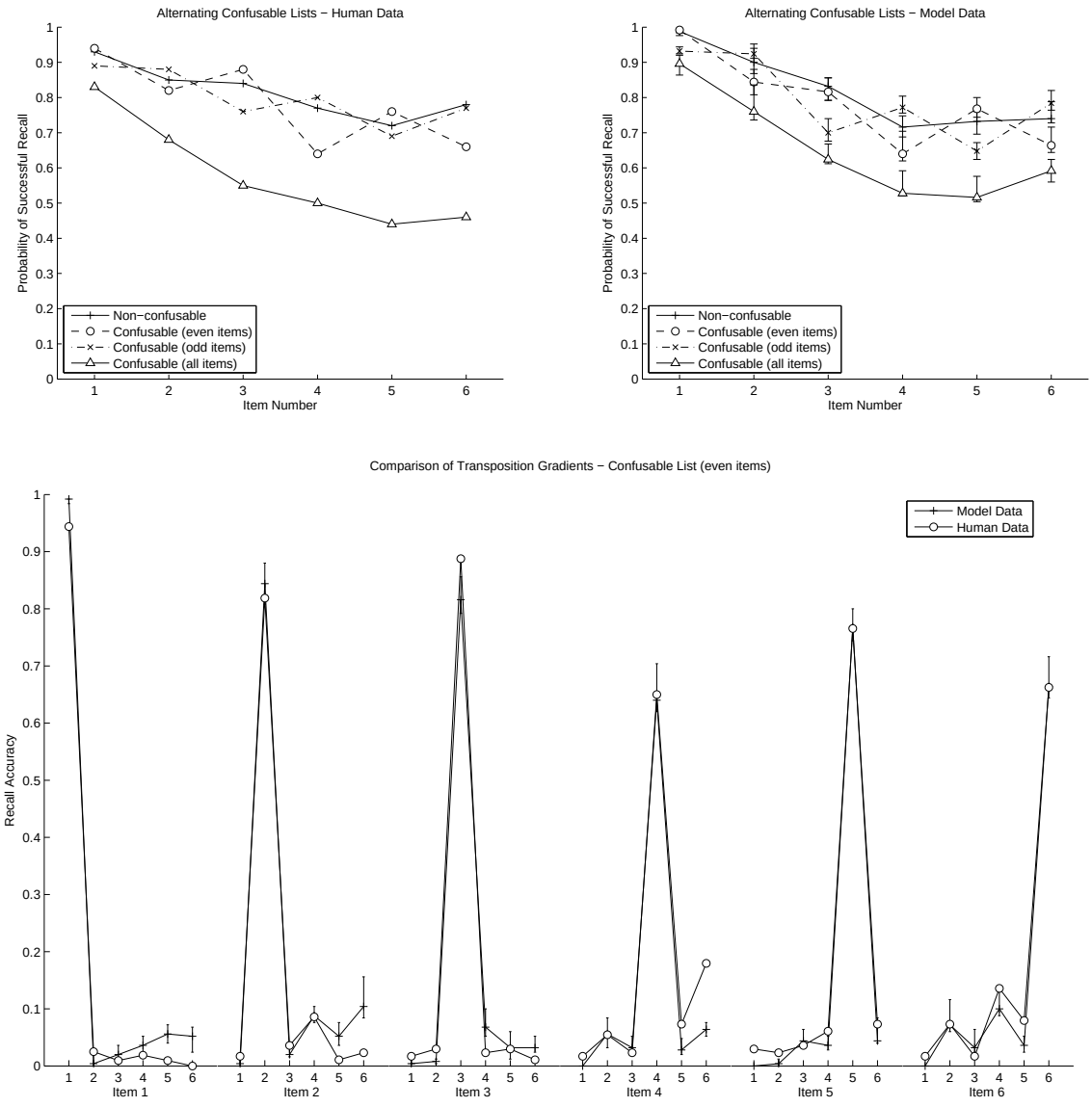

Figure 5.6: Results from the OSE model simulation with confusable items, and comparison to human data. (Top Left) Plot of the data from [Henson et al., 1996, Experiment 1], showing the distinctive saw-tooth pattern in the alternating confusable lists. (Top Right) Results from the OSE model simulation, demonstrating its ability to reproduce the results evidenced in the human data. (Bottom) Comparison of the transposition gradients of the human subjects versus simulations of the OSE model. The list used in this plot contained confusable items at each even-numbered position.

Chapter 6

Extensions to the OSE Model

It is acknowledged that the OSE model presented in this thesis is incomplete. This chapter will examine some possible extensions and improvements that can be made to the OSE model. Several issues will be discussed in this chapter; first, possible methods for generating the position vector; next, grouping effects; and lastly, extending the OSE to model free recall.

6.1 Generating the Position Vector

The position vectors used in the simulations of the OSE model described above are randomly generated before the start of each simulation. However, this method of generating position vectors poses a major problem: if the system requires the position vectors to retrieve each item, the position vectors have to be stored elsewhere. Additionally, if the system is required to recall the items *in order*, then the position vectors have to be stored in the order in which they were generated, which was the original problem. In reality, randomly generated position vectors do not solve the problem of how serial information is stored, rather, the problem has just been deferred from the item vectors to the position vectors. This section will discuss three potential solutions to this problem: hard-coding the position vectors, using convolution powers, and using the random walk process.

6.1.1 Hard-coded Position Vectors

Hard-coding of the position vectors is the simplest of the three solutions, and is the solution currently employed by the OSE model. This solution, however, requires a set number of position vectors to be stored within the system, which causes some undefined behaviours when the list length exceeds the number of position vectors. Do the position vectors loop back on themselves? Or perhaps a more complex encoding scheme, such as grouping (see Section 6.2) is used? Further investigation will be required to test these hypotheses.

6.1.2 Using Convolution Powers

The concept of convolution powers (refer to Section 2.4), and the drawbacks to using them (see Section 2.4.1) has already been discussed before. The issue with using convolution powers in the TODAM2 model is that the convolution powers are used on vectors that contain item information that needs to be encoded and stored. Normalizing these vectors to avoid the problem of the exponential increase in vector magnitude results in the loss of the item information, resulting in the decrease in recall accuracy. However, this problem is not present if convolution powers are used to generate the position vectors. This is because unlike the TODAM2 model, convolution powers used in the OSE model would be used on vectors that do not contain encoded information (i.e. any information that could be lost through the process of normalization), thereby reducing the impact on the recall accuracy of the model. In Figure 6.1, six position vectors were generated using this method, and the similarity of the vectors to each other is plotted. The figure displays the plot of the vector similarities averaged over 250 runs, as well as the vector similarities of a arbitrarily chosen run.

From the results of the simulation, several issues are evident. First, there is no gradual decrease in the similarity of the position vectors. This is not necessarily a problem, since this will also be the case for randomly-generated hard-coded position vectors. The plot of a single run of the algorithm reveals a potential problem, which is that the position vector similarity does not always decline as the distance from the starting position increases. To test the convolution-power-generated position vectors, a simulation was run using a non-spiking implementation of the OSE model. Figure 6.2 shows the results of this simulation for a six-itemed list.

The results indicate that using convolution powers to generate the position vectors do not yield a very satisfactory result. The primacy effect is still present, however, it is much worse than before. Additionally, the recency effect is almost non-existent. What can be concluded from this simulation is that in its current form, the convolution power approach is not feasible.

6.1.3 Using The Random Walk Process

The random walk process[12] is a method by which one vector can be slowly transformed into another vector by perturbing the original vector by a small random amount at every time step. More precisely, at every time step, a random value chosen from a normal distribution is added to each element of the vector, causing the vector to drift away slowly from the starting vector. If the similarity of the drift vector is compared to the starting vector over multiple runs, the result is an exponential decline in similarity (see Figure 6.3). Since the similarity between the position vectors is directly related to the transposition gradient, the exponential decline in similarity would be desired behaviour for a system such as the OSE model.

There is one issue with the random walk process though. By its very nature, no two random walks are identical. This poses a problem with the recall algorithm. Because of this, if the

[12]The random walk process is sometimes referred to as Brownian motion.

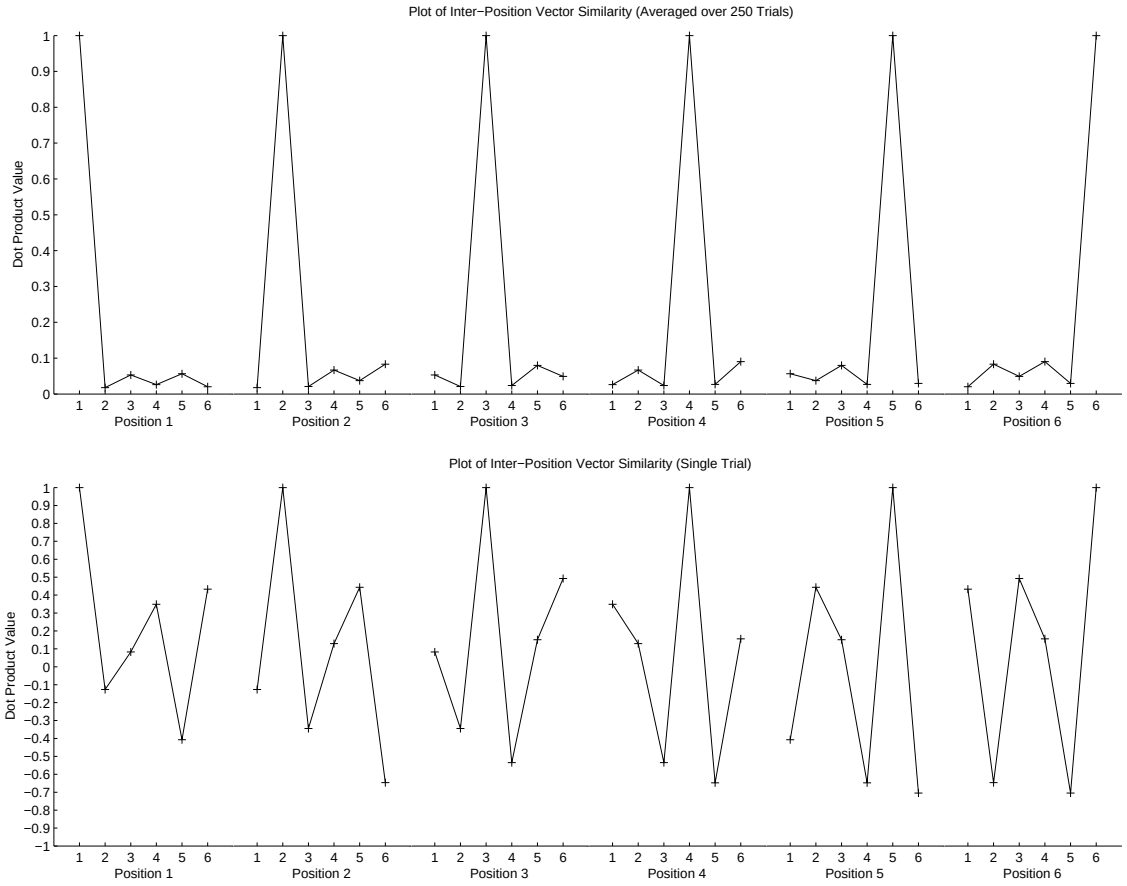

Figure 6.1: Plot of the vector similarities for position vectors generated using the convolution power method. (Top) Plot of the vector similarities averaged over 250 trials. (Bottom) Plot of the vector similarities for a single trial, chosen at random. Note that in this graph, the similarity value can take on a value less than 0 (i.e. highly dissimilar).

system were to only store the starting vector, there would be no way to reconstruct any of the position vectors used to encode the items. The solution to this is to use a pseudo-random walk algorithm. The concept behind this algorithm is simple. First, generate a vector – which shall be referred to as the "reference vector" – that consists of elements chosen from a normal distribution. Then, add this vector to the start vector to generate a new position vector. Next, rearrange the reference vector in a predetermined manner. To generate another position vector, add the permuted reference vector to the previously generated position vector, then permute the reference vector by the same predetermined manner as before. By using this method, a different set of random values is always added to the position vector, simulating the random walk process. To investigate the potential feasibility of this method, the average similarity comparison between each position vector was computed for 250 runs and graphed in Figure 6.4. As with the

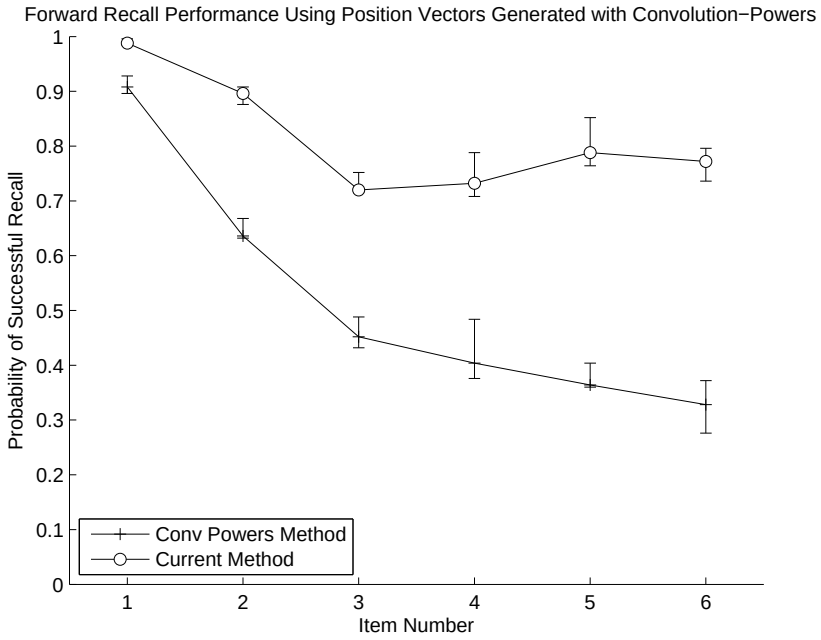

Figure 6.2: Comparison of the recall performance of the OSE model using the current method of position vector generation with the convolution powers method of position vector generation. The results show that the convolution powers method is unable to match the recall performance of the current method.

convolution powers, the results of a single run is also shown.

The results presented in Figure 6.4 show that the pseudo-random walk process seems more promising than the convolution powers method. Averaged over multiple runs, the desired gradual decrease in the similarity is observed. The single run data shows that unlike the convolution power method, the similarity of the position vectors generally decreases as the distance from the original position increases. As with the convolution power method, this method of generating position vectors was also tested using a non-spiking implementation of the OSE model for a six-itemed list. The results of this simulation are presented in Figure 6.5.

The results of the simulation show that while the overall recall accuracy of the model is decreased, the primacy and recency characteristics are consistent with the current method of generating the position vectors; that is, by using randomly generated HRR vectors as position vectors. Increasing the dimensionality of the vectors used in the model (also shown in the figure), results in a recall accuracy that more closely matches that of the current implementation of the OSE model. Further testing will be required to determine if the performance in other recall tasks can also be reproduced.

One advantage to using this method is that all of the components to this method have already been discussed. The pre-generated reference vector, which is a vector containing elements chosen

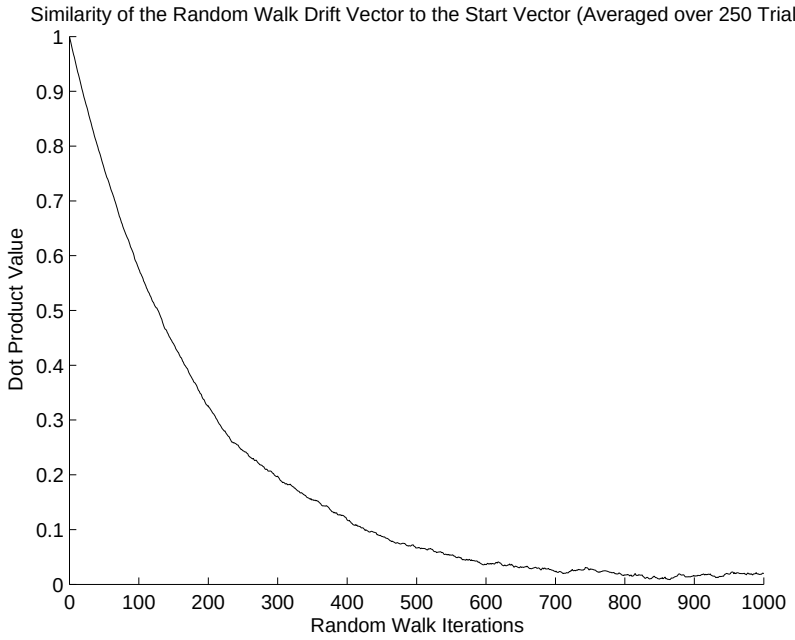

Figure 6.3: Demonstration of the effect the random walk process has on the similarity measure between the random walk (drift) vector and the starting vector. For the simulation, the similarity measures of 250 random walk trials were averaged. In each trial, the drift vector was allowed to drift 1000 randomly chosen steps away from the starting vector.

from a normal distribution, is in fact also an HRR vector. Additionally, the permutation operation can be computed by multiplying with a permutation matrix, in the same way that the involution operation – which is a special case of the permutation operation – is computed. This method of generating the position vectors also adds the ability to perform recency judgements on items. The judgement of recency task [Hinrichs, 1970] is one where subjects are presented a list and a test item, and then instructed to guess the position of the test item from the end of the list without first recalling the entire list. Using the OSE model, it would be trivial to retrieve the position vectors for both the last item and the test item. If the pseudo-random walk had been used to generate the position vectors, computing the similarity of the last item's position vector and the test item's position vector would give some measure of how close the two items were – a high similarity result would indicate the items are close together in the list, and a low similarity result would indicate otherwise.

6.2 Grouping

The grouping effect is characterized by a step-like serial recall curve when subjects are indirectly encouraged to group the list items (see Figure 6.6). Subjects are encouraged to group the list

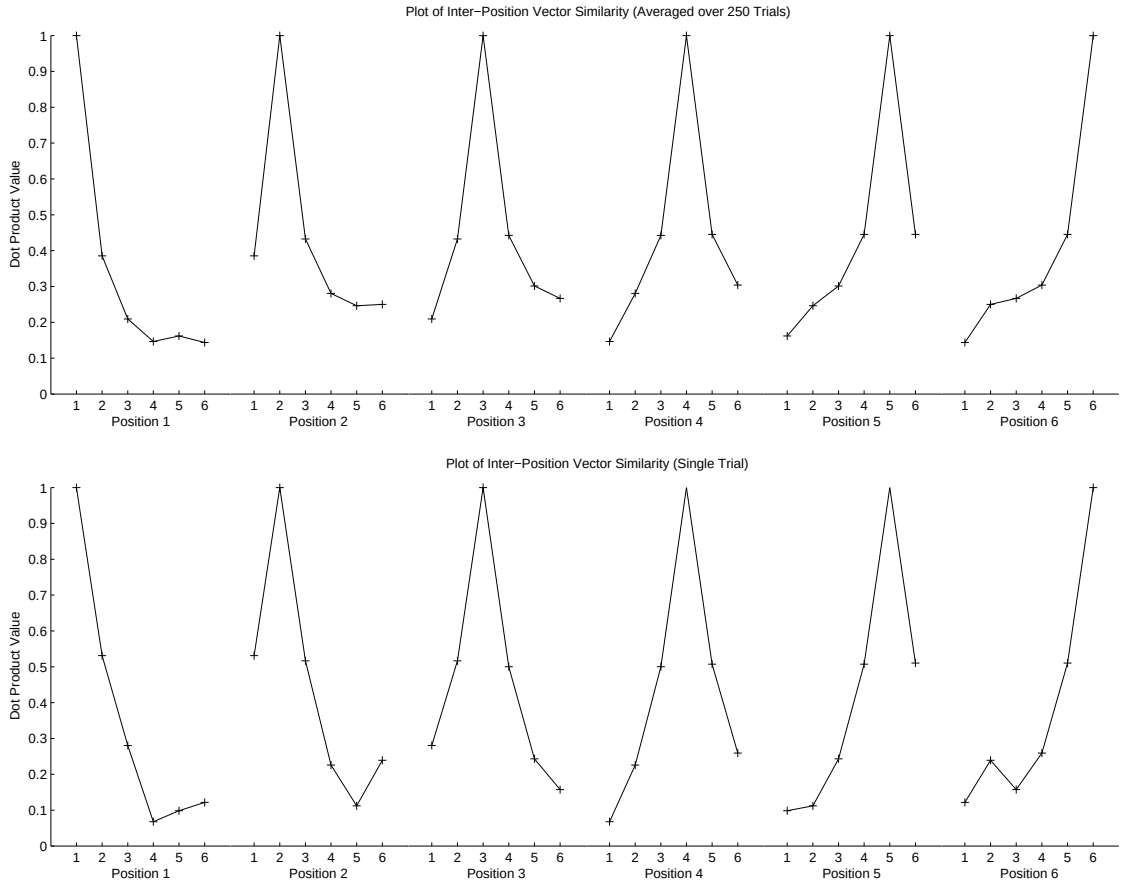

Figure 6.4: Plot of the vector similarities for position vectors generated using the pseudo-random walk process. (Top) Plot of the vector similarities averaged over 250 trials. (Bottom) Plot of the vector similarities for a single trial, chosen at random.

items by introducing a larger-than-normal inter-item interval (compared to the inter-item interval between items within a group).

The OSE model does not currently facilitate the grouping of the items; however, there is one easy way in which grouping can be accomplished in the OSE model. Since the an encoded memory trace can be manipulated like an item vector; the grouping effect can be achieved by encoding each group within a memory trace, and then applying the encoding operation to the memory traces of each group, treating each memory trace like an item in the "super-list". As an example, if the list $[\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}]$ were to be grouped as $[[\mathbf{A}, \mathbf{B}], [\mathbf{C}, \mathbf{D}]]$, then the encoding process will proceed as follows. For simplicity, let $OSE(\ldots)$ represent the result of applying the OSE

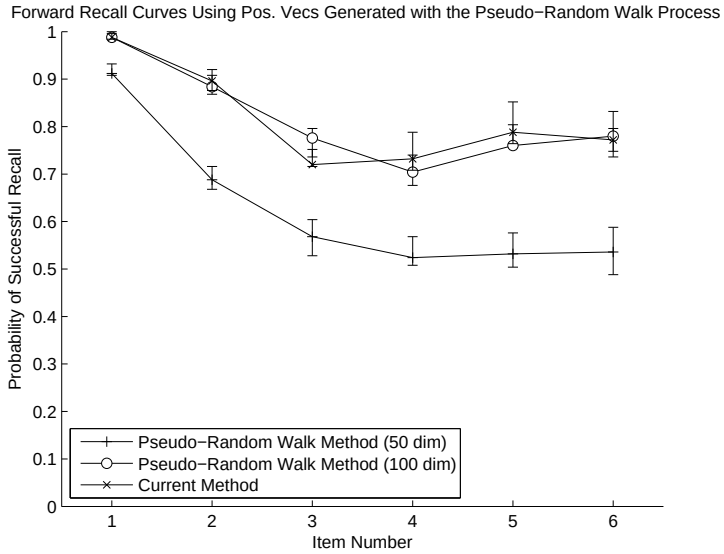

Figure 6.5: Comparison of the recall performance of the OSE model using the current method of position vector generation with the pseudo-random walk method of position vector generation. The results show that the pseudo-random walk method is able to reproduce the general primacy and recency phenomena seen in the current implementation of the OSE model, and for the case where the vector dimensionality has been increased to 100, the recall performance matches that of the current model.

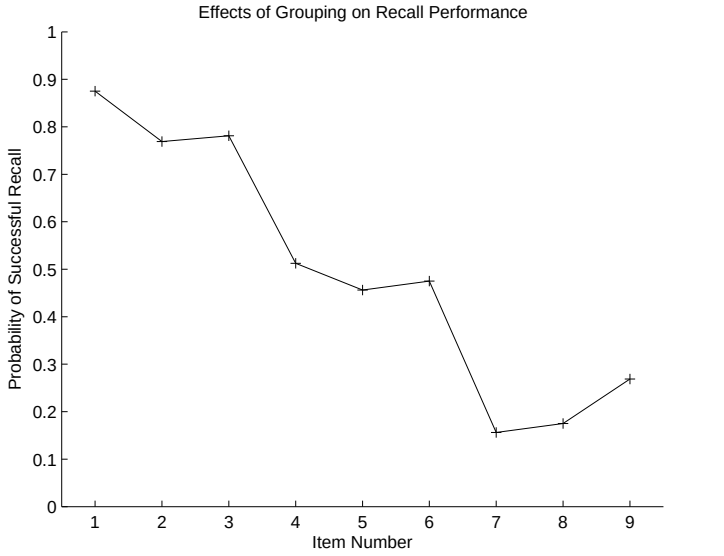

Figure 6.6: Illustration of the stair-like effect that grouping has on the recall performance in immediate forward recall tasks. The data illustrated in this figure is from [Hitch et al., 1996, Experiment 1]. In the experiment, subjects were prompted to group the list items into three groups.

encoding scheme to a list of item vectors. Then,

$$\begin{aligned} \text{The first group is encoded as:} \quad & \mathbf{M}_{AB} = OSE(\mathbf{A}, \mathbf{B}), \\ \text{The second group is encoded as:} \quad & \mathbf{M}_{CD} = OSE(\mathbf{C}, \mathbf{D}), \\ \text{And the entire list is encoded as:} \quad & \mathbf{M}_{ABCD} = OSE(\mathbf{M}_{AB}, \mathbf{M}_{CD}) \end{aligned}$$

There are a few implementation details that need to be worked out, however. For example, should the grouping mechanism be applied to the episodic buffer and input buffer memory traces separately; or should the grouping mechanism be applied to the combined memory trace of both components? Further investigation will be required to determine how best to use the grouping concept described above to reproduce the behaviour seen in human data.

6.3 Free Recall

The task of free recall is similar to the task of serial recall, with one notable difference. In free recall, subjects are not constrained to recall items in a specific order, but rather, they can recall the items in *any* order they wish. It would seem, at first glance, that the results of free recall would be much simpler to reproduce as no positional information is required in free recall. However, one cannot simply add the item vectors together to form a memory trace, as it would be difficult to retrieve the constituent vectors. This section thus explores a possible method of extending the OSE model to encompass free recall as well as serial recall.

Extending the OSE model for free recall involves adding the item information into the encoding process. The encoding equations then become:

$$\begin{aligned} \mathbf{M}_i^{\text{in}} &= \gamma \mathbf{M}_{i-1}^{\text{in}} + (P_i \circledast I_i) + I_i, \\ \mathbf{M}_i^{\text{epis}} &= \rho \mathbf{M}_{i-1}^{\text{epis}} + (P_i \circledast I_i) + I_i, \\ \mathbf{M}^{\text{OSE}} &= \mathbf{M}^{\text{epis}} + \mathbf{M}^{\text{in}} \end{aligned}$$

Although the exact recall strategy employed by human subjects in free recall is not known, for the OSE model, it is hypothesized that items with the strongest contribution to the memory trace are recalled first. The recall strategy that would be implemented in the OSE model would then be:

1. Identify and retrieve the item with strongest contribution.
2. Suppress the retrieved item, and loop back to the first step until all items have been recalled.

6.3.1 Retrieving the Item with the Strongest Contribution

In the original serial-recall encoding, retrieving a specific item was simple. All that needed to be done was to feed the result of the unbinding operation into cleanup memory. Performing the same process with the free recall encoding is possible, however more care needs to be employed when doing so. The serial-recall encoding relied on the unbinding operation to isolate the desired item vector from the superfluous vectors. In the free-recall encoding, no such operation is needed because the item information is already present in the encoding. However, this means that the chance of having multiple item vectors with strong contributions to the memory trace is high, and the cleanup memory module might return a combination of all these vectors, rather than just the desired vector. The chance of this occurring can be reduced by introducing a winner-take-all circuit that would increase the distinctiveness of each vector by means of mutual inhibition. With mutual inhibition, strong contributions suppress weaker contributions, and eventually, through an iterative process, only the strongest contribution would remain.

6.3.2 Response Suppression

Response suppression is needed in the free-recall encoding scheme because the item with the strongest contribution would still have the strongest contribution after recall if not suppressed. This would then lead to the erroneous recall of that item again. The method of response suppression is similar to the solution proposed for the deblurring problem of the TODAM model (refer to Section 2.3). This method involves the maintenance of a record of the retrieved items and subtracting this record from the original memory trace before feeding it through cleanup memory. If "$WTA(\ldots)$" is used to represent the winner-take-all operation mentioned in the previous section, the recall procedure can then be written as:

$$I_i = cleanup\left[WTA(\mathbf{M}^{\mathrm{OSE}} - \mathbf{R}_{i-1})\right] \tag{6.1}$$

$$\mathbf{R}_i = \mathbf{R}_{i-1} + I_i \tag{6.2}$$

While this recall method does prevent the erroneous recall of the same item twice, it will not handle the scenario where duplicate items are encoded in the same memory trace. Neither will this simple response suppression technique replicate the human response suppression behaviour, which indicate that erroneous repetitions are very rare and when they do occur, they are usually separated by three or four items, as reported in [Lewandowsky, 1999]. In order to reproduce this effect, a more complex response suppression algorithm will need to be developed and this is beyond the scope of this paper.

6.3.3 Immediate Free Recall

Unlike the serial recall task, the free recall task has been extensively studied and there is no shortage of data from human studies (e.g. [Postman and Phillips, 1965, Murdock, 1962, Rundus, 1971]).

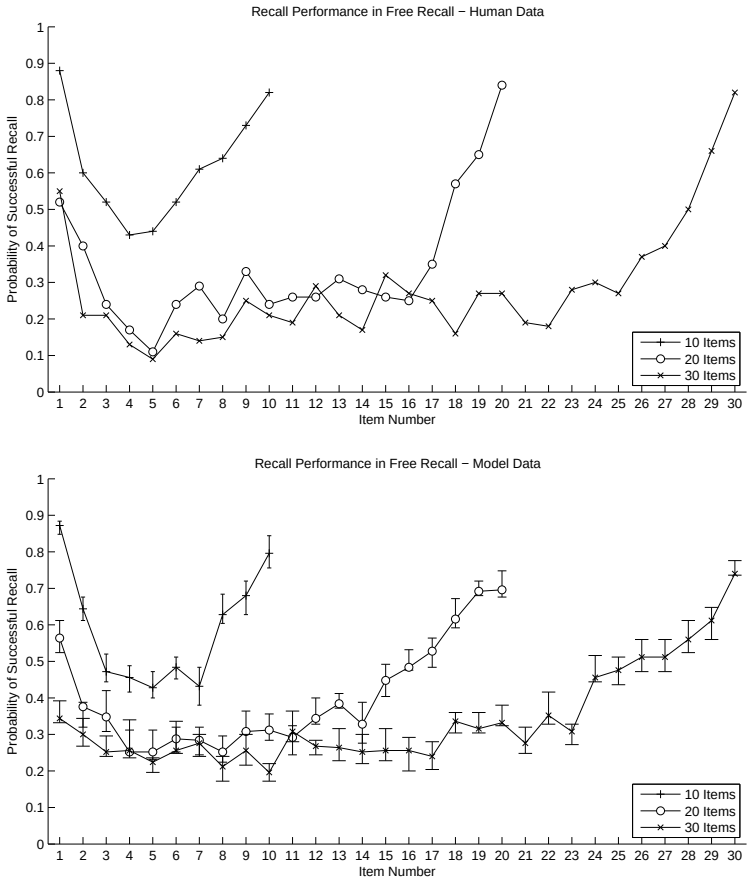

Figure 6.7: Comparison of the free recall data gathered from human subjects (from [Postman and Phillips, 1965]), and the free-recall enabled OSE model. The results demonstrate that even without changing any of the model parameters, it is still able to reproduce the general behaviour of the human data.

This section compares a non-spiking implementation of the OSE model with free-recall encoding to data reported in the three articles mentioned. As this new encoding scheme is merely a proof of concept, an ideal implementation of the winner-take-all algorithm was used in the simulations below. However, it is important to note that all of the model parameters – the episodic buffer parameter, the decay parameter, and the dimensionality of the item vectors – remained unchanged. Figure 6.7 illustrates the comparison between the model simulation and the data collected in one of the aforementioned articles, and as the results demonstrate, the OSE model with free-recall encoding is able to reproduce the curves seen in the human recall data for all of the list lengths shown. Whether this behaviour holds true for a spiking neuron implementation of the OSE model remains to be seen.

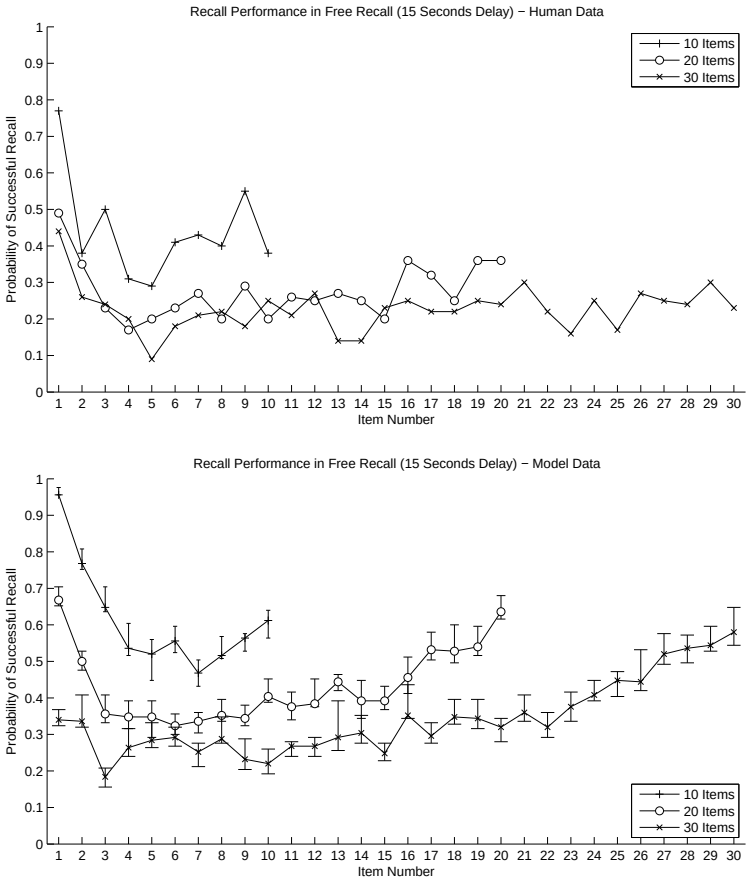

Figure 6.8: Comparison of the delayed free recall data gathered from human subjects (from [Postman and Phillips, 1965]), and the free-recall enabled OSE model. In the experiment, as with the model simulation, recall was delay by 15 seconds. Analysis of the data show that the model is able to capture the reduction of recency in the recall probabilities, in comparison with to the immediate recall scenario (see Figure 6.7). A discussion on the larger-than-data recency see in the model simulation is provided in the text.

6.3.4 Delayed Free Recall

Since the recall accuracy of the OSE model is temporally sensitive, it is also possible to test the OSE model with the task of delayed free recall. Figure 6.8 compares the implementation of the model described in the previous section with data reported in [Postman and Phillips, 1965]. The results of the simulation show that while the recency effect seems a little large, the model is able to reproduce the serial position curves for all of the various list lengths tested. An investigation into the larger-than-data recency effect reveals that it is most likely because of the simplistic method used to simulate rehearsal within the model. Because rehearsal is currently implemented as a static multiplier on the memory trace, for long lists, the effect of vector normalization due to

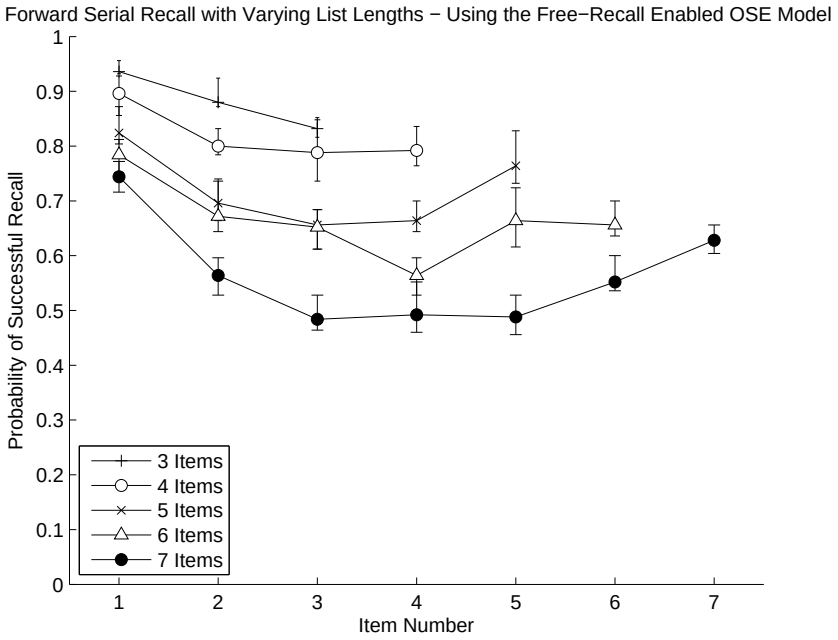

Figure 6.9: Plot of the recall accuracies produced by the free-recall enabled OSE model for the task of immediate forward recall. The simulation results show that although additional information is encoded within the memory trace, the model is still able to reproduce the primacy and recency phenomena observed in human recall data.

the limited representation range of the neural population overcomes the rehearsal multiplier. This means that the earlier list items do not contribute to the overall memory trace as much as later list items, causing the recency effect. This suggests, as has been suggested before, that a more complex model of rehearsal – involving a dynamic rehearsal process – will need to be developed in the future. However, the effect of normalization versus item contribution may help explain why recency effects are seen in long-term memories. It is logical to think that for long-term memories, the older items in the memory trace have been compressed so much that the newer items "stand out" more clearly.

6.3.5 Re-examining Serial Recall

This section re-examines the task of serial recall using the free-recall enabled OSE encoding scheme. Figure 6.9 illustrates the result of performing the immediate forward recall task with the free-recall enabled OSE model. As the simulation results show, even with the additional item information embedded in the memory trace, the model is still able to reproduce the general strong primacy and weak recency effects seen in human forward recall data; implying that the free-recall enabled encoding scheme is more general and better suited for use in future models.

Chapter 7

Conclusions

7.1 Future Work

The OSE model presents many avenues for future work. First and foremost, both the standard OSE encoding and the free-recall enabled OSE encoding require further testing and improvement to account for the other factors that influence the recall accuracy of the models. List presentation modality effects, the suffix effect, and word-length effects are just a few examples of other types of behavioural data that have yet to be accounted for in the OSE model. Apart from further testing to account for all of these different behaviour effects, several key components of the model can also be improved or implemented in a different manner. These improvements are briefly discussed below.

Rehearsal

As mentioned earlier, the method by which rehearsal is simulated in the OSE models is very simplistic. The consequence of the simplicity in the rehearsal mechanism is evident in the recall accuracies for the delayed recall tasks, where the primacy effect is much larger than reflected by the data. One possible improvement to the rehearsal mechanism is to simulate the rehearsal procedure actively. This would involve decoding the memory trace to retrieve each item, and then attempting to re-encode each item within the inter-item interval. In order to simulate the rehearsal process accurately, sub-vocal rehearsal times for each item need to be well defined, and this could possibly be where effects like the word-length effect can be accounted for. The primacy model [Page and Norris, 1998] provide several ideas on how this rehearsal mechanism can be implemented.

Dynamic Cleanup Memory

Perhaps one of the most pressing concerns about the current spiking neuron implementation of the OSE model is the implementation of the cleanup memory. In its current form, each item from the vocabulary needs to be "hard-coded" into the cleanup memory network before each simulation is run. It is hard to imagine that if cleanup memory were implemented in the brain, it would be implemented in such a manner. Instead, a dynamic cleanup memory, where vectors could be "loaded" and "unloaded" from the vocabulary, is more plausible. It is envisioned that a learning-enabled neural population could be used to implement the dynamic cleanup memory, but this has yet to be tested.

A context dependent dynamic cleanup memory could also explain the release-from-proactive-interference effect [Gregg, 1986] seen in human memory experiments. This effect is characterized as a gradual decline in recall performance in consecutive trials containing items from a similar category. Recall performance drastically improves if the item category is changed. Accounting for this effect using a context dependent dynamic cleanup memory is simple. As the memory model is presented with multiple recall trials, increasing numbers of items get loaded into the cleanup memory vocabulary, increasing the chance the cleanup memory will make a mistake, causing a decrease in recall performance. However, when the context is changed, the cleanup memory vocabulary is emptied, which would then result in an increase in recall performance.

Learning

Learning is another human behaviour that the OSE model currently fails to capture. It is currently unknown what is the best way forward with this problem. Perhaps, as a start, inspiration could be drawn from the original TODAM model [Lewandowsky and Murdock, 1989], which implements learning using a "closed-loop" version the model. In this version of the TODAM model, the memory trace encoded following the completion of one memory trial is not cleared before the next trial begins. Instead, the memory trace is used as a foundation on which more information is added on.

Hippocampal Involvement

Perhaps one of the biggest changes to the OSE model would be to alter the way information is encoded in the episodic buffer. From the behaviour and functionality of the episodic buffer, it is foreseeable that the entire component could be replaced with a neural network implementation of the hippocampus. Hippocampal involvement and effects of hippocampal damage to serial recall performance has already been observed in rats [Kesner and Novak, 1982] as well as human subjects [Brown et al., 2007]. As it has been extensively studied, there are already many hippocampal models that can be used to test the viability of this idea. For example, Hasselmo

and Wyble's hippocampal model [Hasselmo and Wyble, 1997], on which the free recall task has already been tested, provides a good foundation on which this work can proceed.

7.2 Closing Remarks

This thesis has presented a VSA-based model of serial recall which has been demonstrated to capture many of the facets of the human serial recall data. The model presented is also a very simple model, consisting of only 2 free parameters, both of which were independently derived and fixed for all of the recall tasks performed on the model. In effect, the model presented was able to reproduce human behavioural data without the need for parameter fitting – a feat that most models of serial recall are unable to do.

However, perhaps the most important question has yet to be answered thus far: "Why use spiking neurons?" It seems logical that with the encoding and decoding equations, it would be possible to construct a purely mathematical network to test the recall performance of the model. This is indeed the case, and the simulations were done early in the research process to test the viability of the model quickly. Truthfully, the OSE model was almost discarded because of this. Figure 7.1 illustrates the recall accuracies of the OSE encoding scheme (without free recall) in which the memory trace vector, $\mathbf{M}_{OSE}$, has either been left unnormalized, or has been normalized using the standard normalization technique.[13] As the figure shows, the serial recall curves do not resemble the human data at all, and it is only with the normalization effects intrinsic to a neural implementation is the model able to reproduce the human recall data.

This may be able to explain why a neural network model is needed, but still does not answer the question of why spiking neurons were necessary. Although not explored in this thesis, a spiking neuron model allows for the comparison of spike data collected from the model to spike data collected from real-world experiments (e.g. [Warden and Miller, 2007]). The spiking model also provides better temporal resolution in the integrators used in the OSE model, which in turn result in more realism in the storage behaviour of the most crucial part of any memory model: the memory modules.

[13] The standard normalization technique involves normalizing the vector such that the normalized vector has a magnitude of 1.

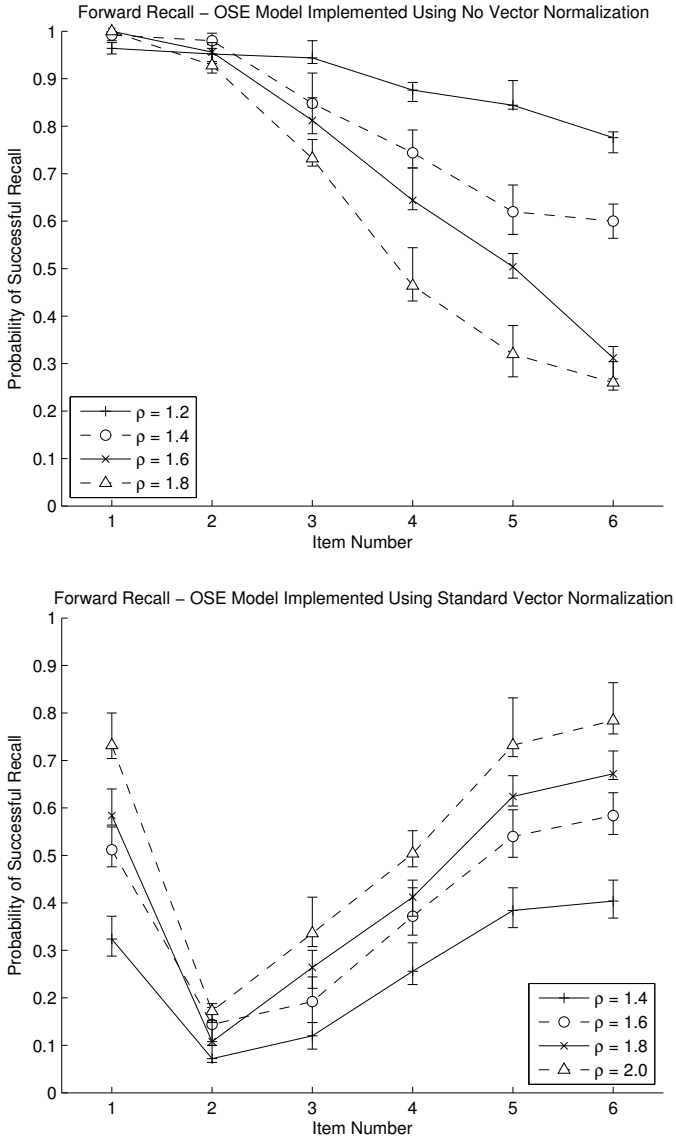

Figure 7.1: Plot of the immediate forward recall performance of the OSE model implemented using normalization techniques other than that intrinsic to a neural implementation. Vector normalization in the current OSE model occurs in virtue of the limited representational range of the neural populations. (Top) Recall data of the OSE model implemented with no vector normalization used. As demonstrated in the figure, decreasing the episodic buffer parameter has no effect in changing the general shape of the recall curve, leaving the model unable to match the recall performance of neither the current OSE model, nor human data. (Bottom) Recall data of the OSE model implemented with the standard vector normalization technique. As the figure shows, no matter how much the episodic buffer parameter is increased, the recall performance fails to match neither the current OSE model, nor human recall data.

References

[Anderson and Matessa, 1997] Anderson, J. R. and Matessa, M. (1997). A production system theory of serial memory. *Psychological Review*, 104(4):728–748.

[Atkinson and Shiffrin, 1968] Atkinson, R. C. and Shiffrin, R. M. (1968). Human memory: a proposed system and its control processes. *Psychology of Learning and Motivation*, 2:89–195.

[Averbeck et al., 2002] Averbeck, B. B., Chafee, M. V., Crowe, D. A., and Georgopoulos, A. P. (2002). Parallel processing of serial movements in prefrontal cortex. *PNAS*, 99(20):13172–13177.

[Baddeley, 2000] Baddeley, A. D. (2000). The episodic buffer: a new component of working memory? *Trends in Cognitive Sciences*, 4(11):417–423.

[Baddeley and Hitch, 1974] Baddeley, A. D. and Hitch, G. (1974). Working memory. In Bower, G. A., editor, *The Psychology of Learning and Motivation*, pages 47–89. Academic Press.

[Botvinick and Plaut, 2006] Botvinick, M. M. and Plaut, D. C. (2006). Short-term memory for serial order: a recurrent neural network model. *Psychological Review*, 113(2):201–233.

[Brown et al., 2007] Brown, G. D. A., Sala, S. D., Foster, J. K., and Vousden, J. I. (2007). Amnesia, rehearsal, and temporal distinctiveness models of recall. *Psychonomic Bulletin & Review*, 14(2):256–260.

[Drewnowski and Murdock, 1980] Drewnowski, A. and Murdock, B. B. (1980). The role of auditory features in memory span for words. *The Journal of Experimental Psychology: Human Learning and Memory*, 6(3):319–332.

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

[Gayler, 2003] Gayler, R. W. (2003). Vector symbolic architectures answer jackendoff's challenges for cognitive neuroscience. In Slezak, P., editor, *ICCS/ASCS International Conference on Cognitive Science*, CogPrints, pages 133–138, Sydney, Australia. University of New South Wales.

[Georgopoulos et al., 1986] Georgopoulos, A. P., Schwartz, A., and Kettner, R. E. (1986). Neuronal population coding of movement direction. *Science*, 233:1416–1419.

[Goldman-rakic, 1995] Goldman-rakic, P. S. (1995). Cellular basis of working memory. *Neuron*, 14:477–485.

[Gregg, 1986] Gregg, V. H. (1986). *Introduction to human memory*. Routledge & Kegan Paul, London.

[Hasselmo and Wyble, 1997] Hasselmo, M. E. and Wyble, B. P. (1997). Free recall and recognition in a network model of the hippocampus: simulating effects of scopolamine on human memory function. *Behavioural Brain Research*, 89:1–34.

[Henson, 1998] Henson, R. N. A. (1998). Short-term memory for serial order: the start-end model. *Cognitive Psychology*, 36:73–137.

[Henson et al., 2000] Henson, R. N. A., Burgess, N., and Frith, C. D. (2000). Recoding, storage, rehearsal and grouping in verbal short-term memory: an fmri study. *Neuropsychologia*, 38:426–440.

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

[Hinrichs, 1970] Hinrichs, J. V. (1970). A two-process memory-strength theory for judgment of recency. *Psychological Review*, 77(3):223–233.

[Hitch et al., 1996] Hitch, G. J., Burgess, N., Towse, J. N., and Culpin, V. (1996). Temporal grouping effects in immediate recall: a working memory analysis. *The Quarterly Journal of Experimental Psychology A*, 49(1):116–139.

[Jahnke, 1968] Jahnke, J. C. (1968). Delayed recall and the serial-position effect of short-term memory. *Journal of Experimental Psychology*, 76(4):618–622.

[Kesner and Novak, 1982] Kesner, R. P. and Novak, J. M. (1982). Serial position curve in rats: role of the dorsal hippocampus. *Science*, 218:173–175.

[Lee and Estes, 1977] Lee, C. L. and Estes, W. K. (1977). Order and position in primary memory for letter strings. *Journal of Memory and Language*, 16:395–418.

[Lewandowsky, 1999] Lewandowsky, S. (1999). Redintegration and response suppression in serial recall: a dynamic network model. *International Journal of Psychology*, 34(5/6):434–466.

[Lewandowsky and Li, 1994] Lewandowsky, S. and Li, S.-C. (1994). Memory for serial order revisited. *Psychological Review*, 101(3):539–543.

[Lewandowsky and Murdock, 1989] Lewandowsky, S. and Murdock, B. B. (1989). Memory for serial order. *Psychological Review*, 96(1):25–57.

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

[Madign, 1971] Madign, S. A. (1971). Modality and recall order interactions in short-term memory for serial order. *Journal of Experimental Psychology*, 87(2):294–296.

[McCormick et al., 1985] McCormick, D. A., Connors, B. W., Lighthall, J. W., and Prince, D. A. (1985). Comparative electrophysiology of pyramidal and sparsely spiny stellate neurons of the neocortex. *Journal of Neurophysiology*, 54(4):782–806.

[Murdock, 1961] Murdock, B. B. (1961). The retention of individual items. *Journal of Experimental Psychology*, 62(6):618–625.

[Murdock, 1962] Murdock, B. B. (1962). The serial position effect in free recall. *Journal of Experimental Psychology*, 64:482–488.

[Murdock, 1979] Murdock, B. B. (1979). Convolution and correlation in perception and memory. In Nilsson, L.-G., editor, *Perspectives on memory research*, pages 105–119. Lawrence Erlbaum Associates, Hillsdale, New Jersey.

[Murdock, 1982] Murdock, B. B. (1982). A theory for the storage and retrieval of item and associative information. *Psychological Review*, 89(6):609–626.

[Murdock, 1983] Murdock, B. B. (1983). A distributed memory model for serial-order information. *Psychological Review*, 90(4):316–338.

[Murdock, 1992] Murdock, B. B. (1992). Serial organization in a distributed memory model. In Healy, A. F., Kosslyn, S. M., and Shiffrin, R. M., editors, *From Learning Theory to Connectionist Theory*, pages 201–225. Lawrence Erlbaum Associates, Hillsdale, New Jersey.

[Murdock, 1993] Murdock, B. B. (1993). Todam2: a model for the storage and retrieval of item, associative and serial-order information. *Psycholocial Review*, 100(2):183–203.

[Murdock, 1995] Murdock, B. B. (1995). Developing todam: three models for serial-order information. *Memory & Cognition*, 23(5):631–645.

[Nairne, 1990] Nairne, J. S. (1990). Similarity and long-term memory for order. *Journal of Memory and Language*, 29:733–746.

[Orlov et al., 2000] Orlov, T., Yakovlev, V., Hochstein, S., and Zohary, E. (2000). Macaque monkeys categorize images by their ordinal number. *Nature*, 404:77–80.

[Page and Norris, 1998] Page, M. P. A. and Norris, D. G. (1998). The primacy model: a new model of immediate serial recall. *Psychological Review*, 105(4):761–781.

[Peterson and Peterson, 1959] Peterson, L. R. and Peterson, M. J. (1959). Short-term retention of individual verbal items. *Journal of Experimental Psychology*, 58(3):193–198.

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

[Postman and Phillips, 1965] Postman, L. and Phillips, L. W. (1965). Short-term temporal changes in free recall. *Quarterly Journal of Experimental Psychology*, 17(2):132–138.

[Rauch et al., 2003] Rauch, A., La Camera, G., Luscher, H.-R., Senn, W., and Fusi, S. (2003). Neocortical pyramidal cells respond as integrate-and-fire neurons to in vivo-like input currents. *Journal of Neurophysiology*, 90(3):1598–1612.

[Reitman, 1974] Reitman, J. S. (1974). Without surreptitious rehearsal, information in short-term memory decay. *Journal of Verbal Learning and Verbal Behavior*, 13:365–377.

[Rundus, 1971] Rundus, D. (1971). Analysis of rehearsal processes in free recall. *Journal of Experimental Psychology*, 89(1):63–77.

[Smolensky, 1990] Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence*, 46:159–216.

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

[Talmi et al., 2005] Talmi, D., Grady, C. L., Goshen-Gottstein, Y., and Moscovitch, M. (2005). Neuroimaging the serial position curve. *American Psychological Society*, 16(9):716–723.

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