An Integrated Model of Context, Short-Term, and Long-Term Memory

by

Jan Gosmann

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2018

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner	Marc Howard Professor
Supervisor	Chris Eliasmith Professor
Internal Member	Bryan Tripp Associate Professor
Internal-external Member	Jeff Orchard Associate Professor
Internal-external Member	Sue Ann Campbell Professor

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the 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.

Statement of Contributions

Chapter 10 (excluding Section 10.4) paraphrases a conference submission that was co-authored by myself, a PhD student Aaron R. Voelker, and my supervisor, Dr. Chris Eliasmith (Gosmann, Voelker, and Eliasmith 2017). Appendix C is a verbatim copy from the supplementary material accompanying the same paper. I implemented the network models, performed the benchmarks, and data analysis. Mr. Voelker contributed the mathematical analyses.

Abstract

I present the context-unified encoding (CUE) model, a large-scale spiking neural network model of human memory. It combines and integrates activity-based short-term memory with weight-based long-term memory. The implementation with spiking neurons ensures biological plausibility and allows for predictions on the neural level. At the same time, the model produces behavioural outputs that have been matched to human data from serial and free recall experiments. In particular, well-known results such as primacy, recency, transposition error gradients, and forward recall bias have been reproduced with good quantitative matches. Additionally, the model accounts for the effects of the acetylcholine antagonist scopolamine, and the Hebb repetition effect.

The CUE model combines and extends the ordinal serial encoding (OSE) model, a spiking neuron model of short-term memory, and the temporal context model (TCM), a mathematical model of free recall. To the former, a neural mechanism for tracking the list position is added. The latter is converted into a spiking neural network under considerations of the main features and simplification of equations where appropriate. Previous models of the recall process in the TCM are replaced by a new independent accumulator recall process that is more suited to the integration into a large-scale network. To implement the modification of the required association matrices, a novel learning rule, the association matrix learning rule (AML), is derived that allows for one-shot learning without catastrophic forgetting. Its biological plausibility is discussed and it is shown that it accounts for changes in neural firing observed in human recordings from an association learning experiment. Furthermore, I discuss a recent proposal of an optimal fuzzy temporal memory as replacement for the TCM context signal and show it to be likely to require more neurons than there are in the human brain.

To construct the CUE model, I have used the Neural Engineering Framework (NEF) and Semantic Pointer Architecture (SPA). This thesis makes novel contributions to both. I propose to distribute NEF intercepts according to the distribution of cosine similarities of random uniformly distributed unit vectors. This leads to a uniform distribution of active neurons and reduces the error introduced by spiking noise considerably in high-dimensional neuronal representations. It improves the asymptotic scaling of the noise error with dimensions d from $O(d)$ to $O\left(d^{3/4}\right)$. These results are applied to achieve

improved Semantic Pointer representations in neural networks are on par with or better than previous methods of optimizing neural representations for the Semantic Pointer Architecture. Furthermore, the vector-derived transformation binding (VTB) is investigated as an alternative to circular convolution in the SPA, with promising results.

Acknowledgements

I would like to thank my supervisor, Chris Eliasmith, for his continued support and guidance. I am also grateful to my colleagues at the Centre for Theoretical Neuroscience, in particular Terrance Stewart and Aaron Voelker, for their invaluable knowledge and helpful discussions. The research in this thesis would have not been possible without the excellent and continuously improving work of the Nengo development team, spearheaded by Trevor Bekolay. This research was also enabled in part by support provided by SHARCNET[1] and Compute Canada[2]. Many CPU cycles were burned on their hardware, simulating iterations of the CUE model. I also would like to acknowledge the financial support for this research by the Canada Research Chairs program, the NSERC Discovery grant 261453, the Air Force Office of Scientific Research grant FA8655-13-1-3084, CFI, and OIT.

A special mention is deserved by the Interdisciplinary College, a unique annual spring-school at the Lake Möhne in Germany. It is the place where I met Chris Eliasmith for the first time and ultimately made me pursue this research direction.

Finally, I would like to thank all the people I climbed with over the years as they helped me to stay sane during the work on this thesis.

[1] www.sharcnet.ca

[2] www.computecanada.ca

Dedication

This is dedicated to my parents, for everything they gave me.

Table of Contents

List of Tables

List of Figures

List of Symbols

$\boldsymbol{A}$	activity matrix
a	neural spiking activity
α	NEF neuron gain
$\mathcal{B}$	binding operator
B	beta function
β	TCM beta parameter
$\mathcal{B}_{\mathrm{V}}$	vector-derived transformation binding operator
$\boldsymbol{c}$	TCM context vector
$\boldsymbol{c}^{\mathrm{IN}}$	TCM input context vector
$\mathcal{CS}$	cosine similarity distribution
$\boldsymbol{D}$	NEF decoder matrix
$\boldsymbol{d}$	NEF decoding vector
d	dimensionality
$\boldsymbol{e}$	NEF encoding vector
E_{d}	distortion error
E_{n}	noise error
$\boldsymbol{E}$	NEF encoder matrix
E	error
E_{tot}	total error
$\mathbb{E}$	expected value
$\mathcal{F}$	Fourier transform
$\boldsymbol{f}$	TCM item vector
$\boldsymbol{f}^{\mathrm{IN}}$	recalled TCM item vector
G	neuron nonlinearity
Γ	gamma function
γ	OSE short term decay
h	synaptic filter
Θ	Heaviside function
$\boldsymbol{I}$	identity matrix
$\boldsymbol{i}$	identity under binding
i	imaginary unit
J^{bias}	NEF bias input current
δ	Kronecker delta
λ	regularization scale
$\boldsymbol{M}^{\mathrm{CF}}$	TCM context to item matrix
$\boldsymbol{m}^{\mathrm{epis}}$	OSE episodic memory trace
$\boldsymbol{M}^{\mathrm{FC}}$	TCM item to context matrix
$\boldsymbol{m}^{\mathrm{stm}}$	OSE short-term memory trace
μ	null choice bias in recall
$\mathcal{N}$	normal distribution
$\boldsymbol{n}$	absorbing element under binding
$\mathbb{N}_{>0}$	positive natural numbers (excluding zero)
O	big O notation
Ω	solid angle
$p_{\mathcal{CS}}$	PDF of cosine similarity distribution
ϕ	distractor rate
r	NEF representational radius
ρ	OSE episodic scaling
$\mathcal{S}$	superposition operator
s	similarity measure
σ	standard deviation of noise in recall
τ_{RC}	membrane time constant

τ_{ref}	refractory period		perball
τ_{syn}	synaptic time constant	$\boldsymbol{W}$	synaptic weight matrix
$\boldsymbol{T}_{\mathcal{F}}$	discrete Fourier transform matrix	$\boldsymbol{X}$	evaluation point matrix
		$\boldsymbol{x}$	evaluation point
V_d	volume of a d-dimensional hy-	$\mathcal{X}$	NEF representational space

1. Introduction

> If you have no memory, how can you be certain of anything?
>
> *(Monkey in Kubo and the Two Strings)*

Memory in its different forms is an important aspect of human and animal cognition. It allows such agents to return to previously visited water and food locations (Vorhees and Williams 2014), allows them to act more optimally in situations similar to prior experiences, or allows them to avoid dangerous situations. In this way, memory allows for adaptation to a complex environment on a faster time scale than genetic selection. This is especially important in unstable and changing environments. In humans, memory is also important for forming social relationships, a shared culture, and even a functioning society. For example, strategies in the repeated prisoner's dilemma[1] depend on working memory capacity (Milinski and Wedekind 1998). Moreover, memory contributes to our individual sense of self (Prebble, Addis, and Tippett 2013).

In addition to these functional reasons, an addressable memory is also important from a computational perspective. It allows for a more compact implementation of many computational processes than a pure state machine could achieve (Gallistel and King 2009). Overall, the storage of information over time is a fundamental requirement in many cognitive systems.

Accordingly, there is a long history of memory research. The well-known primacy and recency effects (discussed in more detail in Section 1.2) have already been described by Robinson and M. A. Brown (1926). Nevertheless, there are

[1] The scenario of the prisoner's dilemma poses that two subjects are arrested, but the evidence is not sufficient to convict them on the principle charge. If both subjects stay silent, they each get a prison sentence of one year. Either subject can decide to betray the other in which case they get no prison sentence while the other subject gets a sentence of three years. However, if both subjects decide to betray each other, each gets a prison sentence of two years. The subjects are not allowed to communicate, but in the repeated prisoner's dilemma they can penalize each other for previous decisions.

still many open questions regarding memory function and implementation in the brain. One important challenge that memory systems have to solve is the so-called *stability-plasticity dilemma* (Abraham and A. Robins 2005). On the one hand, there is a need to quickly form new memories, sometimes even with a single exposure (also known as *one-shot learning*). On the other hand, such high plasticity can easily lead to overwriting of old memories, rendering memory systems useless. Buzsáki (1989) and others have proposed multi-stage memory models to address this dilemma, where different memory systems are in place for different timescales with different levels of plasticity.

Despite this, much experimental research and modeling treat different memory systems as isolated. This simplifies the analysis of results to an extent, but to get a general understanding of memory as a whole our characterization of memory phenomena needs to be integrated at some point. Furthermore, many models of memory or learning focus on either small scale neural changes without any direct connection to behaviour (e.g., Levy, Hocking, and Wu 2005), or on idealized behaviour described with mathematical equations, but no solid grounding in biological plausibility (e.g., Milford, Wyeth, and Prasser 2004). So it is not only important to integrate our understanding of memory systems, but also to bridge the gap from neural mechanisms to behaviour. With this thesis I attempt to advance our understanding of memory in these ways by proposing a model integrating short- and long-term memory, thus modeling their interaction. Moreover, I implement the model as a spiking neural network to ensure biological plausibility, while at the same time matching behavioural data.

While this work is still limited with regard to the variety of memory systems covered, there are promising long-term prospects for a better understanding of human memory. Many forms of memory loss are currently untreatable. However, diseases associated with aging, like Alzheimer's, significantly impair the function of memory systems and are getting more common as our life-span increases due to medical and nutritional advances. A better understanding of memory might allow us to devise better treatments, or even stop and reverse the memory detoriation. A potential route to such treatments is through memory implants of the kind already demonstrated in rats by Berger et al. (2011). For these sort of implants, an understanding of how memories are encoded is at least helpful, if not crucial.

These are certainly strong motivators for research into memory, but it is also worthwhile to advance our general understanding of how the human

brain works. An important step in testing our current understanding of the brain is the Spaun model (Eliasmith, Stewart, et al. 2012). Spaun is a spiking neural network of 2.5 million neurons, grounded in biology, that can perform 8 different tasks. It gets sensory input (low resolution black and white symbols) and produces a behavioural motor output with a simulated arm. The number of tasks it can perform demonstrates that it is not a specialized model for a single task, but can switch between different tasks. Obviously, Spaun is still much simpler (and has many fewer neurons) than an actual brain, and is still far from capturing the complexity of a real brain. But it incorporates a number of qualitative key aspects, such as the ability to switch tasks, making human-like errors, and being implemented in an anatomically constrained spiking neural network. However, one key aspect is still missing. While it can perform working memory dependent tasks and has simple reinforcement learning, it is missing a declarative and episodic long-term memory. The work in this thesis can also be seen as a step toward implementing a model of such memory for the future integration in even larger scale models, combining further key aspects of cognition within a single model.

1.1. Behavioural characterization of memory

Memory systems have been characterized in different and sometimes contradictory ways. However, one commonly used distinction is made along two orthogonal dimensions: by timescale and by type of information stored. On the timescale axis one can differentiate between *short-term memory (STM)* lasting for up to tens of seconds and *long-term memory (LTM)* which can last between hours to decades (Chaudhuri and Fiete 2016). However, there is no single agreed upon definition of the exact boundaries. Sometimes the term *very long-term memory (vLTM)* is used in addition to STM and LTM for memory that exceeds LTM (Solso 1998).

Another use of these terms defines STM as memory being maintained through sustained neural firing, while LTM and vLTM are realized by synaptic-weight changes. The volatility of sustained neural firing explains why in this use of terms, STM usually corresponds to information maintained on shorter timescales than in LTM. Moreover, the term working memory (WM) is often used to refer to active representations that allow direct mental manipulation. In this thesis, I will use the terms STM and LTM primarily to distinguish between

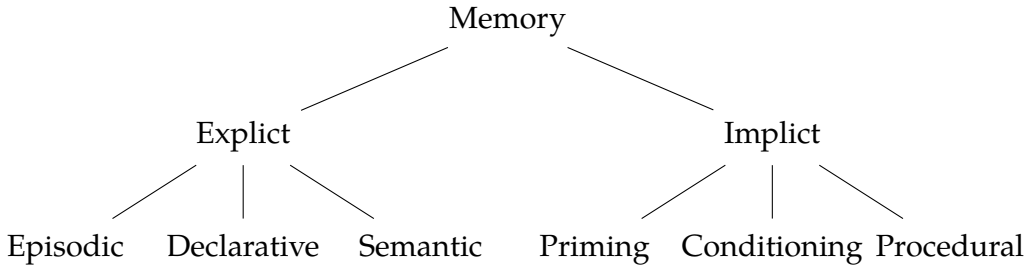


Figure 1.1.: Categorization of memory by type of stored information.

activity-based and weight-based forms of memory.

When classifying memory by type of information, a representation as a tree structure can be helpful (Fig. 1.1). On the highest level, we have the distinction between implicit and explicit memory. Implicit memory does not allow for conscious access. A typical example is procedural or motor memory like the exact muscle contractions for keeping balance when riding a bike. Explicit memory allows for conscious access and is further subdivided into declarative and episodic memory. The declarative memory allows us to store and reproduce facts like the birthday of a friend, whereas the episodic memory provides a recollection of life experiences. Here, I will focus on declarative memory as this type of memory has been well-studied in many memory experiments.

1.2. Experimental findings in memory research

Many memory experiments require the participants to memorize lists of words and recall them afterwards. Usually an experiment will fall into one of two categories depending on how the subjects are required to recall the list items: in free recall experiments, the subjects are free to recall the items in any order; in serial recall experiments, the subjects are required to recall the items in the order they were presented.

The hallmark finding in memory research, especially for serial recall experiments, are the *primacy* and *recency* effects. Subjects tend to remember the start of list (primacy) and the end of list (recency) better (Fig. 1.2). Furthermore, if subjects in a serial recall experiment recall an item at the wrong position, a so-called *transposition*, it is more likely to be an item that was close in the list than an item several positions away.

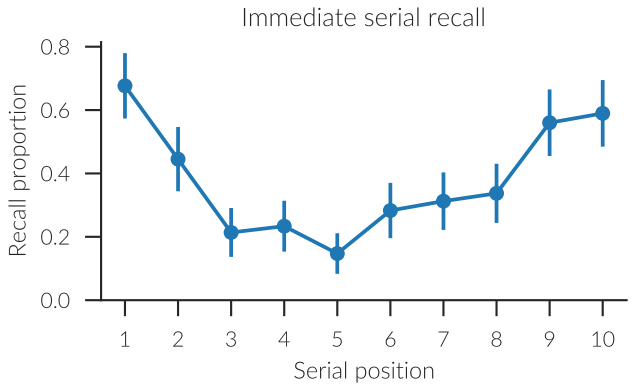


Figure 1.2.: Serial position curve of a immediate serial recall experiment with a 10 item list. Data reproduced from Jahnke (1968) with 95 % confidence intervals.

In free recall experiments, a recency effect is observed as well. This does not only show in the serial position curve, but participants often start out with recalling the last item first. This can be measured by the *probability of first recall* (Fig. 1.3). Another important aspect of free recall is captured by the conditional response probability (CRP). It gives the probability for the difference (the *lag*) in serial positions of two recalled items. It is peaked around zero, indicating that items in proximity in the learned list tend to be recalled together, while jumps to remote items are rarer. This is known as *contiguity* or the *lag-recency effect*. The CRP curve also has a characteristic asymmetry which shows the bias for forward (opposed to backward) recall.

By introducing a delay filled with a distractor task, referred to as *delayed (free) recall*, both the probability of first recall and the CRP curve will become much flatter. This effect cannot solely be attributed to (suppressed) rehearsal effects as introducing an equally long distractor interval in-between the list items, known as *continuous distractor (free) recall*, partially restores the recency effect in the probability of first recall.

While there are many other experimental findings in memory research, there are two more findings that I will focus on in this thesis. The first is the differential effects of the acetylcholine antagonist scopolamine on encoding and recall (Ghoneim and Mewaldt 1975). Recall performance is normal when scopolamine is injected after the presentation phase, but the number of successful recalls is considerably lower when the injection is done before the presentation

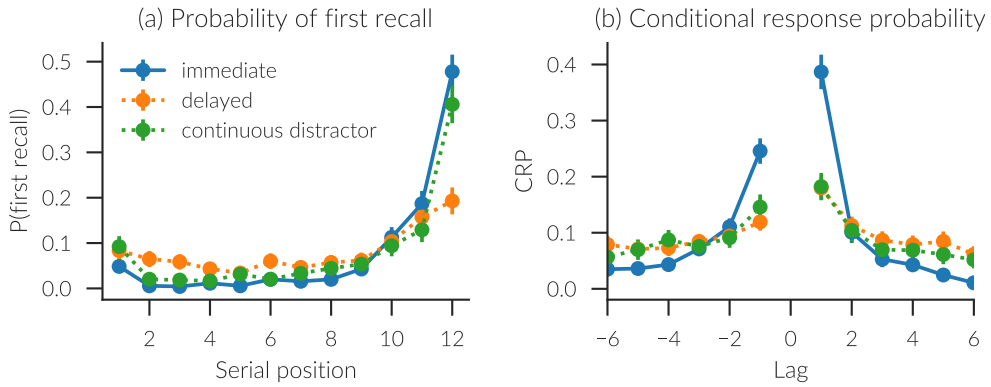


Figure 1.3.: Data from free recall experiments with 12 item lists. (a) Probability of first recall. (b) Conditional response probability. All data from Howard and Kahana (1999) with 95 % confidence intervals.

phase. The number of successfully recalled items in the latter case is around the short-term memory span, which might show an effect purely on LTM, but not STM. This experiment is of interest as a neural network model is more suited to modeling such drug effects than typical mathematical models.

Second, the *Hebb repetition effect* is the observation that in repeated immediate recall experiments, the recall accuracy of a repeated list (typically every third list) will increase with repetitions (Hebb 1961). Whether the test subject has to become consciously aware of the repetition is debated (Stadler 1993). It is of interest in this thesis that the Hebb repetition effect has been regarded as useful in testing the interaction of STM and LTM, or as Burgess and Hitch (2005) phrase it, this effect is a "powerful vehicle for developing and testing models of the relationship between STM and LTM".

1.3. Neuroanatomy of memory

Short-term memory is attributed to cortical brain regions, but there is no single region that is the unique locus of STM. Sensory short-term memory is distributed across the corresponding cortical sensory and related brain regions (Zelano et al. 2009; Todd and Marois 2004; Baldo, Katseff, and Dronkers 2012). For example, auditory memory can be found in the temporal cortex close to speech processing areas, whereas visual short-term memory is located in

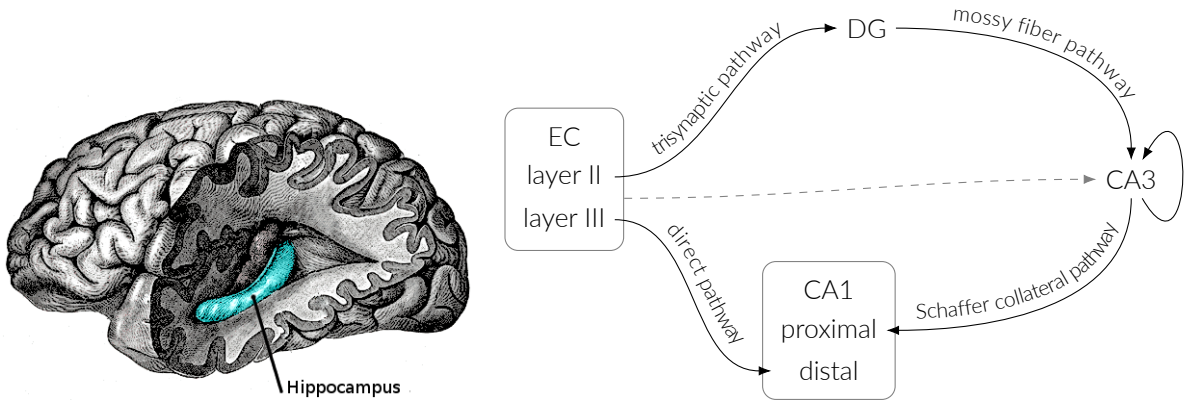


Figure 1.4.: (a) Location of the hippocampus in the human brain. (b) Connectivity between entorhinal cortex (EC) and hippocampal subregions: dentate gyrus (DG), CA1, CA3.

parietal cortex. Furthermore, multimodal integration and manipulation in working memory has been found to lead to increased activation in prefrontal cortex (Rypma et al. 1999).

Compared to short-term memory, the formation of new long-term memories is more localized. In particular, the hippocampus (HC) has been implicated in the acquisition of new declarative and episodic memories (Eichenbaum 2001). A finding originally derived from patients with hippocampal lesions, most famously HM who got his hippocampus (but also other brain regions) removed due to severe epilepsy (Penfield and Milner 1958; Scoville and Milner 1957; Squire 2009). The hippocampus is named for its sea horse shaped structure. It is located in the medial temporal lobe (Fig. 1.4) and lies near the subiculum and entorhinal cortex (EC). The hippocampus is further divided into substructures CA3, CA1, and the dentate gyrus (DG) by its cytoarchitectural structure. While the CA3 and CA1 regions consist mainly of pyramidal neurons and about 10 % GABAergic, inhibitory interneurons (Freund and Buzsáki 1996), granule cells are the main constituent of the dentate gyrus. Furthermore, the dentate gyrus has a large number of cells that exhibit sparse activity compared to CA3. In humans the cell count of the DG is about twice as high as in CA3, whereas in rats DG has about five times more cells than the CA3 region.

The connectivity to, from, and within hippocampus is exceptionally well

known. There are two major pathways from entorhinal cortex. The *direct pathway* originates in layer III of the entorhinal cortex and targets the distal apical dendrites of CA1 neurons. The *trisynaptic pathway* originates in layer II and leads via the dentate gyrus, and CA3 also to CA1, but targeting the proximal dendrites (a signal thus crosses three synapses). The connections from the dentate gyrus to the CA3 region are termed the *mossy fiber pathway*. Each of these mossy fibers innervates about 15 neurons (Claiborne, Amaral, and Cowan 1986) which results in each CA3 pyramidal neuron receiving input from about 50 to 90 granule cells (Squire, Shimamura, and Amaral 1989, p. 230). The final bundle of connections from CA3 to CA1 is known as the *Schaffer collateral pathway*. Further major hippocampal connections exist from the entorhinal cortex to CA3, mostly targeting the same cells as the connection from dentate gyrus (Paxinos 2014), and recurrent connections in CA3. However, a single cell in CA3 is not recurrently innervated by more than 5% of all CA3 cells (Squire, Shimamura, and Amaral 1989, p. 231).

Given these neuroanatomical properties, some of the hippocampal regions have been assumed to be involved in specific tasks. The sparse coding and large cell count of the dentate gyrus led to the belief that it is responsible for pattern separation (Rolls 2013). The CA3 region with its recurrent connectivity could be involved in pattern completion or forward predictions (Guzowski, Knierim, and Moser 2004; S. Leutgeb and J. K. Leutgeb 2007; Rolls 2013). Additional evidence for this comes from unimpaired recognition memory after lesioning the CA3 to CA1 connections despite impairing selective recall (Brun et al. 2002).

A review of hippocampal structures would not be complete without mentioning hippocampal place cells and entorhinal grid cells (Hafting et al. 2005), first discovered in rats, but existent also in other animals (Buzsáki and Moser 2013). A grid cell fires when the animal is in locations arranged in a regular hexagonal grid in an environment. Place cells are similar, but fire only for one specific location in an environment and remap between environments. While these cells are involved in navigational tasks, a connection to memory is possible (Buzsáki and Moser 2013). However, paying further attention to these cells is out of the scope of this thesis.

Closely related to grid and place cells is the hippocampal theta oscillation of 4 Hz to 8 Hz found in rats. The spiking of place cells during a theta cycle is timed according to the distance from corresponding land marks, a phenomen termed *phase precession* (O'Keefe and Recce 1993). The coupling of the theta and gamma bands has also been shown to be important in the learning of item-

context associations (Tort et al. 2009). Despite some debate, there is evidence for similar, but slower (< 4 Hz), hippocampal oscillations in humans related to episodic memory encoding (Lega, Jacobs, and Kahana 2012). However, their functional relevance is not entirely clear at this point.

Finally, most commonly during sleep, but also in immobile rats, sharp waves (SPWs) are observed in the rat hippocampus (Chrobak and Buzsáki 1994; Girardeau et al. 2009). These have been implicated to be involved in memory consolidation from hippocampus to the neocortex, a process not covered in this thesis.

1.4. Memory models

Different memory models can be roughly sorted into three classes. Conceptual models describe different components and processes of memory and their interaction. Often they are presented as box and arrow diagrams. They do not allow for a precise mechanistic explanation or quantitative predictions. Mathematical models address this by providing exact equations that can be evaluated to obtain quantitative predictions. However, they do not explain the neural implementation and thus are not constrained to biologically plausible mechanisms. Finally, connectionist models use neural networks with varying degrees of abstraction. As such their biological plausibility also varies. While those models come closer to explaining neural mechanisms of memory, they less often address the behavioural data from high-level cognitive experiments.

1.4.1. Conceptual models

Yntema and Trask (1963) presented one of the first models of memory. They conceptualized retrieval as a search process where each item in memory is associated with tags referencing further information. For example, a time tag would encode the time an item was observed. The whole description, however, is more based on how a memory system could be implemented on a classical von Neumann computer and does not consider if or how those operations could be neurally implemented.

Atkinson and Shiffrin (1968) developed another early memory model consisting of a sensory memory, short-term store, and long-term store. Sensory information enters the short-lived sensory memory, before it is (potentially

partially) transmitted to the short-term store. This store can hold information indefinitely with rehearsal, but without it is assumed to be lost within thirty seconds. Furthermore, items in the short-term store can be transferred to and retrieved from the long-term store. The model is very general and does not provide many constraints or detailed mechanisms apart from this basic distinction.

To date, the most influential conceptual organization of working memory was proposed by Baddeley (1986). He proposed, based on experimental data, separate stores for visual and acoustic information, termed the *visuospatial sketchpad* and *phonological loop* respectively. These are controlled by a *central executive*. Furthermore, in Baddeley (2000) the model was extended with an episodic buffer for the binding of multimodal information and transfer to episodic long-term memory. The main relevance of the model is that it informs us about the organization of (working) memory by modality; but it does less to elucidate mechanisms.

In general, conceptual models can give a high-level account and they can be evaluated with respect to their qualitative agreement to behavioural data. However, they cannot provide us with quantitative predictions which makes a more rigorous validation difficult. They, also, do not explain the cognitive mechanisms in detail, and an account of the neural implementation is completely out of their reach.

1.4.2. Mathematical models

Some of the weaknesses of conceptual models are addressed by mathematical models. These models make quantitative predictions about the behavioural data and describe underlying cognitive mechanisms to a varying degree, but do not provide a neural implementation. There is a vast number of such models and not all can be discussed here. Thus I will focus on some of the most influential ones.

Such a list certainly contains the perturbation model for serial order by Estes (1972), the free recall model Search of Associative Memory (SAM) by Raaijmakers and Shiffrin (1981), the recognition memory model Retrieving Effectively from Memory (REM) by Shiffrin and Steyvers (1997), and the episodic memory model MINERVA2 by Hintzman (1988). The perturbation model is an early attempt to provide a mathematical framework for how remembered item positions can drift over time. In SAM, cues are assembled in a short-term memory

to retrieve associations from a long-term associative memory (the search part of the model). In the REM, model error prone copies of feature vectors derived from the study items are stored. A recognition probe is matched to the stored feature vectors and a likelihood ratio of the match scores being generated by an old versus a new item is calculated. The MINERVA2 model also uses feature vectors that are stored as traces and can be probed by cues.

All of these models, and most other mathematical models, either assume item-to-item or position-to-item associations. The primacy model by Page and Norris (1998) is worth noting because it uses a different approach. In that model, items are activated according to a primacy gradient, but the model is agnostic as to how this gradient is generated. This, however, also leaves that aspect underspecified.

Common to all of these models is that they do not consider biological plausibility. Thus it is unclear whether any of the models can be implemented in a neural substrate while preserving their predictions. Or, similarly what the limitations with regard to noise and requirements for neural resources are. Nevertheless, mathematical modelling is an important first step in figuring out what sort of processes are worth considering for a neural implementation. Also, there are some reoccurring ideas in these models that are useful to consider in the context of a neural model. For example, starting with J. A. Anderson (1973) many models have used random feature vectors to represent individual items. That approach is similar to the Semantic Pointer Architecture (SPA) presented in Chapter 5 which can be implemented neurally with the methods of the Neural Engineering Framework (Chapter 3).

An especially influential model based on such random feature vectors was TODAM2 (Murdock 1993). It was able to fit a large body of experimental data and, in that regard, aims to be a general theory for item recognition, serial order, and associative memory. A neural implementation could very well be possible with the NEF, but has not been attempted so far. However, Choo (2010) pointed out that the dimensionality of the vectors in the model increases with each stored item due to the use of convolution powers. Thus, the requirement for neural resources grows in an unbounded manner. Even when using circular convolution in the model, a neural implementation might be problematic because the vector norm can grow without bound (Section 5.1.3). It is also worth noting, that TODAM2 does not give an account of how responses are generated.

Another useful idea, that had its origin in mathematical models, is the

idea of a randomly drifting context signal that items get associated to. Estes (1955) presented the first model of this type and Murdock (1997) extended the TODAM2 model in this way to explain additional data. Two open questions in these sort of models are, (1) how is the context at an earlier time re-instantiated to start the recall; and (2) how the context is advanced in the same way during recall as in the study phase to recall the remaining items? As the signal drifts randomly, a memory for the context signal itself would be needed. The OSCAR model (G. D. A. Brown, Preece, and Hulme 2000) solves part of this by using a deterministic context signal that is generated from multidimensional oscillators. This allows replay of the exact same context signal once it has been reset. It still does not describe how the context is re-instantiated to start the recall, as this still requires knowledge of the oscillator states at the beginning of the study phase.

All of these context-based models cannot explain the asymmetric CRP curves. The temporal context model (Howard and Kahana 2002), however, was specifically constructed to explain these free recall data. It also uses a context signal, but this signal is updated by the studied (and recalled) items themselves. Each item recalls a prior context associated with that item to partially update the current context, and the updated context gets associated with the studied item. This solves the re-instantiation problem, as the right cue can set the context to be similar to the study context to retrieve an item. That retrieved item in turn updates the context to retrieve more items related to the updated context. Thus, it is also available to appropriately advance the context after each recall. The TCM model is discussed in more detail in Chapter 8. But it is not perfect. It did not capture immediate free recall data involving short-term memory, even though it was presented as a single-store framework, i.e. a single memory for both STM and LTM. This single-store assumption was criticized by Davelaar et al. (2008) and is in contradiction to neuroimaging data (Talmi et al. 2005).

Many of these models are also vague on the exact processes of recall. Though, the ACT-R model of serial recall by J. R. Anderson and Matessa (1997), in which items are associated with their serial positions, describes detailed steps necessary for recall. Unfortunately, it is vague on the exact storage mechanism of items.

Recently, Shankar and Howard (2013) proposed an *optimally fuzzy temporal memory* that is less motivated by behavioural data, but more by a mathematical optimal and scale-free storage of a time-varying signal. Nevertheless, it has been proposed to model the coding in hippocampus (Howard, MacDonald,

et al. 2014). In this framework, the input signal is stored by a bank of leaky accumulators with different time constants. This corresponds to a Laplace transform of the signal. To decode the history of the input signal, a linear operator approximating the inverse Laplace transform is used. I discuss this model in more detail in Section 13.2, and demonstrate that it is highly sensitive to noise, which is prohibitive to a neural implementation.

1.4.3. Connectionist models

Compared to the vast number of mathematical models, there are many fewer connectionist models, although most of those that exist try to ensure more biological realism. Many of these models focus on reproducing low-level findings in the hippocampus, such as sequence compression in replay (Levy, Hocking, and Wu 2005) or place cells (Milford, Wyeth, and Prasser 2004). The model presented by Hasselmo (2012), might very well be the most comprehensive hippocampus model to date, describing the storage of episodic memories as a spatial trajectory. It addresses experimental data on place cells and the theta rhythm. A very recent model by Yu et al. (2017), constructs a three-layer spiking neural network that is able to encode a sequence over several iterations and replay it during a cycle of the theta rhythm. These models, however, still leave a large gap to high-level cognitive behaviour as modelled by mathematical models. In addition, the Hasselmo (2012) relies on hypothetical "arc length" cells to disambiguate memories (cp. S. Robins 2015).

Nevertheless, there are some connectionist models that try to reduce this gap by addressing behavioural effects with a neural network implementation. Namely, Burgess and Hitch (1992) and Burgess and Hitch (1996) propose models for the articulary loop, Norman and O'Reilly (2003) for recognition and familiarity effects, and Botvinick and Plaut (2006) for immediate serial recall. All of these models use rate based neurons as an abstraction. While rate neuron models are a useful tool to build tractable models with a degree of biological realism, one has to be careful to not introduce biologically implausible features. For example, the noise introduced by discrete spikes is neglected. In particular, Norman and O'Reilly (2003) use a k-winner-take-all mechanism that is hard to realize in spiking neurons as it requires a fine balance of excitation and inhibition. Potentially even more problematic is the use of back-propagation learning in the model by Botvinick and Plaut (2006). While learning with back-propagation is a tremendously successful technique in machine learning,

it is still unclear whether biological neural networks can implement this sort of learning. This remains true despite new techniques like feedback-alignment (Lillicrap et al. 2016) and related work (Bengio et al. 2015), that might eventually provide biological plausible implementations.

To the best of my knowledge there are only two memory-related models that address these concerns about biological plausibility by using spiking neurons while at the same time connecting to behavioural data. The first one is the ordinal serial encoding (OSE) model of serial recall by Choo (2010). As a primarily short-term memory model, it uses recurrently connected neurons to store the memory trace in neural activity. This approach is also used to model a long-term memory component that can be attributed to hippocampal storage. While this allows for a first approximation, a storage in synaptic weights would be more plausible for such a component. The second model was specifically developed to model the storage of serial lists with hippocampus by Oliver Trujillo (2014). It is able to reproduce neural data like replay and theta rhythm. However, the length of stored lists is limited in similar fashion to the STM capacity in the OSE model, despite long term memory being certainly able to learn longer lists. Learning longer lists in the model would require the chaining of individual lists with different contexts, but no mechanism for this has been given in the model. Another questionable feature is the usage of a clock signal that is adjusted to speed up compressed replay.

1.4.4. Summary

Despite many existing models, a number of questions have not been sufficiently addressed (cp. Horwitz and J. F. Smith 2008). To date no model demonstrates a satisfying degree of biological plausibility while at the same time addressing behavioural data from cognitive psychology. Many models are not concerned with the interaction of short-term and long-term memory despite the importance for many fundamental effects on memory performance. This includes neural processes for the coordination and control of the interplay of these memory components. Finally, the recall process or reinstantiation of recall context is often not precisely explained even though it is an essential part of memory function.

I address these points with the context-unified encoding (CUE) model presented in this thesis. It combines activity-based short-term memory with weight-based long-term memory and also specifies the required control and

recall processes. Biological plausibility is ensured by an implementation as a spiking neural network. Despite the low-level implementation, it is validated against human, behavioural data.

This thesis is divided into two parts. In the first part, all the basic methods required for the construction of such a large-scale spiking neural network model are developed and discussed. This includes some secondary research objectives to improve neural representations to ultimately require fewer neurons for a higher simulation throughput. In the second part, the methods from the first part are employed to build up the CUE model and compare the model predictions against human data.

Part I.

Methods

2. Modeling neurons

The information processing in the brain is performed by neurons (see Fig. 2.1). Despite a wide variety of different neuron types and behaviours, the typical behavior of most neurons can be described as follows. These cells build up an electrical potential of about $-70\,\mathrm{mV}$, the *resting membrane potential*, across their cell membrane with the ions in the intra- and extracellular fluid. When the membrane potential is sufficiently depolarized (to about $-50\,\mathrm{mV}$), voltage gated ion channels trigger a sudden and short-lived depolarization (typically about 1 ms), generating a so-called action potential or spike. This action potential travels along the neuron's axon and triggers the release of neurotransmitters at the boutons, which open into synapses, where the axon comes extremely close to other neuron's dendrites. The released neurotransmitters will influence the ion channels of the post-synaptic neuron and either lower (inhibitory synapse) or raise (excitatory synapse) the membrane potential. The deflection of the membrane potential depends on the strength of the synapse and the speed of the release and uptake of the neurotransmitter. If the post-synaptic neuron receives sufficient input from other neurons, its membrane potential will be sufficiently deflected to initiate a new action potential. By the pattern of connectivity that controls what activity in some neurons triggers activity in other neurons, the brain is able to perform the computations that lead to an agent's behavior.

To model neurons computationally, it is necessary to chose a level of abstraction. On the one hand, it is possible to create very detailed models with spatial extent (e.g., Markram et al. 2015; Bahl et al. 2012) where individual ion channels and the propagation of electrical potentials is modeled. On the other hand, one can use very abstract neuron models that basically consist of nodes summing their inputs and applying non-linearity using real valued stand-ins for firing rates (instead of discrete spikes). This latter type of model is common in artificial neural networks and deep learning.

A widely used neuron model in computational neuroscience, that is midway between these two extremes, is the leaky integrate-and-fire (LIF) neuron model.

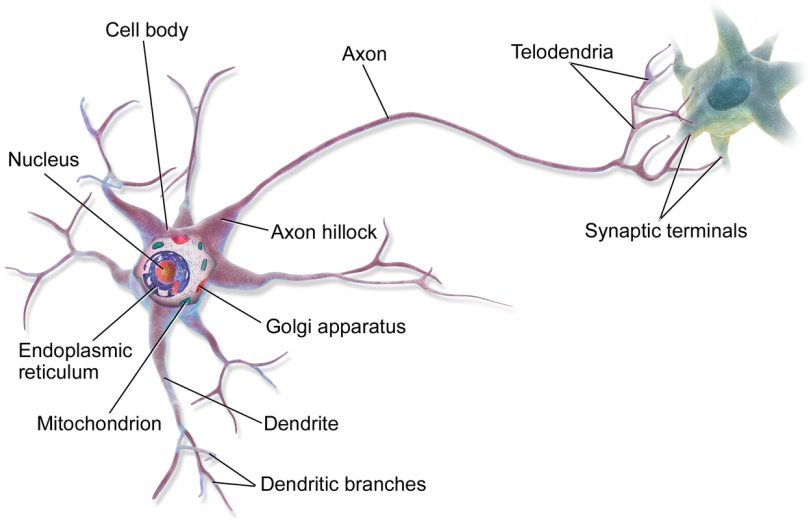


Figure 2.1.: A typical neuron. Image by Blausen Medical Communications, Inc. and reused under the Creative Commons Attribution 3.0 Unported license.

It is a point neuron model, thus not modeling any spatial extend. It models a single membrane voltage $V(t)$ described by the differential equation

$$\tau_{RC}\frac{\mathrm{d}V}{\mathrm{d}t} = -V(t) + RJ(t) \tag{2.1}$$

where R is the membrane resistance, $J(t)$ the input current, and τ_{RC} the membrane time constant. Whenever the membrane voltage reaches a certain threshold V_{th}, the LIF neuron transmits a spike and its membrane voltage is reset for a refractory period of τ_{ref}. This type of neuron model allows the firing rate to be derived for a given constant input current J as

$$a(J) = \frac{1}{\tau_{\text{ref}} - \tau_{RC}\ln\left(1 - \frac{V_{\text{th}}}{RJ}\right)}. \tag{2.2}$$

The LIF neuron model is a good choice for the undertaking of this thesis as it captures many aspects of neuron behaviour. It is detailed enough to relate parameters like the membrane time constant to the actual biological correspondents. This allows us to fix these parameters to values within a biologically

plausible range instead of having them as free parameters in the model that would require parameter matching and give the model additional degrees of freedom to match the data. At the same time, the model is simple enough to allow for reasonable performance when simulating large-scale models with these neurons.

3. The Neural Engineering Framework

To construct a large-scale spiking neural network with a certain behaviour, some method for obtaining that behaviour is required. In most cases this method will be a learning algorithm (e.g., O'Reilly and M. J. Frank 2006). However, this requires time-intensive training of the model and is often not viable for large models, complex behaviours, or models combining different behaviours. In this work I opt to use the Neural Engineering Framework (NEF; Eliasmith and C. H. Anderson 2003) which allows the direct construction of a spiking neural network from the mathematical equations describing the desired dynamics without the time-intensive training. As such the final model does not provide a developmental account of how the neural network became organized or learned to perform its task. But, it provides a biologically plausible explanation of how the developed brain might perform that task. As well, it allows for the inclusion of biologically plausible online learning rules, to test adaptation in the adult system. Furthermore, it allows for manipulations to test known experimental results in the model or obtain new predictions. The NEF consists out of the three core principles for *representation*, *transformation*, and *dynamics* in a neural network that I introduce in this order.

3.1. Representation

Neurons within a neural network will have a preferred stimulus: they will fire most strongly for that stimulus and less strongly as the stimulus gets more dissimilar to the preferred stimulus (see Fig. 3.1). To capture this in a mathematical description, we can treat the stimulus as a vector $\boldsymbol{x}(t)$ that varies over time. The preferred stimulus vector, that is the vector a neuron i fires most strongly for, will be denoted with $\boldsymbol{e}_i$. The spiking activity $a_i(t)$ of a neuron can then be described with

$$a_i(t) = G\Big[\alpha_i \langle \boldsymbol{e}_i, \boldsymbol{x}(t) \rangle + J_i^{\text{bias}}\Big] \tag{3.1}$$

where α_i is a neuron gain factor, J_i^{bias} a bias input current, and G the neuron nonlinearity. The nonlinearity G represents the neuron model and converts an input current into spikes. Usually this is the spiking, leaky integrate-and-fire model (LIF) discussed in Chapter 2, which provides a good trade-off of captured neuron behaviour, detail, and simulation effort. But simpler neuron models (e.g., a rate-based LIF model, or rectified linear units) could be used, as well as much more complex neuron models, like the compartmental models in Eliasmith, Gosmann, and Choo (2016) and Duggins (2017). The input current to the neuron is obtained from how well the stimulus aligns with the preferred stimulus as measured by the dot product. As this alignment with the preferred stimulus "encodes" the stimulus into the neural representational space, $\boldsymbol{e}_i$ is usually referred to as *encoder* in the context of the NEF. Furthermore, the gain factor α_i and bias term J_i^{bias} can be used to adjust the neuron's tuning curve to experimentally observed firing rates. However, it is also common to use higher maximal firing rates to use fewer neurons in simulations while achieving a similar accuracy. While this is not entirely adhering to biological constraints, in most cases NEF models behave similarly when lowering the firing rates and increasing neuron numbers accordingly (e.g., Gosmann and Eliasmith 2015). As it is rare to have detailed information about the tuning curves in many parts of the brain, those values are usually not directly set in the NEF. Instead a representational space $\mathcal{X}$ is defined, usually as the d-dimensional hyperball with radius r. Furthermore, for each neuron a maximum firing rate $a_{\max,i}$ and an intercept point p_i are sampled from random distributions. These values are used to calculate the gain and bias so that the neuron starts firing at $p_i \boldsymbol{e}_i$ when $\boldsymbol{x}$ varies along the encoder $\boldsymbol{e}_i$ and that the maximum $a_{\max,i}$ is not exceeded across the representational space $\mathcal{X}$.

Given a population of neurons, also called a neural *ensemble* in NEF terms, how can the encoded vector $\boldsymbol{x}(t)$ be recovered? First the activity or spike trains $a_i(t)$ are convolved with a synaptic filter h to obtain the induced post-synaptic voltage change. Usually this is a decaying exponential

$$h(t) = \tau_{\text{syn}}^{-1} \exp\bigl(-t/\tau_{\text{syn}}\bigr), \tag{3.2}$$

but other filters can be used to more precisely model the dynamics of the synapse (Voelker, Benjamin, et al. 2017) and even extensions to conductance-based synapses are possible (Stöckel, Voelker, and Eliasmith 2017). From the filtered activity, the represented vector can be reconstructed with a linear,

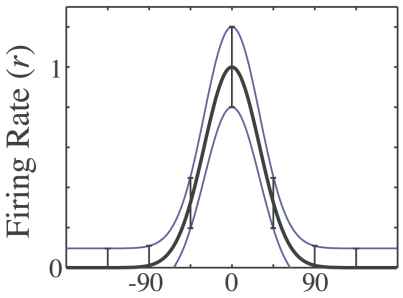


Figure 3.1.: A typical neuron tuning curve. The thick black line shows the mean firing rate in dependence of a stimulus parameter (e.g., rotation of a bar). The thin blue lines show the standard deviation. Figure reproduced from Butts and Goldman (2006) and reused under the Creative Commons Attribution license.

weighted decoding

$$\hat{\boldsymbol{x}}(t) = \sum_i \boldsymbol{d}_i \cdot [a_i * h](t) \tag{3.3}$$

with decoding weights $\boldsymbol{d}_i$.

To get a good reconstruction of the represented value, the decoding weights should minimize the error

$$E = \int_{\mathcal{X}} \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2 \, \mathrm{d}\boldsymbol{x}. \tag{3.4}$$

In general, this minimization cannot be solved analytically. Thus, in the NEF the integral is approximated by randomly sampling M *evaluation points* $\boldsymbol{x}_k \in \mathcal{X}$. Given the finite number of evaluation points, it becomes possible to solve for the decoding weights with a least-squares minimization. The detailed derivation is given in Eliasmith and C. H. Anderson (2003, Ch. 2). In short, one obtains the matrices

$$\boldsymbol{A} = \begin{bmatrix} a_1(\boldsymbol{x}_1) & a_1(\boldsymbol{x}_2) & \cdots & a_1(\boldsymbol{x}_M) \\ a_2(\boldsymbol{x}_1) & a_2(\boldsymbol{x}_2) & \cdots & a_2(\boldsymbol{x}_M) \\ \vdots & \vdots & \ddots & \vdots \\ a_N(\boldsymbol{x}_1) & a_N(\boldsymbol{x}_2) & \cdots & a_N(\boldsymbol{x}_M) \end{bmatrix} \text{ and } \boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_M \end{bmatrix} \tag{3.5}$$

where one can use the steady-state activities in the activity matrix $\boldsymbol{A}$ which can be obtained analytically for LIF neurons. Given these two matrices the decoding weights can be obtained with the regularized pseudo-inverse as

$$\begin{bmatrix} \boldsymbol{d}_1^\top \\ \boldsymbol{d}_2^\top \\ \vdots \\ \boldsymbol{d}_N^\top \end{bmatrix} = \left(\boldsymbol{A}\boldsymbol{A}^\top + M\lambda^2 \max(\boldsymbol{A})^2 \boldsymbol{I} \right)^{-1} \boldsymbol{A}\boldsymbol{X} \tag{3.6}$$

where λ is the regularization scale (usually $\lambda = 0.1$).

The encoders and decoders not only allow us to encode information into a neural ensemble and decode it back out, but also to transmit that information from one neural population to another. By decoding from the pre-synaptic ensemble and encoding into the post-synaptic ensemble, the connection weights required between the two populations can be obtained as

$$\boldsymbol{W}_{ij} = \boldsymbol{e}_i^\top \boldsymbol{T} \boldsymbol{d}_j \tag{3.7}$$

where $\boldsymbol{T}$ can describe a linear transform to implement across the neural connections. If $\boldsymbol{T}$ is the identity matrix, the connection will be a pure communication channel.

3.2. Transformation

To be useful, a neural network has to transform or compute functions on the represented information. In the NEF, it is straight-forward to implement a given transformation in the connection weights between two ensembles. To implement a function $f(\boldsymbol{x})$, one replaces the matrix $\boldsymbol{X}$ with

$$\boldsymbol{X}_{f(\boldsymbol{x})} = \begin{bmatrix} f(\boldsymbol{x}_1) \\ f(\boldsymbol{x}_2) \\ \vdots \\ f(\boldsymbol{x}_M) \end{bmatrix} \tag{3.8}$$

when solving for decoders. This corresponds to a minimization of the modified error $E_{f(\boldsymbol{x})} = \int_{\mathcal{X}} \left\| f(\boldsymbol{x}) - \hat{\boldsymbol{x}} \right\|^2 \mathrm{d}\boldsymbol{x}$.

Eliasmith and C. H. Anderson (2003, Ch. 7) shows that a neural network constructed in this way is typically best at computing low-order polynomials. Non-smooth or discontinuous functions might require a large number of neurons. In some cases a better function approximation can be achieved by appropriately selecting parameters like intercepts or encoders or by changing the network structure to decompose a function in a different way. An example is the calculation of products and has been discussed in Gosmann (2015), and Section 4.4 shows this for the thresholding of values.

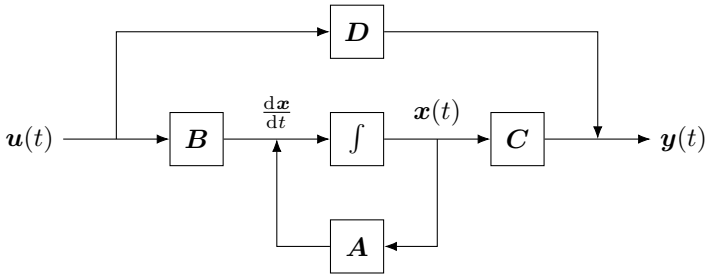


Figure 3.2.: Visualization of (3.9) and (3.10) as block diagram.

3.3. Dynamics

The final principle of the NEF addresses dynamics. In linear control theory a dynamical system is often described by state equations of the form

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) \tag{3.9}$$

$$\boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}\boldsymbol{u}(t) \tag{3.10}$$

where $\boldsymbol{x}(t)$ is the state vector, $\boldsymbol{u}(t)$ the input vector, and $\boldsymbol{y}(t)$ the output vector. The system behaviour is determined by the dynamics matrix $\boldsymbol{A}$, input matrix $\boldsymbol{B}$, output matrix $\boldsymbol{C}$, and the feedthrough matrix $\boldsymbol{D}$. Figure 3.2 shows a graphical representation.

In the NEF, we want to map a given dynamical system onto neural components (Fig. 3.3). The neuron dynamics are dominated by the synaptic filter (Eliasmith and C. H. Anderson 2003, Appendix F.1) which becomes the transfer function and gives

$$\boldsymbol{x}(t) = h(t) * \left[\boldsymbol{A}'\boldsymbol{x}(t) + \boldsymbol{B}'\boldsymbol{u}(t)\right]. \tag{3.11}$$

With help of the Laplace transform one can obtain $\boldsymbol{A}'$ and $\boldsymbol{B}'$ from $\boldsymbol{A}$ and $\boldsymbol{B}$ as

$$\boldsymbol{A}' = \tau_{\text{syn}}\boldsymbol{A} + \boldsymbol{I} \tag{3.12}$$

$$\boldsymbol{B}' = \tau_{\text{syn}}\boldsymbol{B}. \tag{3.13}$$

This implies that to implement a dynamical system with a neural ensemble, the input has to be multiplied by τ_{syn} to account for the synaptic filtering. In addition, one needs to add a recurrent connection implementing the function $f(\boldsymbol{x}) = \tau_{\text{syn}}\boldsymbol{A}\boldsymbol{x} + \boldsymbol{x}$. Note that combining this third principle with the principle of transfomation, allows the implementation of nonlinear dynamics in a spiking neural network.

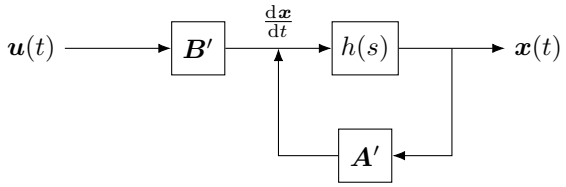


Figure 3.3.: Block diagram of the linear system of a neural population. $h(s)$ is the Laplace transformed synaptic filter $h(t)$.

3.4. Simulating NEF networks

To simulate NEF style networks, I use the Python library Nengo (Bekolay, Bergstra, et al. 2014; Sharma, Aubin, and Eliasmith 2016). It supports different backends to run neural models on different hardware platforms. For example, Nengo OCL targets GPUs with an OpenCL implementation. While this allows better simulation performance, special case implementations are necessary for certain features. In particular, this applies to the association matrix learning rule (see Chapter 9). Moreover, I utilized the Sharcnet and Compute Canada high-performance clusters which typically provide more CPU resources than GPU resources. Thus, I mostly used the Nengo reference (CPU) backend. To obtain sufficient simulation performance for the size of models constructed in this work, it was necessary to optimize memory organization of the internal data structures of the backend. While the exact details are out of the scope of this thesis, they are published in Gosmann and Eliasmith (2017).

4. Basic NEF networks

When constructing neural models with the NEF, there are certain networks that are often used in multiple places and constitute basic building blocks for larger scale networks. In the following, I briefly discuss how to create a differentiator, an integrator, a gated memory buffer based on that integrator, an ensemble applying a threshold to a signal, and a multiplier in neurons.

Often it is helpful to use a graphic representation to understand NEF networks. The graphical primitives used in the network diagrams in this thesis are shown in Fig. 4.1. Normal ensembles are marked with a circle. Sometimes the dimensions of a vector are split across multiple ensembles, called an *ensemble array*, and marked by a stack of circles. Multiple NEF components that constitute a network not shown in full detail, are identified with rectangles with rounded corners. Sometimes it is useful to combine decoded representations before transmitting them to the destination neurons in a *pass-through* node. Such nodes are marked by filled circles. Note that such pass-through nodes could be replaced with the actual neuron-to-neuron connections and are not biological implausible. *Rectification ensembles* are useful enough to warrant a

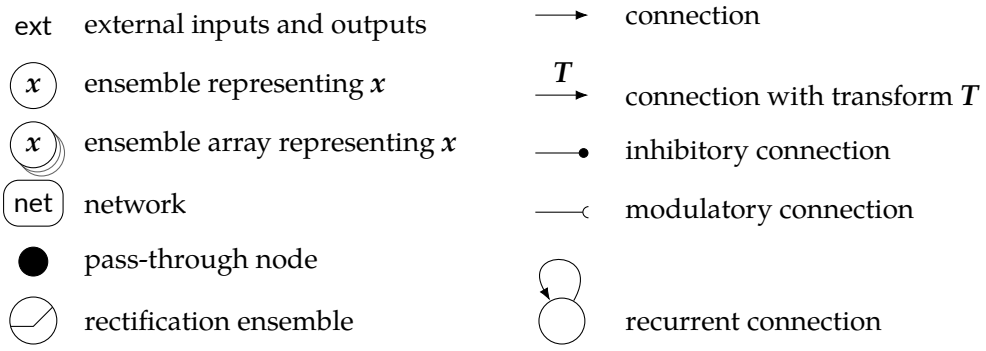


Figure 4.1.: Graphical elements in NEF network diagrams. See text for further explanation.

special symbol: a rectified linear plot in its circle. These ensembles are discussed in detail in Section 4.4. Normal NEF connections are represented by arrows and typically have a synaptic time constant of 5 ms, which is in the range of experimentally measured values for AMPA-type glutamate receptors (Jonas, Major, and Sakmann 1993; Spruston, Jonas, and Sakmann 1995). Linear transforms or functions computed across the connection are denoted along the arrow. Inhibitory connections bypass the encoders of the target ensemble and use a negative transform to suppress firing in the target ensemble. Finally, modulatory connections do not influence the representation in an ensemble directly, but the connection weights of another connection, and are indicated with a half open circle terminating the connection.

4.1. Differentiator

While a perfect differentiator is not physically realizable (it would require infinite gain), it is often sufficient to detect a sudden change in a signal. For this an approximation can be constructed out of two synaptic low pass filters

$$h_1(t) = \tau_1^{-1} \exp(-t/\tau_1) \quad \text{and} \quad h_2(t) = \tau_2^{-1} \exp(-t/\tau_2)$$

with time constants $\tau_1 < \tau_2$ as

$$\begin{aligned} \frac{\mathrm{d}x}{\mathrm{d}t} \approx \boldsymbol{y}(t) &= \boldsymbol{x} * h_1 - \boldsymbol{x} * h_2 \\ &= \boldsymbol{x} * (h_1 - h_2)\,. \end{aligned} \tag{4.1}$$

Fig. 4.2 shows the impulse and the magnitude response of a differentiator constructed this way. The attenuation of low frequencies is desired as this corresponds to the differentiation. The attenuation of high-frequencies above the pass-band is undesired, but given appropriate τ_1 and τ_2 unproblematic in a neural network because signals will in general be subject to synaptic filtering of high frequencies. The implementation of such a differentiator is straight-forward with the NEF (Fig. 4.3) by feeding the same input with two different synaptic time constants into an ensemble. In this work $\tau_1 = 5\,\mathrm{ms}$ and $\tau_2 = 50\,\mathrm{ms}$ were used, which are in the range of experimentally observed neurotransmitter decay constants (Sah, Hestrin, and Nicoll 1990; Moreno-Bote and Parga 2005).

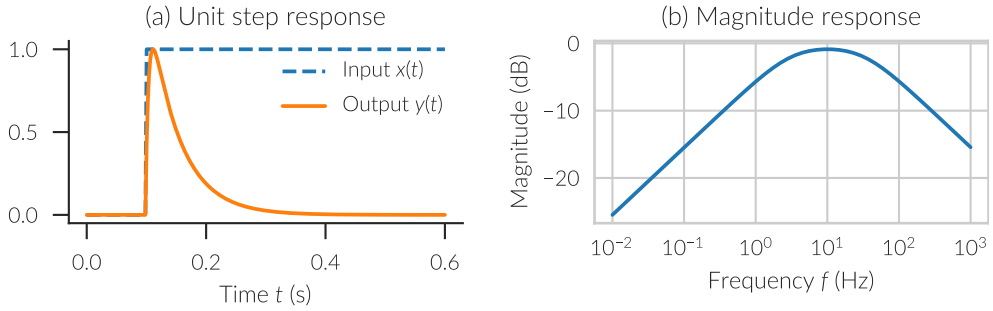


Figure 4.2.: Differentiator with $\tau_1 = 5\,\text{ms}$ and $\tau_2 = 50\,\text{ms}$. (a) Response of the differentiator to a unit step input. The output has been normalized to the maximum. (b) Magnitude response for sinusoidal inputs with different frequencies.

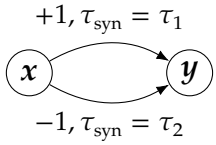


Figure 4.3.: Implementation of a differentiator with the NEF. The connection labels state the transform and synaptic time constants with $\tau_1 < \tau_2$.

4.2. Integrator

Integrators are important components in many NEF models because they enable the storage of values over some time span in neural activity. An integrator is described by the differential equation

$$\frac{\mathrm{d}\boldsymbol{x}(t)}{\mathrm{d}t} = \boldsymbol{u}(t) \tag{4.2}$$

where $\boldsymbol{u}(t)$ is the external input to the integrator. Applying principle 3 of the NEF tells us that the input has to be scaled by the synaptic time constant τ_{syn}. Furthermore, a recurrent connection feeding the output of the integrator back to itself is needed (Fig. 4.4). To get a stable representations over a sufficient time window, it is best to use a long time constant like $\tau_{\text{syn}} = 0.1\,\text{s}$ which is the range measured for the NMDA neurotransmitter (Sah, Hestrin, and

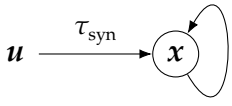


Figure 4.4.: Implementation of an integrator with the NEF.

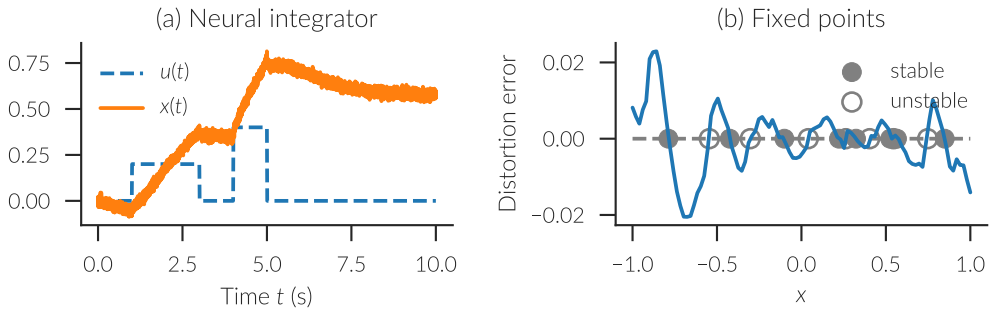


Figure 4.5.: Neural integrator with 100 neurons and a recurrent synaptic time constant of $\tau_{\text{syn}} = 100\,\text{ms}$. (a) Example input $u(t)$ and output $x(t)$ of the neural integrator. (b) Distortion error (blue curve) and fixed points. Stable fixed points are marked by filled circles, unstable fixed points by unfilled circles.

Nicoll 1990; Moreno-Bote and Parga 2005). Due to neural noise and distortion error, the represented value can drift over time (Fig. 4.5). The distortion in the decoded representation leads to a finite number of fixed points and the represented value will shift to one of the stable fixed points. The neural noise prevents the representation from staying in an unstable fixed point and can make the representation move out of the basin of attraction of a stable fixed point. Adding more neurons to an integrator makes it more stable as the spiking noise is decreased. Similarly, a longer recurrent synaptic time constant also makes the integrator more stable. Increasing the number of represented dimensions makes the integrator less stable because there are fewer fixed points.

4.3. Gated memory buffer

While the integrator enables us to store a value over time, it does not allow for particularly quick updating. A quicker update can be achieved by adding a difference ensemble (Fig. 4.6). By scaling the difference with a gain factor g the updating speed can be regulated. However, too large of a value leads to oscillations in the integrator. Note that feeding a null vector to the difference ensemble clears out the memory instead of keeping the current value. Thus, the input the integrator needs to be gated. This can be done by inhibiting the

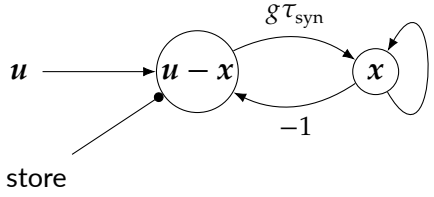


Figure 4.6.: A gated memory buffer network. The store input controls whether the input $u(t)$ is allowed to overwrite the stored value $x(t)$.

neurons of the difference ensemble to keep the current value in the integrator.

4.4. Thresholding ensembles

Often one needs to apply a threshold to value, i.e. implement the function

$$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}, \tag{4.3}$$

or compute the Heaviside step function

$$\Theta(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}. \tag{4.4}$$

Both of these functions are non-differentiable at zero. The Heaviside function is even discontinuous at that spot. These properties make it problematic to implement this function with a standard NEF ensemble. Nevertheless, good approximations of these functions can be achieved by aligning the neuron's tuning curves according to the shape of these functions.

Instead of choosing encoders randomly as -1 and 1, all encoders are set to 1 and all intercepts of the neuron tuning curves are chosen from $x \in [0; 1]$. Choosing the intercept distribution in this interval appropriately can further increase the accuracy. An exponential distribution that clusters intercepts close to zero performs best (Fig. 4.7). Note that this is even better than setting all intercepts to zero as this gives more variation in the tuning curves. The uniform distribution often does not produce intercepts close enough to the threshold value which leads to an increased effective threshold compared to the desired threshold. To adjust the threshold to a non-zero value a constant bias term can be fed into the thresholding ensemble.

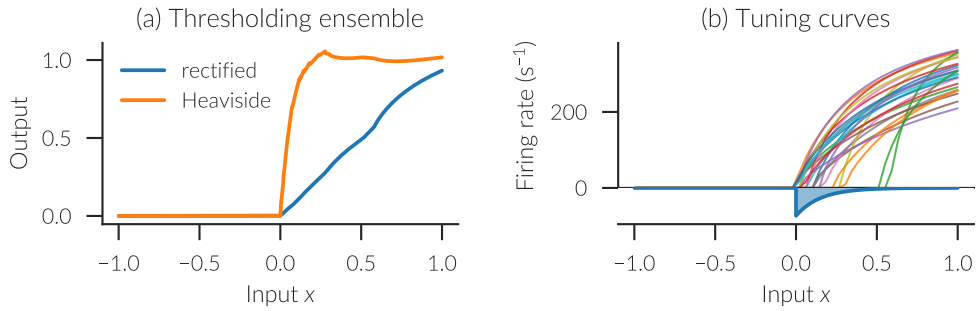


Figure 4.7.: Thresholding ensemble with 25 neurons. (a) Decoded rectified and Heaviside output. (b) Tuning curves in the upper part; probability density function of the exponential intercept distribution with inverted y-axis (i.e. probability density increases downward).

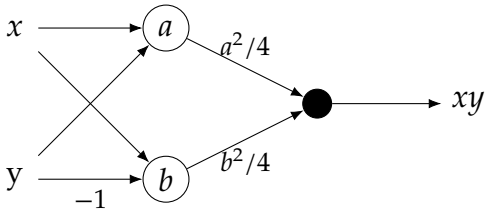


Figure 4.8.: Implementation of an accurate product with the NEF.

4.5. Product

A product of two scalar numbers x and y could be computed by feeding them into separate dimensions of a two-dimensional ensemble and decoding out the product. A 37 % more accurate implementation (with the same number of neurons) is, however, possible (Gosmann 2015) by rewriting the product with squares as

$$xy = \frac{1}{4}\left(x^2 + 2xy + y^2\right) - \frac{1}{4}\left(x^2 - 2xy + y^2\right) = \frac{1}{4}\left(x + y\right)^2 - \frac{1}{4}\left(x - y\right)^2. \tag{4.5}$$

The neural implementation of this equation is straight-forward (Fig. 4.8).

Multiple scalar product networks can be combined to compute element-wise vector products. By summing across those element-wise products a dot product can be computed. Product networks are also used in the computation of circular convolution as the binding operation in the Semantic Pointer Architecture (Chapter 5).

5. The Semantic Pointer Architecture

While the Neural Engineering Framework allows us to encode vectors into spiking neural networks and transform them, it does not tell us how to use those vectors to represent structured, conceptual, or symbol-like information. Different such methods could be devised, though in the context of the NEF the most widely used method is the Semantic Pointer Architecture (SPA; Eliasmith 2013). The SPA has been used to build a multitude of cognitive models, including the n-back task (Gosmann and Eliasmith 2015), the Tower of Hanoi task (Stewart and Eliasmith 2011), human-scale knowledge representation (Crawford, Gingerich, and Eliasmith 2016). The largest and most complex example of a SPA model is Spaun, the Semantic Pointer Architecture Unified Network, combining eight different tasks in a single functional spiking-neuron model (Eliasmith, Stewart, et al. 2012).

The conceptual representations in the SPA are a specific instance of a Vector Symbolic Architecture (VSA; Gayler 2004). In VSAs concepts are represented with vectors, and linear and nonlinear operators are used to combine basic concepts in more complex structured representations. Three types of operators are considered essential in a VSA. First, a measure of similarity

$$s : \mathbb{R}^d \times \mathbb{R}^d \longrightarrow \mathbb{R} \tag{5.1}$$

for which I use the normalized dot product (also known as cosine similarity)

$$s(\boldsymbol{x}, \boldsymbol{y}) := \frac{\langle \boldsymbol{x}, \boldsymbol{y} \rangle}{\|\boldsymbol{x}\| \cdot \|\boldsymbol{y}\|} \tag{5.2}$$

for the remainder of this thesis. Second, a superposition operator

$$\mathcal{S} : \mathbb{R}^d \times \mathbb{R}^d \longrightarrow \mathbb{R}^d \tag{5.3}$$

is required that produces a vector similar to both inputs, i.e., $s(\mathcal{S}(\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{x}) \approx s(\mathcal{S}(\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{y}) \gtrapprox \sqrt{1/2}$ for $s(\boldsymbol{x}, \boldsymbol{y}) \approx 0$. This is usually, and will be for the

remainder of this thesis, simple elementwise addition, i.e., $\mathcal{S}(\boldsymbol{x}, \boldsymbol{y}) := \boldsymbol{x} + \boldsymbol{y}$. Finally, a binding operator

$$\mathcal{B} : \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} \longrightarrow \mathbb{R}^{d} \tag{5.4}$$

is needed with an approximate inverse or unbinding operation

$$\mathcal{B}^{+} : \mathbb{R}^{d} \times \mathbb{R}^{d_2} \longrightarrow \mathbb{R}^{d_1}. \tag{5.5}$$

To be able to build up and retrieve information from representations, the binding and unbinding operations are required to be distributive, i.e.,

$$\mathcal{B}(\boldsymbol{x} + \boldsymbol{y}, \boldsymbol{z}) = \mathcal{B}(\boldsymbol{x}, \boldsymbol{z}) + \mathcal{B}(\boldsymbol{y}, \boldsymbol{z}) \tag{5.6}$$

$$\mathcal{B}^{+}(\boldsymbol{x} + \boldsymbol{y}, \boldsymbol{z}) = \mathcal{B}^{+}(\boldsymbol{x}, \boldsymbol{z}) + \mathcal{B}^{+}(\boldsymbol{y}, \boldsymbol{z}). \tag{5.7}$$

For some proposed binding operations, like Tensor products (Smolensky 1990), the unbinding operation is the exact inverse $\mathcal{B}^{+} = \mathcal{B}^{-1}$ with

$$\mathcal{B}^{-1}(\mathcal{B}(\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{y}) = \boldsymbol{x}.$$

However, Tensor products increase the vector dimensionality with each successive binding which leads to biological implausible scaling problems (Eliasmith 2013, Appendix D.5). For that reason, I only consider binding methods that keep the dimensionality $d = d_1 = d_2$ constant.

In pure math, it is still possible to define binding operators with an exact inverse, but we need to keep the implementation in neurons in mind. This introduces additional constraints. First, the actual usable representational space $\mathcal{X} \subsetneq \mathbb{R}^{d}$ is limited, constraining unlimited growth of bound vectors. Second, neural noise limits the representational accuracy, preventing highly non-linear operators. It follows that for a binding operator in a neural system it is desired to maximize the set $\mathcal{V} \subseteq \mathcal{X}$ for which $s(\mathcal{B}^{+}(\mathcal{B}(\boldsymbol{x}, \boldsymbol{y}) + \boldsymbol{\zeta}, \boldsymbol{y}), \boldsymbol{x}) \approx 1$ for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{V}$ and samples of noise $\boldsymbol{\zeta}$ in the neural system. Note that this condition would be satisfied by using the identity $\mathcal{B}(\boldsymbol{x}, \boldsymbol{y}) = \mathcal{B}^{+}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}$. Thus, simultaneously it needs to hold that $s(\mathcal{B}^{+}(\mathcal{B}(\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{z}), \boldsymbol{x}) \approx 0$ for all $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathcal{V}$ with $s(\boldsymbol{y}, \boldsymbol{z}) \approx 0$. At this point, it is useful to introduce three more definitions.

Definition 1 (identity vector)**.** *A vector $\boldsymbol{i}_{\mathcal{B}}$ with the property $\mathcal{B}(\boldsymbol{x}, \boldsymbol{i}_{\mathcal{B}}) = \boldsymbol{x}$ is called* identity vector *under $\mathcal{B}$.*

Definition 2 (absorbing element)**.** *A vector $\boldsymbol{n}_{\mathcal{B}}$ with the property $\mathcal{B}(\boldsymbol{x}, \boldsymbol{n}_{\mathcal{B}}) = c \cdot \boldsymbol{n}_{\mathcal{B}}$ where $c \in \mathbb{R}$ is called an* absorbing element *under $\mathcal{B}$.*

Such an absorbing element effectively destroys the information in the vector $\boldsymbol{x}$. For that reason, absorbing elements should be avoided when constructing representations with binding. Note that this definition slightly differs from the usual definition of absorbing elements by allowing for a scaling factor.

Definition 3 (unitary vector)**.** *A vector $\boldsymbol{u}$ with the property $\langle \mathcal{B}(\boldsymbol{x}, \boldsymbol{u}), \mathcal{B}(\boldsymbol{y}, \boldsymbol{u}) \rangle = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$ is called unitary.*

In other words, a unitary vector preserves the dot product under binding. This is in analogy to unitary transformation matrices that also preserve the dot product. It also implies that binding with a unitary vector preserves the length of the bound vector.

5.1. Binding operations

I introduce two specific binding operations now. First, circular convolution is discussed, which has been the binding operation of choice in the SPA so far. Second, I introduce a new binding method, termed vector-derived transformation binding, with some trade-offs. The two binding approaches and their suitability for different problems is discussed.

5.1.1. Circular convolution

The binding operator classically used in the SPA is circular convolution, and was suggested by Plate (1995) and Plate (2003) for his Holographic Reduced Representations (HRRs).

Definition 4 (circular convolution binding)**.** *The circular convolution binding operator is given by*

$$\mathcal{B}_{\circledast}(\boldsymbol{x}, \boldsymbol{y}) := \boldsymbol{x} \circledast \boldsymbol{y} \text{ with } \left(\boldsymbol{x} \circledast \boldsymbol{y}\right)_i = \sum_{j=0}^{d-1} x_j y_{(i-j) \bmod d} \tag{5.8}$$

and has the approximate inverse (Plate 2003)

$$\mathcal{B}^{+}_{\circledast}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x} \circledast \boldsymbol{y}^{+} \text{ with } \boldsymbol{y}^{+} := \left(y_0, y_{d-1}, y_{d-2}, \cdots, y_1\right)^{\top}. \tag{5.9}$$

The basic properties of

$$\text{distributivity:} \quad (\boldsymbol{x}_1 + \boldsymbol{x}_2) \circledast \boldsymbol{y} = \boldsymbol{x}_1 \circledast \boldsymbol{y} + \boldsymbol{x}_2 \circledast \boldsymbol{y}, \tag{5.10}$$

$$\text{associativity:} \quad (\boldsymbol{x} \circledast \boldsymbol{y}) \circledast \boldsymbol{z} = \boldsymbol{x} \circledast (\boldsymbol{y} \circledast \boldsymbol{z}), \tag{5.11}$$

$$\text{commutativity:} \quad \boldsymbol{x} \circledast \boldsymbol{y} = \boldsymbol{y} \circledast \boldsymbol{x} \tag{5.12}$$

hold for circular convolution as a binding operator. A useful property of circular convolution for the implementation in a neural network with the NEF is, that it becomes element-wise multiplication in the Fourier space defined by

$$\boldsymbol{x} \circledast \boldsymbol{y} = \boldsymbol{T}_{\mathcal{F}}^{-1} \left[(\boldsymbol{T}_{\mathcal{F}} \boldsymbol{x}) \circ (\boldsymbol{T}_{\mathcal{F}} \boldsymbol{y}) \right] \tag{5.13}$$

where $\boldsymbol{T}_{\mathcal{F}}$ is the discrete Fourier transform (DFT) matrix. The linear transform with the DFT matrix can be put easily into the neural connection weights and the element-wise product can be done with a well-optimized product network (Gosmann 2015).

The expression in Fourier space also allows the derivation of the special elements of circular convolution. The identity vector must not change the complex Fourier coefficient in the element-wise multiplication. Thus, its Fourier coefficients must all be $1 + 0\mathrm{i}$ and the identity vector is given by

$$\boldsymbol{i}_{\circledast} = (1, 0, 0, \dots, 0)^{\top}. \tag{5.14}$$

Furthermore, all vectors with Fourier coefficients $c_n \in \mathbb{C}$ that have an absolute value of $|c_n| = 1$ are unitary, as one can easily verify. A trivial example of a unitary vector is the identity vector $\boldsymbol{i}_{\circledast}$. Finally, all vectors $(z, \dots, z)^{\top}$ with $z \in \mathbb{R}$ are absorbing elements.

5.1.2. Vector-derived transformation binding

Circular convolution can be interpreted as moving one of the operands around in the d-dimensional space in a way defined by the other operand. This leads to the question, whether there are other ways to project one vector to a new location based on the other vector. One such way is what I call *vector-derived transformation binding (VTB)*, which to my knowledge has not been described before.

Definition 5 (vector-derived transformation binding, VTB). *Given a* $d' = d^{1/2} \in \mathbb{N}_{>0}$*, the vector-derived transformation binding operator* $\mathcal{B}_{\mathrm{V}} : \mathbb{R}^d \times \mathbb{R}^d \longrightarrow \mathbb{R}^d$ *is defined as*

$$\mathcal{B}_{\mathrm{V}}(\boldsymbol{x}, \boldsymbol{y}) := \bar{\boldsymbol{V}}_{\boldsymbol{y}} \boldsymbol{x} = \begin{bmatrix} \boldsymbol{V}_{\boldsymbol{y}} & 0 & 0 \\ 0 & \boldsymbol{V}_{\boldsymbol{y}} & 0 \\ 0 & 0 & \ddots \end{bmatrix} \boldsymbol{x} \tag{5.15}$$

with

$$\boldsymbol{V}_{\boldsymbol{y}} = d^{\frac{1}{4}} \begin{bmatrix} y_1 & y_2 & \cdots & y_{d'} \\ y_{d'+1} & y_{d'+2} & \cdots & y_{2d'} \\ \vdots & \vdots & \ddots & \vdots \\ y_{d-d'+1} & y_{d-d'+2} & \cdots & y_d \end{bmatrix}. \tag{5.16}$$

The approximate inverse is given by

$$\mathcal{B}^{+}_{\mathrm{V}}(\boldsymbol{x}, \boldsymbol{y}) = \bar{\boldsymbol{V}}_{\boldsymbol{y}}^{\top} \boldsymbol{x} = \begin{bmatrix} V_{\boldsymbol{y}}^{\top} & 0 & 0 \\ 0 & V_{\boldsymbol{y}}^{\top} & 0 \\ 0 & 0 & \ddots \end{bmatrix} \boldsymbol{x}. \tag{5.17}$$

This binding method is based on the fact that in the SPA vectors are usually randomly generated and uniformly distributed with identically distributed components. In that case each subvector (e.g., each row in $\boldsymbol{V}_{\boldsymbol{y}}$) is also uniformly distributed with identically distributed components. Furthermore, for high-dimensional vector spaces almost all (uniformly sampled) vectors are orthogonal and semantic pointers are usually picked to have unit-length. Thus, the matrix $\bar{\boldsymbol{V}}_{\boldsymbol{y}}$ is almost orthogonal with the implication $\bar{\boldsymbol{V}}_{\boldsymbol{y}}^{\top} \bar{\boldsymbol{V}}_{\boldsymbol{y}} \approx \boldsymbol{I}$. Vectors $\boldsymbol{y}$, that give a perfectly orthogonal matrix $\boldsymbol{V}_{\boldsymbol{y}}$, are unitary. One special unitary vector is the identity vector.

Corollary 1 (VTB identity vector). *The identity vector for VTB is given by*

$$[\boldsymbol{i}_{\mathrm{V}}]_i = \begin{cases} d^{-\frac{1}{4}} & i \in \left\{(k-1)d' + k : k \leq d', k \in \mathbb{N}_{>0}\right\} \\ 0 & \textit{otherwise} \end{cases}. \tag{5.18}$$

Proof. By writing $\boldsymbol{i}_{\mathrm{V}}$ as $\boldsymbol{V}_{\boldsymbol{i}_{\mathrm{V}}}$ one can easily verify that $\boldsymbol{V}_{\boldsymbol{i}_{\mathrm{V}}} = \boldsymbol{I} \Rightarrow \bar{\boldsymbol{V}}_{\boldsymbol{i}_{\mathrm{V}}} = \boldsymbol{I}$. □

Example for $d = 9$: $\boldsymbol{i}_{\mathrm{V}}^{(9)} = (1, 0, 0, 0, 1, 0, 0, 0, 1)^{\top}$.

Corollary 2 (VTB distributivity). *VTB is distributive:*

$$\begin{aligned} &\mathcal{B}_{\mathrm{V}}(\boldsymbol{x}_1 + \boldsymbol{x}_2, \boldsymbol{y}) = \mathcal{B}_{\mathrm{V}}(\boldsymbol{x}_1, \boldsymbol{y}) + \mathcal{B}_{\mathrm{V}}(\boldsymbol{x}_2, \boldsymbol{y}) \textit{ and} \\ &\mathcal{B}_{\mathrm{V}}(\boldsymbol{x}, \boldsymbol{y}_1 + \boldsymbol{y}_2) = \mathcal{B}_{\mathrm{V}}(\boldsymbol{x}, \boldsymbol{y}_1) + \mathcal{B}_{\mathrm{V}}(\boldsymbol{x}, \boldsymbol{y}_2). \end{aligned} \tag{5.19}$$

Proof. By applying the definitions for both directions of the distributivity:

- $\mathcal{B}_{\mathrm{V}}(\boldsymbol{x}_1 + \boldsymbol{x}_2, \boldsymbol{y}) = \bar{\boldsymbol{V}}_{\boldsymbol{y}} (\boldsymbol{x}_1 + \boldsymbol{x}_2) = \bar{\boldsymbol{V}}_{\boldsymbol{y}} \boldsymbol{x}_1 + \bar{\boldsymbol{V}}_{\boldsymbol{y}} \boldsymbol{x}_2 = \mathcal{B}_{\mathrm{V}}(\boldsymbol{x}_1, \boldsymbol{y}) + \mathcal{B}_{\mathrm{V}}(\boldsymbol{x}_2, \boldsymbol{y})$
- $\mathcal{B}_{\mathrm{V}}(\boldsymbol{x}, \boldsymbol{y}_1 + \boldsymbol{y}_1) = \bar{\boldsymbol{V}}_{\boldsymbol{y}_1 + \boldsymbol{y}_2} \boldsymbol{x} = \left(\bar{\boldsymbol{V}}_{\boldsymbol{y}_1} + \bar{\boldsymbol{V}}_{\boldsymbol{y}_2} \right) \boldsymbol{x} = \bar{\boldsymbol{V}}_{\boldsymbol{y}_1} \boldsymbol{x} + \bar{\boldsymbol{V}}_{\boldsymbol{y}_2} \boldsymbol{x} = \mathcal{B}_{\mathrm{V}}(\boldsymbol{x}, \boldsymbol{y}_1) + \mathcal{B}_{\mathrm{V}}(\boldsymbol{x}, \boldsymbol{y}_2)$

□

In contrast to circular convolution, VTB is neither commutative

$$\mathcal{B}_{\mathrm{V}}(\boldsymbol{x}, \boldsymbol{y}) = \bar{\boldsymbol{V}}_{\boldsymbol{y}} \boldsymbol{x} \neq \bar{\boldsymbol{V}}_{\boldsymbol{x}} \boldsymbol{y} = \mathcal{B}_{\mathrm{V}}(\boldsymbol{y}, \boldsymbol{x}), \tag{5.20}$$

nor associative

$$\mathcal{B}_{\mathrm{V}}(\boldsymbol{x}, \mathcal{B}_{\mathrm{V}}(\boldsymbol{y}, \boldsymbol{z})) = \bar{\boldsymbol{V}}_{\bar{\boldsymbol{V}}_{\boldsymbol{z}} \boldsymbol{y}} \boldsymbol{x} \neq \bar{\boldsymbol{V}}_{\boldsymbol{z}} \bar{\boldsymbol{V}}_{\boldsymbol{y}} \boldsymbol{x} = \mathcal{B}_{\mathrm{V}}(\mathcal{B}_{\mathrm{V}}(\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{z}). \tag{5.21}$$

This implies that unlike circular convolution, multiple bindings cannot be undone in a single step, but a separate unbinding step is required for each binding. Despite the non-commutativity, it is possible to flip the operands in the bound state $\mathcal{B}_{\mathrm{V}}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{V}_{\leftrightarrow} \mathcal{B}_{\mathrm{V}}(\boldsymbol{y}, \boldsymbol{x})$ with the matrix

$$[\boldsymbol{V}_{\leftrightarrow}]_{ij} = \begin{cases} 1 & j = 1 + \left\lfloor \frac{i-1}{d'} \right\rfloor + d' \left[(i-1) \bmod d' \right] \\ 0 & \text{otherwise} \end{cases} . \tag{5.22}$$

Example for $d = 4$:

$$\boldsymbol{V}_{\leftrightarrow}^{(4)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5.1.3. Comparison of circular convolution and vector-derived transformation binding

Both circular convolution and VTB are compressed binding operations. Because of the lossy compression, we lose some information in each binding which makes it increasingly harder to recover the original unbound vectors. To combat this effect (and the effect of neuron noise) clean-up memories like the one by

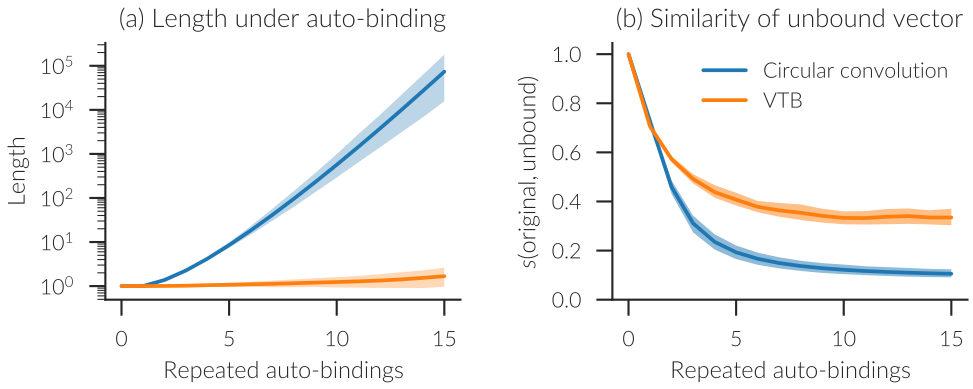


Figure 5.1.: Repeatedly binding a 256-dimensional vector with itself. (a) Length of the resulting vector. (b) Similarity of the original vector to the vector obtained when undoing all bindings. The shaded areas represent 95 % confidence intervals.

Stewart, Tang, and Eliasmith (2011) are required. Likewise the independent accumulator model discussed in Section 10.2 can be used as a clean-up memory.

In addition to the information loss, the binding operations change the length of the vector (if neither operand is unitary). This can be a problem in a neural representation as neurons saturate and might not represent the vector accurately anymore. In particular, neural ensembles in the NEF are optimized for a certain representational space, usually a hyperball with a given radius r. It is convenient to set $r = 1$ and try to keep the Semantic Pointer vectors at unit-length.

Figure 5.1 shows how the mean length of random vectors repeatedly bound with themselves changes. For the circular convolution the length increases much faster than for VTB. Such a rapid increase in length can be problematic in a neural network with a limited representational radius as in the NEF. Moreover, VTB preserves more information of the bound vectors as shown by the similarity to the original vectors when undoing all bindings. After repeated binding with itself and then the same number of unbindings, the resulting VTB bound vector is more similar to the original vector than the circular convolution bound vector.

While binding vectors with themselves can sometimes be useful (e.g., for generating Semantic Pointers with a successive relationship like position indices), it is much more common to bind randomly sampled vectors. Figure 5.2

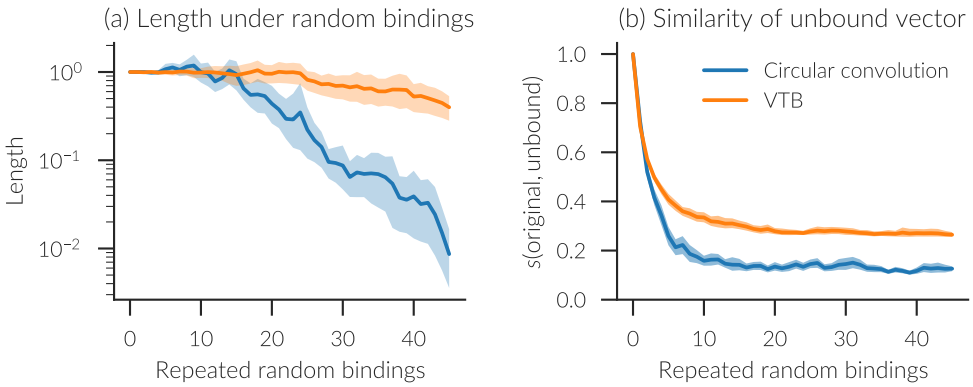


Figure 5.2.: Repeatedly binding a 256-dimensional vector with random vectors. (a) Length of the resulting vector. (b) Similarity of the original vector to the vector obtained when undoing all bindings. The shaded areas represent 95 % confidence intervals.

shows the same experiment where a random vector was used in each binding. In this case, the vector length decreases to zero, even if it might increase in some of the early bindings. Again, this decrease is much quicker for circular convolution binding than for VTB and the latter method also preserves more of the similarity across bindings.

It is conceivable that the problems with scaling of the vector length can be fixed by normalizing after each binding. This, however, does not affect the loss of information in each binding (Fig. 5.3). Also, implementing normalization in a neural network is notoriously difficult because of the division involved with an unbounded output as the divisor approaches zero. Good approximations of normalization are only possible for a defined and finite input range. It is worth noting that the neurons in the NEF perform a sort of "soft normalization" for large values, as the neuron's firing rates saturate. But this only affects vectors exceeding a certain length and can lead to other distortions in the representation.

Another approach to preventing the growth or decay of the vector length, and even prevent the information loss, is the usage of unitary vectors. These keep the vector length constant and perform lossless binding due to their perfect inverse. Note that binding two unitary vectors gives another unitary vector. Thus, repeated binding is not a problem. However, the scaling properties of VSAs are based on the fact that the number of almost orthogonal vectors

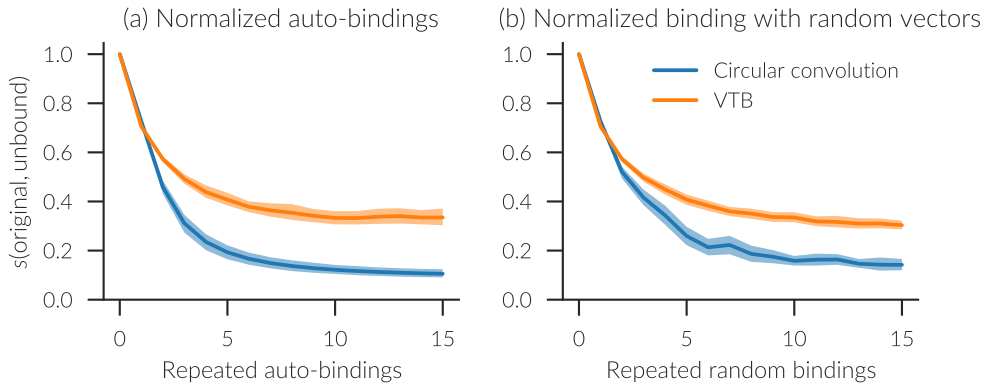


Figure 5.3.: Repeatedly binding a 256-dimensional vector with (a) itself or (b) random vectors and normalizing in each step. The similarity to the original vector after undoing all bindings is shown. The shaded areas represent 95 % confidence intervals.

that fits into a vector space grows exponentially with the dimensionality of that space (Wyner 1967; Cai, Fan, and Jiang 2013). Because not all vectors are unitary, this scaling property might be lost when restricted to unitary vectors. It might be best to use unitary vectors only for those Semantic Pointers that are repeatedly used in bindings. But it is also worth keeping in mind that achieving the theoretical limit of almost orthogonal vectors in a space is a hard problem, closely related to sphere packing and unsolved for arbitrary dimensionality (Cohn et al. 2017). Thus, the practical scaling of the number of useable vectors might be less then the exponential scaling for both unitary and non-unitary vectors.

So far VTB looks like the better choice for a binding operation. But it does not come without downsides. In contrast to circular convolution, it is not associative or commutative. While the desirability of commutativity depends on the employed representation scheme, the non-associativity implies that each binding has to be undone in an individual step, while circular convolution allows us to undo a chain of bindings in a single step, if the vector representing that chain is available. Thus, circular convolution might allow the recovery of information more quickly. Ultimately, this is a question of what binding operations the brain applies for different forms of processing (if such binding is at all related to what the brain does). Potentially, the binding operations lead to different timing predictions, as unbinding with the VTB may take more time.

Deriving such predictions and testing them experimentally is, however, out of the scope of this thesis.

Finally, we have to consider the neural implementation of these binding operations. Both essentially require a set of multiplication networks. For the circular convolution, the DFT (and inverse DFT) can be implemented in feed-forward connection weights that do not require any additional neurons. For each of the input vectors, d complex Fourier coefficients are produced, but as the inputs are real-valued, half of these are the complex conjugate of the other half. Thus, only $d/2$ coefficients have to be considered. Each coefficient is a complex number multiplied with one coefficient of the other vector. That results in four real-valued multiplies per coefficient. In total, $2d$ multiplications are required for a circular convolution. For VTB, there are $d^{1/2}$ multiplications of $d^{1/2} \times d^{1/2}$ matrices with a vector, resulting in a total of $d^{3/2}$ multiplications. However, each column vector in $\bar{\boldsymbol{V}}_{\boldsymbol{y}}$ gets scaled by the same component of $\boldsymbol{x}$. That would allow the storage of each of these column vectors in a single NEF ensemble with the respective x_i in an additional dimension, and then decode the scaled column vector. This requires only d ensembles to decode from, but each ensemble needs to represent $d^{1/2} + 1$ dimensions, and to keep the noise error constant (as discussed in the next chapter[1]), the number of neurons in each ensemble then needs to be scaled by $(d^{1/2} + 1)^{3/2} \approx d^{3/4}$. Across all ensembles this amounts to a scaling by $d^{7/4}$. Thus, the VTB requires more neural resources. It should be noted, that for either binding method, the binding with a fixed vector can be implemented purely in the connection weights as it reduces to a simple matrix multiplication in either case.

Despite VTB having many advantages over circular convolution, I decided to use circular convolution in the memory model. The main reason is that support for circular convolution is already implemented in Nengo and the model does not use a lot of binding operations. Nevertheless, it would be interesting to switch the model over to VTB and investigate effects on the performance in the future.

5.2. Structured representations

Once we defined a binding operation, it can be used to build up structured representations with Semantic Pointers. For example, consider a scene with a

[1] Noise error scales by $O\left(d^{3/4}/\sqrt{n}\right)$, thus the neuron number has to be scaled by $O\left(d^{6/4}/E^2\right)$.

red square and blue circle that we want to encode as Semantic Pointer. Assume that we have Semantic Pointers RED, SQUARE, BLUE, and CIRCLE. One possible encoding would be

$$\boldsymbol{s} = \mathcal{B}(\text{RED}, \text{SQUARE}) + \mathcal{B}(\text{BLUE}, \text{CIRCLE}). \tag{5.23}$$

We could then recover the color of the square as

$$\mathcal{B}^+(\boldsymbol{s}, \text{SQUARE}) = \mathcal{B}^+(\mathcal{B}(\text{RED}, \text{SQUARE}), \text{SQUARE}) + \mathcal{B}^+(\mathcal{B}(\text{BLUE}, \text{CIRCLE}), \text{SQUARE}) \tag{5.24}$$

$$\approx \text{RED} + \text{noise}. \tag{5.25}$$

Note, however, that the binding operation needs to be commutative to recover the shape from the color with this encoding scheme. Other encoding schemes can be devised to alleviate this concern. For example, the properties can be bound to a role and each object to an object identifier:

$$\boldsymbol{o}_1 = \mathcal{B}(\text{RED}, \text{COLOR}) + \mathcal{B}(\text{SQUARE}, \text{SHAPE}) \tag{5.26}$$

$$\boldsymbol{o}_2 = \mathcal{B}(\text{BLUE}, \text{COLOR}) + \mathcal{B}(\text{CIRCLE}, \text{SHAPE}) \tag{5.27}$$

$$\boldsymbol{s} = \mathcal{B}(\boldsymbol{o}_1, \text{OBJ}_1) + \mathcal{B}(\boldsymbol{o}_2, \text{OBJ}_2). \tag{5.28}$$

To find the color of a specific shape, each object would have to be retrieved, then the shape of the object to compare it to the target shape, and finally the color has to recovered if the shape matches. Thus, different encoding schemes can lead to different timing predictions. The latter approach requires scanning and through multiple objects and thus it should take longer to recover information with more items, while the former approach can recover any property in a single step. Despite those differences, both encoding approaches are similar, in so far as pairs of semantic pointers are bound together.

Definition 6 (encoding with binding)**.** *Given k pairs* $(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathbb{R}^d \times \mathbb{R}^d$*, the encoding of these pairs into a single Semantic Pointer* $\boldsymbol{m}$ *with binding is given by*

$$\boldsymbol{m} = \sum_{i=1}^{k} \mathcal{B}(\boldsymbol{x}_i, \boldsymbol{y}_i). \tag{5.29}$$

An $\boldsymbol{x}_i$ from such a trace can be recalled as $\boldsymbol{x}_i \approx \hat{\boldsymbol{x}}_i = \mathcal{B}^+(\boldsymbol{m}, \boldsymbol{x}_i)$. While most concrete encoding schemes make use of encoding with binding, at least one other method, encoding with tagging, has been proposed (Recchia et al. 2015).

Definition 7 (encoding with tagging). *Given k pairs* $(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathbb{R}^d \times \mathbb{R}^d$ *and a matrix* $\boldsymbol{M} \in \mathbb{R}^{d \times d}$ *with an approximate inverse* $\boldsymbol{M}^+$ *satisfying* $\boldsymbol{M}^+\boldsymbol{M} \approx \boldsymbol{I}$*, the encoding into a single Semantic Pointer with tagging is given by*

$$\boldsymbol{m} = \sum_{i=1}^{k} \boldsymbol{M}^{2i-1} \left(\boldsymbol{y}_i + \boldsymbol{M}\boldsymbol{x}_i \right) = \sum_{i=1}^{k} \boldsymbol{M}^{2i-1}\boldsymbol{y}_i + \boldsymbol{M}^{2i}\boldsymbol{x}_i. \tag{5.30}$$

The retrieval of an $\boldsymbol{x}_i \approx \hat{\boldsymbol{x}}_i$ is accomplished with

$$\hat{\boldsymbol{x}}_i := \left(\boldsymbol{M}^{2c} \right)^+ \boldsymbol{m} \tag{5.31}$$

$$c = \underset{j \in [1,\, k]}{\arg\max}\; s\left(\boldsymbol{y}_i, \left(\boldsymbol{M}^{2j-1} \right)^+ \boldsymbol{m} \right). \tag{5.32}$$

There are a number of sensible choices for the matrix $\boldsymbol{M}$.

- First of all, for both, circular convolution and VTB, binding with a *fixed* vector $\boldsymbol{v}$ can be expressed as a matrix multiplication. A matrix $\boldsymbol{M}$ derived in this way is unitary, if and only if the vector $\boldsymbol{v}$ is unitary for the given binding operation.

- A common choice for encoding with tagging are permutation matrices. These are unitary and an exact inverse is given by the transpose.

- A special permutation matrix is the matrix that shifts all vector components by one. For a given permutation matrix, the vector space dimensions can be reordered such that the permutation matrix becomes the shift by one. Interestingly, the shift by one is also equivalent to a circular convolution with the vector $(0, 1, 0, 0, \dots)^\top$.

- Other good candidates for M, that have not been considered to my knowledge, are (random) orthogonal matrices. These are unitary operators on $\mathbb{R}^d$ and thus have an exact inverse given by the transpose.

The relations of these different matrix choices is shown in the Venn diagram in Fig. 5.4.

Either encoding method allows the recovery of encoded vectors, but the encoding vector itself is (in general) not similar to those encoded vectors. In this way, the vector is analogous to a pointer in computer science that doesn't

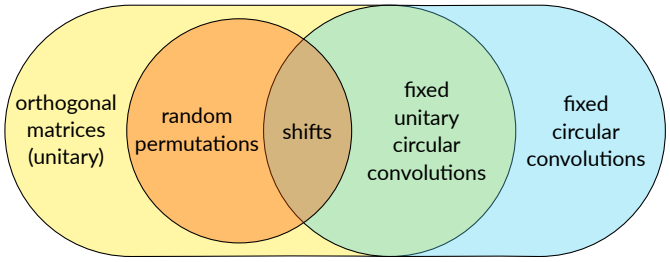


Figure 5.4.: Venn diagram of the relations between different classes of matrices $\boldsymbol{M}$ that can be used for encoding with tagging.

directly contain the information, but can be dereferenced to access the information it is pointing to. Unlike pointers, however, these vectors can also capture semantic information by their distance in vector space. Due to the combination of these two facts, these vectors are called Semantic Pointers.

5.2.1. Comparison of encoding methods

Based on the first experiment from Recchia et al. (2015), the encoding methods can be compared by measuring the pairwise binding capacity. To do so, a set of 1000 vectors with normally distributed components $x_i \sim \mathcal{N}(0, \sqrt{1/d})$ are created (note that $\mathbb{E}[\|\boldsymbol{x}\|] = 1$). From this set 500 pairs are sampled with replacement. Then k of these pairs are encoded into a single vector and it is tested whether the $\boldsymbol{x}$ of a random encoded pair $(\boldsymbol{x}, \boldsymbol{y})$ can recovered with the cue $\boldsymbol{y}$. A vector counts a successfully recovered if $\hat{\boldsymbol{x}}$ is more similar to $\boldsymbol{x}$ than any other vector in the initial set of vectors. 1000 trials were run and averaged over for each encoding scheme and vector dimensionality. Because VTB requires the dimensionality d to be square, 256, 484, 1024, and 2025 were picked.

Figure 5.5a shows the results for encoding with binding. Both binding operators perform about the same. While this experiment essentially follows Recchia et al. (2015), it gives much better results. It is not clear whether they omitted an essential detail from their description or whether their implementation of the binding operation is flawed.

The results for encoding with tagging by a shift of one are shown in Fig. 5.5b and closely match the results presented in Recchia et al. (2015). Note that the same result applies to any other permutation matrix as the vector components can be reordered accordingly. Very similar results are obtained with orthogonal

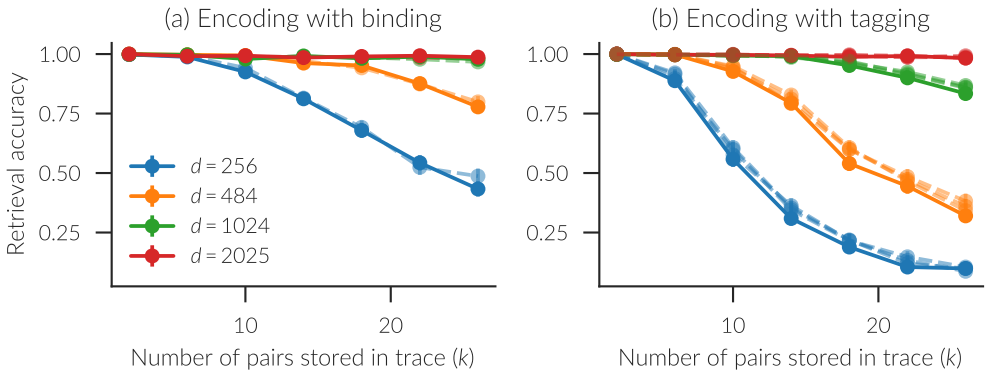


Figure 5.5.: Retrieval accuracy for (a) encoding with binding and (b) encoding with unitary tagging matrices. Solid lines show results for VTB. The dashed lines in (a) show results for binding with circular convolution, in (b) results for unitary circular convolution matrices, shift matrices, and orthogonal matrices without further indication as the results are almost identical. Error bars denote 95 % confidence intervals.

matrices, fixed unitary circular convolution, and unitary VTB. In all these cases, the pairwise binding capacity is below the encoding with binding, opposed to what Recchia et al. (2015) reported. Intuitively, this can be explained by the fact that encoding with binding is a sum of k vectors, where each vector has encoded information about the pair's constituents, whereas encoding with tagging is a sum of $2k$ vectors because each pair's constituent gets encoded separately. In other words, encoding with binding has a pairwise binding capacity about twice as high as encoding with tagging. In the plots, the retrieval accuracies for encoding with binding at $2k$ matches roughly with the retrieval accuracy with tagging at k, illustrating this fact.

Finally, we can observe that encoding with tagging completely fails with fixed non-unitary circular convolution matrices and does not do much better with non-unitary VTB matrices either (Fig. 5.6). This is likely due to the fact that repeated circular convolution with the same (non-unitary) vector leads to a super-exponential increase in length. That causes one pair (or even one vector of that pair) to be much stronger than all the other pairs, preventing their recovery.

These experiments clearly show that encoding with binding is preferable.

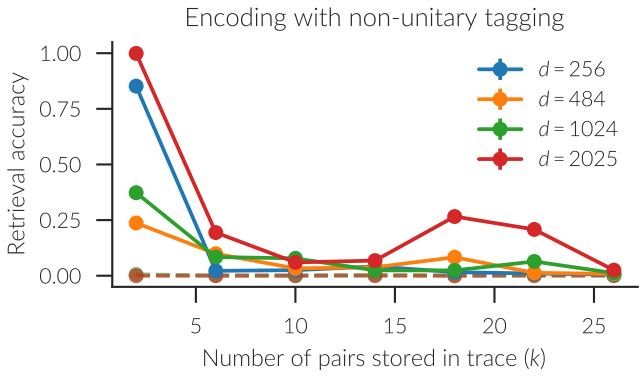


Figure 5.6.: Retrieval accuracy for encoding with non-unitary tagging matrices. Solid lines show results for VTB matrics, while dashed lines show results for circular convolution matrices. Error bars for 95 % confidence intervals are smaller than then marker size.

This is even more so the case, as retrieving items from an encoding with tagging is much more involved. Each potential cue vector $\boldsymbol{y}_i$ has to be recovered and compared to the actual cue $\boldsymbol{y}$ to determine the maximum similarity to the cue, and which $\boldsymbol{x}_i$ has to be decoded from the encoding. The encoding with binding allows direct querying of an $\boldsymbol{x}_i$ with a given $\boldsymbol{y}_i$. This makes for a simpler implementation in a neural network. That being said, if the main concern is not the implementation in a neural network, but the performance of the vector operations on a classical digital computer, one might come to a different conclusion as Recchia et al. (2015) did.

6. Optimized high-dimensional representation in spiking neurons

The implementation of the Semantic Pointer Architecture in a spiking neural network requires the representation of high-dimensional vectors with a certain accuracy. One of the main factors influencing the accuracy is the number of neurons. A higher number of neurons improves accuracy and, unfortunately, increases simulation times. If the accuracy of the representation can be improved in a different way, it would allow a reduction in the neuron number accordingly and a reduction in simulation times without sacrificing accuracy. I previously proposed one such optimization method (Gosmann and Eliasmith 2016), which improved the accuracy of SPA operations by up to 25 times. Here, I describe a more generally applicable method that supersedes the previous method by matching or exceeding the previously achieved performance. The new method also has a number of additional advantages. It does not require any prior simulation to obtain empirical noise estimates and is simpler in its implementation.

6.1. Types of error in neural representations

In the NEF the total mean representational error is given by

$$E_{\text{tot}}^2 = \left\langle E_{\text{tot}}^2(\boldsymbol{x}) \right\rangle_{x \in \mathcal{X}} = \left\langle \left\| \boldsymbol{x} - \hat{\boldsymbol{x}}(t) \right\|^2 \right\rangle_{t, x \in \mathcal{X}}. \tag{6.1}$$

As detailed by Eliasmith and C. H. Anderson (2003, pp. 47–48), the total error is constituted out of the error caused by spiking noise E_{n} and the error due to the static distortion E_{d} from the imperfect decoding:

$$E_{\text{tot}}^2(\boldsymbol{x}) = E_{\text{n}}^2(\boldsymbol{x}) + E_{\text{d}}^2(\boldsymbol{x}) \tag{6.2}$$

$$E_{\text{n}}(\boldsymbol{x}) = \left\langle \left\| \hat{\boldsymbol{x}}(t) - \langle \hat{\boldsymbol{x}}(t) \rangle_t \right\|^2 \right\rangle_t \tag{6.3}$$

$$E_{\mathrm{d}}(\boldsymbol{x}) = \left\| \boldsymbol{x} - \langle \hat{\boldsymbol{x}}(t) \rangle_t \right\|^2 . \tag{6.4}$$

The relation of the error terms is explained by the partitioning of the sum of squares in an ordinary least squares model (which is typically used to solve for decoders in the NEF). Note that the noise error depends on the decoding synapse. As $\tau_{\mathrm{syn}} \to \infty$, the noise error approaches zero ($E_{\mathrm{n}} \to 0$). Because the synapse limits how fast the neural representation can be updated, we get a trade-off of the noise in the system and how fast it reacts to new inputs.

Due to the neuron nonlinearities, finding analytical solutions for the error terms is likely not possible (except for constrained special cases). However, we can estimate the error terms from computational experiments. To do so, we sample $\boldsymbol{x} \in \mathcal{X}$ or use a regular grid of $\boldsymbol{x}$. Each $\boldsymbol{x}$ is then presented for some duration Δt_{ss} to reach the steady state and then $\hat{\boldsymbol{x}}(t)$ is measured for some sample duration $\Delta t_{\mathrm{sample}}$. Appropriate durations depend on the decoding synapse (longer synapses require more time to reach the steady state) and firing rate (a longer sampling duration is required for accurate estimates with low firing rates).

As the dimensionality of the higher-dimensional space increases, it becomes increasingly difficult to cover the whole space with samples from $\mathcal{X}$. Most of the time, though, we can treat the space as an isotropic hyperball, i.e., it does not matter along which direction we move through the space. This requires that the NEF ensemble's encoders are uniformly sampled from the hypersphere surface, which is usually the case (but there are some exceptions like certain implementations of a product network, Gosmann 2015, or thresholding ensembles, Section 4.4). Without loss of generality, we assume the representational radius of the hyperball to be $r = 1$ (as it is purely a scaling factor). The isotropy property allows us to cut through the center of the hyperball with a one-dimensional line. Measuring the error $E(x) = E(\boldsymbol{x})$ at m regularly spaced points $\boldsymbol{x}_i = (x_i, 0, \ldots, 0)^\top$ with $x_i = i * \Delta x - \Delta x/2$, $\Delta x = 1/m$, $1 \leq i \leq m$ along such a line, the mean error for the hyperball can be estimated as

$$E = \frac{\Omega_d}{V_d} \sum_{i=1}^{m} E(x_i) \cdot \Delta x \cdot r(x_i) \tag{6.5}$$

$$\Omega_d = \frac{2\pi^{\frac{d}{2}}}{\Gamma\left(\frac{d}{2}\right)} \tag{6.6}$$

$$V_d = \frac{\pi^{\frac{d}{2}}}{\Gamma\left(\frac{d}{2}+1\right)} \tag{6.7}$$

$$r(x) = \frac{1}{q}\sum_{i=1}^{q}\left|x - \left(1+\frac{1}{q}\right)\frac{\Delta x}{2} + i\frac{\Delta x}{q}\right|^{d-1} \tag{6.8}$$

where Ω_d is the d-dimensional solid angle, V_d the volume of a d-ball with radius $r = 1$, and $r(x)$ estimates the radius to the power of $d-1$ for an x with q evaluation points. This latter estimation of the radius across the Δx interval is necessary to not under- or overestimate the integral by a large amount. This would happen if only the radius at the exact evaluation point was used.

6.2. Properties of the error in neural representations

When looking at the representation of a spiking neural network, the noise error is the main factor to consider. It goes down by $O(1/\sqrt{n})$ where n is the number of neurons (Fig. 6.1a), whereas the distortion error decreases by $O(1/n)$. The total error is thus dominated by the noise error for a sufficient number of neurons (Eliasmith and C. H. Anderson 2003, Fig. 2.6). In contrast, for rate neurons without spiking noise ($E_{\mathrm{n}} = 0$) only the distortion error is relevant. Furthermore, with the Nengo default parameters, the increase in the noise error with dimensions d is of $O(d)$ (Fig. 6.1b).

When looking at the error along a line through the hyperball (Fig. 6.2), it becomes apparent that the distortion is mostly flat, but increases near the surface. The noise error is slightly larger in the center of the ball than towards the surface with higher dimensionalities (it is a flat line for $d = 1$). Both of these effects become more pronounced as the dimensionality increases. Moreover, the one-dimensional cut shows a distorted picture of the importance of different regions in the distortion. As the dimensionality increases, most of the volume of the hyperball is close to the surface. Thus, the distortion (and noise) close to -1 and 1 of the cut comes to dominate the mean error.

The main cause for the observed shape of the distortion is the uniform sampling of evaluation points from the hyperball (Fig. 6.3). When looking at the convex hull of the sample points, this hull is always smaller than the hyperball (even if some evaluation points are exactly on the surface). Thus, parts of the hyperball near the surface are not covered by the evaluation points

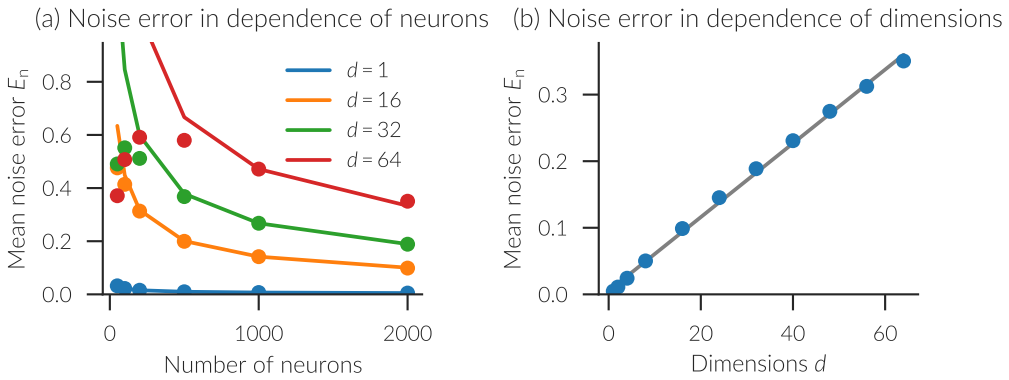


Figure 6.1.: The mean noise error E_n in dependence of (a) the number of neurons and (b) the number of dimensions. Scatter points show empirically determined values with 95 % confidence intervals smaller than the marker size. Solid lines in (a) are extrapolated from the mean values for 1000 neurons assuming the noise error is in $O(n)$. The solid gray line in (b) is a linear regression through all shown data points.

and are not considered in the least squares optimization of the decoders. As the number of dimensions increases, this becomes a bigger problem as the volume for a hyperball goes to zero as $d \to \infty$ (all of the ball is a surface). To show that this distortion is indeed caused by the partial covering, we can increase the radius of the hyperball for sampling the evaluation points slightly to cover more of the unit-ball. This is done in Fig. 6.4 for a 64-dimensional representation with 50 neurons per dimension (3200 in total). While this makes the distortion more even (mean distortion reduced by 0.011), it unfortunately also increases noise level (by 0.019) because evaluation points have a larger spacing now.[1]

Vectors in the SPA are often of unit-length and thus a good, low-distortion representation of the hyperball surface is desirable. Unfortunately, I am not aware of any method to flatten out the distortion at the surface without a higher increase in noise error. To completely cover the ball in a convex hull of evaluation points, it is necessary to place some evaluation points outside of the ball, which will cover and optimize for space outside of the representational

[1]Both values are statistically highly significant with $p < 0.0001$ as determined by bootstrapping.

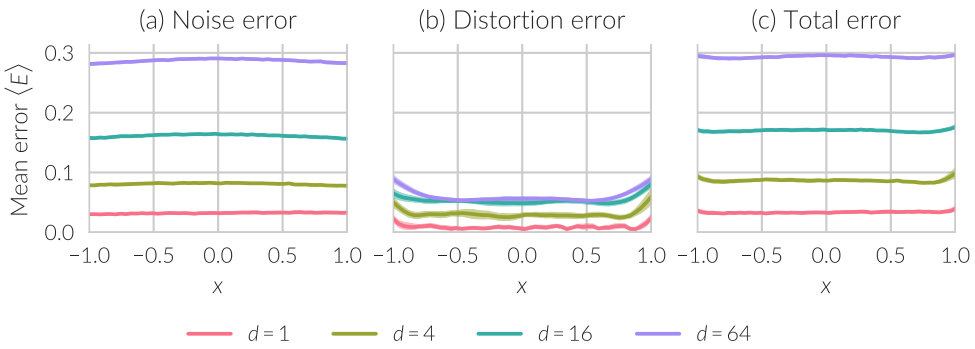


Figure 6.2.: Mean error along a line cut through the d-dimensional hyperball with $50d$ neurons. The (a) noise error component, (b) distortion error component, and (c) the total error are shown. The shaded regions indicate 95 % confidence intervals.

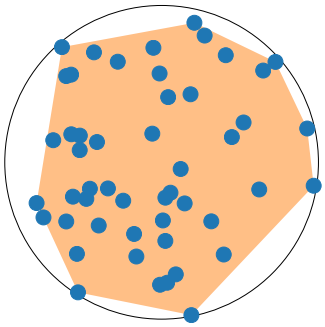


Figure 6.3.: Covering of the two-dimensional unit-circle with 50 uniformly sampled evaluation points. The orange, shaded region shows the convex hull which fails to cover the area close to the circle boundary.

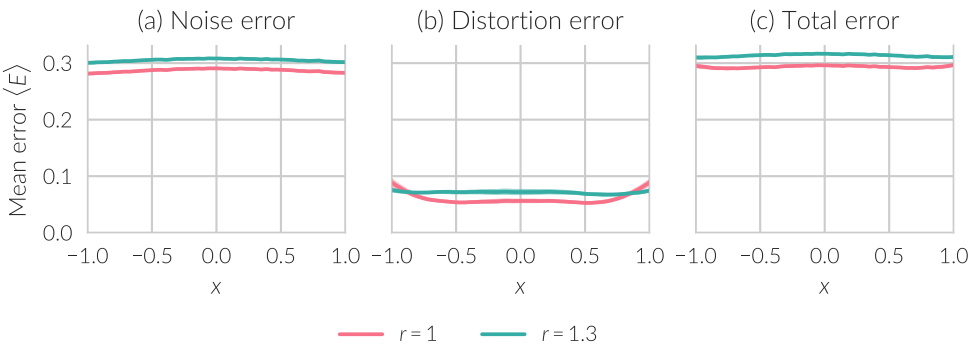


Figure 6.4.: Mean error along a line cut through the 64-dimensional hyperball with 3200 neurons and different radii r from which evaluation points are picked. The shaded regions indicate 95 % confidence intervals.

space. This leads to a trade-off of flatness of the distortion and baseline of the distortion. Ultimately, the problem of distortion is minor, as for spiking neurons the error is dominated by the noise component.

6.3. Effect of the intercept distribution on noise and distortion

The intercepts in Nengo are chosen to be distributed uniformly by default. In higher dimensions, this has the effect that most neurons are either almost never or almost always active for values in the representational space (Fig. 6.5). Neurons that are always silent do not contribute to the representation as they provide no information about the represented value. But also always active neurons only contribute minimally to the representation. Even though the firing rate still varies a bit over the representational space, the response curve is steepest and carries the most information closest to the intercept for most neuron models. The mapping of a small change in the represented value to a large change in firing rate also allows for a less noisy decoding as a single spike changes the decoded value less.

Thus, a better intercept distribution should have fewer neurons that are barely ever active, but should also distribute the intercepts so that there is an even

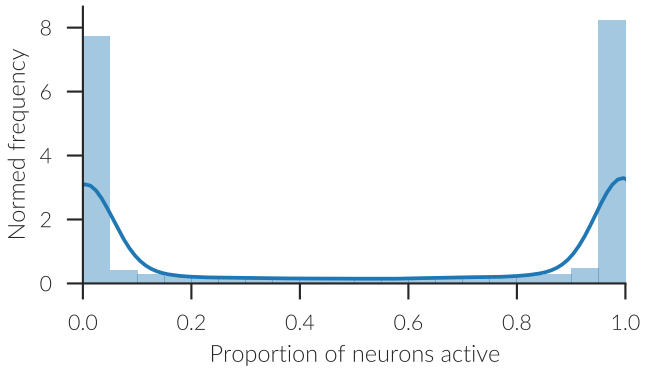


Figure 6.5.: The histogram (shaded bar plot) and kernel-density estimate (line) of the proportion of neurons active for uniformly sampled 64-dimensional vectors.

distribution of the fraction of space a neuron is active for. The latter criterion can be achieved by distributing the intercepts according to $\mathcal{CS}(d+2)$ where $\mathcal{CS}(d_{\mathcal{CS}})$ is the distribution of cosine similarities between random uniformly distributed $d_{\mathcal{CS}}$-dimensional unit-vectors. Its probability density function is given by (also see Fig. 6.6, derivation in Appendix A)

$$p_{\mathcal{CS}}(x; d_{\mathcal{CS}}) = \frac{1}{B\left(\frac{1}{2}, \frac{d_{\mathcal{CS}}-1}{2}\right)} \cdot \left(1-x^2\right)^{(d_{\mathcal{CS}}-3)/2}, \quad x \in [-1, 1]. \tag{6.9}$$

$\mathcal{CS}(d+2)$ is equivalent to the distribution of single coordinates of points uniformly sampled from within the d-dimensional unit-ball (Voelker, Gosmann, and Stewart 2017). Thus, by using this intercept distribution, the frequency of intercepts corresponds to the distribution of $\langle \boldsymbol{x}, \boldsymbol{e} \rangle$, $\boldsymbol{x} \in \mathcal{X}$, i.e., values in the representational space projected onto the (uniformly distributed) neuron encoders. Note that for $d = 1, \mathcal{CS}(d+2)$ is the uniform distribution. Figure 6.7 compares the relative number of neurons that do not fire for any of the evaluation points. For the standard uniform distribution, this fraction rises to above 0.35, but is close to zero with the cosine similarity intercept distribution.

While this gives a mathematical motivation to choose this intercept distribution, it must be shown empirically that it performs better than a uniform intercept distribution. In the following, I present a number of experiments to make this case and give an intuition about the effect of this particular intercept

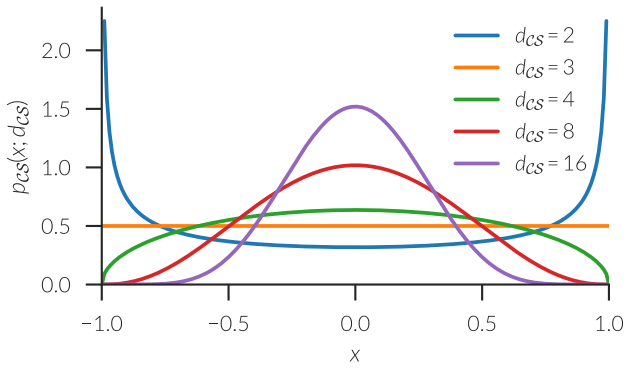


Figure 6.6.: PDF $p_{\mathcal{CS}}(x; d_{\mathcal{CS}})$ of the cosine similarity distribution.

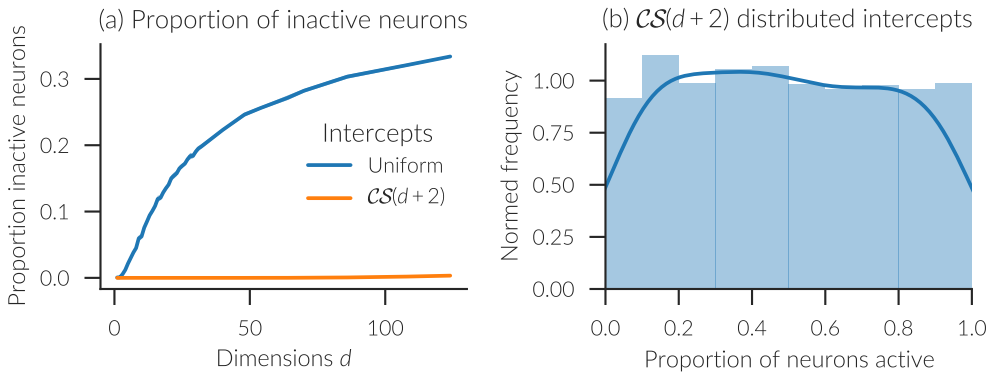


Figure 6.7.: (a) Proportion of neurons inactive for all 10 000 uniformly sampled d-dimensional vectors. The proportion is plotted for neural ensembles with a uniform intercept distribution (blue) and intercepts distributed according to $\mathcal{CS}(d + 2)$ (orange). The shaded areas denote 95 % confidence intervals. (b) Histogram (shaded bar plot) and kernel-density estimate (line) of the proportion of neurons active for uniformly sampled 64-dimensional vectors when distributing intercepts according to $\mathcal{CS}(d + 2)$.

distribution. Further evidence is presented in Appendix B. All experiments have been performed with $d = 64$ dimensional representations and $50d = 3200$ spiking LIF neurons unless otherwise noted. For each experiment 15 samples using different random number seeds were obtained. The benefit of choosing the $\mathcal{CS}(d + 2)$ intercept distribution is larger for increasing dimensionality, but 64 dimensions is sufficient to see a clear effect. For fewer dimensions the benefit is less, but not worse than the uniform intercept distribution (Appendix B). This is consistent with the $\mathcal{CS}(d + 2)$ distribution approaching a uniform distribution for $d \to 1$. The asymptotic scaling of the noise error component with the number of neurons does not change for the given choices of the intercept distribution. Any neuron number that is large enough for the noise error to dominate can be used in these experiments, and noise error values for other neuron numbers can be extrapolated. This might invalidate the results for small neuron numbers, but such cases should be considered as special for any intercept distribution. Finally, spiking LIF neurons are of primary interest, as they are the most commonly used neuron model in the NEF. But, I give some consideration to other neuron types in Appendix B. Any stated differences in error are based on the aforementioned parameters. Furthermore, all stated differences have been found to be statistically highly significant with $p < 0.0001$ using bootstrapping.

The basic effects of the $\mathcal{CS}(d+2)$ intercept distribution can be seen in Fig. 6.8. The total mean error is reduced by -0.130. This is mainly due to a reduction in the noise error. The distortion seemingly increases, but because most of the volume of the space is near the surface and the distortion there ends up a little bit lower, the total mean distortion also decreases (-0.020). However, where the space is distorted changes. While the uniform distribution leads to an even distortion except towards the hypersphere surface, the cosine similarity distribution gives a distortion that varies more across the space. In particular, there is a ring of higher distortion between the center and the surface of the hyperball and another such ring around the surface.

In general, there is a noise-distortion trade-off. Reducing the noise error by changing the intercept distribution, leads to a more uneven distortion and can potentially increase the total distortion. For spiking neurons with short synaptic time constants, the noise error is usually much higher and thus the change in the distortion is often negligible. Longer synaptic time constants shift this trade-off as the noise error is lower. Still, for biologically realistic time constants of up to 0.1 s, the cosine similarity intercept distribution performs better. It is also

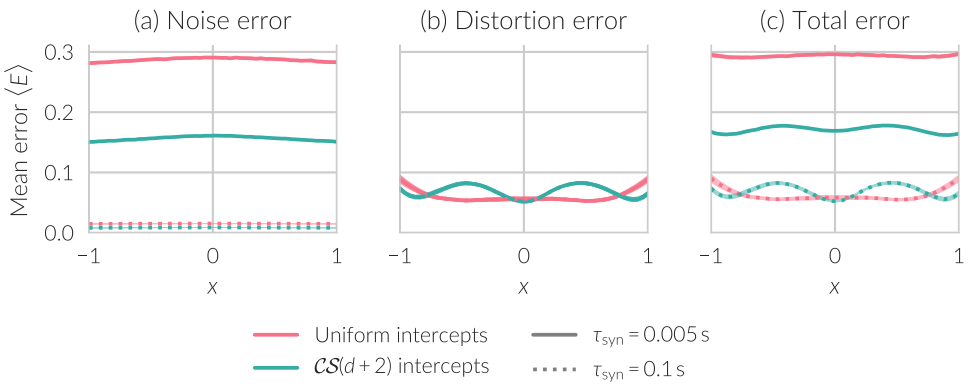


Figure 6.8.: Mean error along a line cut through the 64-dimensional hyperball with 3200 neurons, different intercept distributions (pink vs. turquoise) and different synaptic time constants (solid vs. dotted line). The shaded regions indicate 95 % confidence intervals.

worth noting that this trade-off is slightly affected by the regularization term when solving for the decoders (Fig. 6.9). A higher regularization decreases the noise as the decoders get less sensitive to small fluctuations, but increases the distortion towards the hypersphere surface. The opposite effects are observed with less regularization.

Interestingly, the cosine intercept distribution not only reduces the noise by a constant factor compared to the uniform distribution, it also improves the scaling of the noise error from $O(d/\sqrt{n})$ to $O\left(d^{3/4}/\sqrt{n}\right)$ as shown in Fig. 6.10.

So far, we only considered spiking neurons, but the NEF also allows the use of rate neuron models. In that case, the noise error is zero because the firing rate can be represented with arbitrary precision. With the default parameters, the cosine similarity distribution still gives a slightly lower total distortion (-0.020). However, due to the non-existent noise, the default regularization of $\lambda = 0.1$ is too large and a lower error can be obtained by reducing the regularization to $\lambda = 0.01$. Then the uniform distribution performs better, while the cosine similarity intercept distribution increases the distortion (by 0.0262).

These results demonstrate the benefit of the $\mathcal{CS}(d + 2)$ intercept distribution for the most commonly used neuron types and parameters in the NEF. Technically, the optimal intercept distribution depends on the exact parameters set and can differ depending on them. It is not possible to do an exhaustive

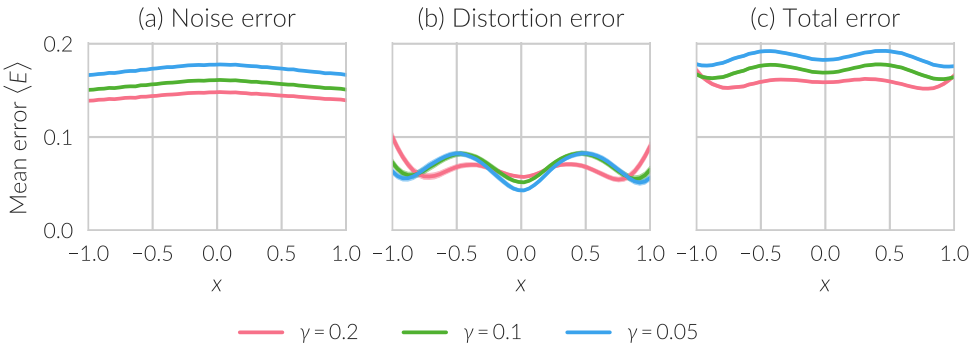


Figure 6.9.: Mean error along a line cut through the 64-dimensional hyperball with 3200 neurons and different regularization λ. The shaded regions indicate 95 % confidence intervals.

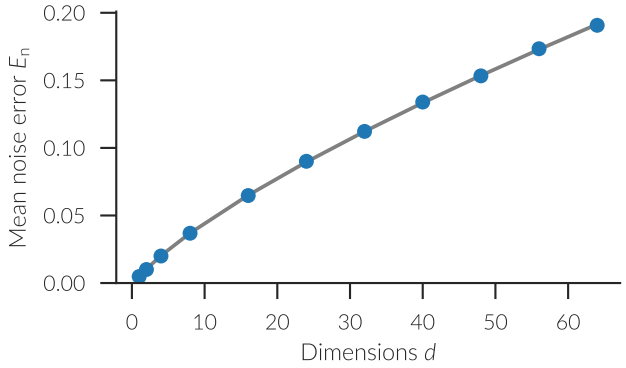


Figure 6.10.: The mean noise error E_n with $\mathcal{CS}(d+2)$ distributed intercepts in dependence of the number of dimensions. Scatter points show empirically determined values with 95 % confidence intervals smaller than the marker size. The solid gray line in is a linear regression fitted to $d^{3/4}$.

comparison due to the multitude of parameters and valid parameter choices. Nevertheless, Appendix B demonstrates the benefit of distributing spiking neuron intercepts according to $\mathcal{CS}(d+2)$ for a wide range of parameter manipulations including varied neuron numbers, dimensionalities, neuron models, and nonlinear function decoding.

It is also worth noting that the present analysis assumes that all values in the representational space $\mathcal{X}$ are uniformly distributed. If certain values appear more frequently than others, the optimal intercept distribution might change. The experimental procedure used here to compare intercept distributions might still be used, but the error needs to be weighted by the frequency of the represented values $\boldsymbol{x}$.

6.4. Optimized Semantic Pointer Representation

The NEF only requires multiple vector dimensions to be represented in a single ensemble if functions with a nonlinear interaction of the individual dimensions is decoded. Such interactions are rarely needed in the Semantic Pointer Architecture and there are a number of reasons to split up the d-dimensional vector into q vectors of $d_q = d/q$ dimensions.

First of all, solving for decoders requires the inversion of an $n \times n$ matrix with a runtime complexity of $O(n^3)$ where n is the total number of neurons in the ensemble. When instead using q ensembles with n/q neurons each to represent d_q-dimensional segments of the vector, only q inversions of $n/q \times n/q$ matrices are required, which give a runtime complexity of only $O(n^3/q^2)$. Thus, solving for the decoders can be a lot more efficient when splitting up a vector into multiple ensembles for the representation. Second, the noise error grows only in $O(\sqrt{q/n})$ if d_q is kept constant. In other words, adding d_q more dimensions, requires only n/q additional neurons to keep the noise error constant.

Splitting up the input space in this manner, however, introduces a slight complication. While we can assume the full vectors $\boldsymbol{x} \in \mathcal{X}$ to be uniformly distributed, this is not the case for the d_q-dimensional subvectors. The distribution of the subvectors clusters around shorter vectors. One way to account for that, is to reduce the representational radius r of the ensembles, which improves the representation of the most common vectors, but worsens the representation of subvectors that fall outside of that radius (which might still occur occasionally). Gosmann and Eliasmith (2016) describe in detail how to find a good value for

r.

Here, I use a different method, where both the intercept distribution and evaluation point distribution are set to the cosine similarity distribution $\mathcal{CS}(d+2)$ (note that this uses the dimensionality of the complete vector, not d_q). As before, the distribution of evaluation points accounts for the fact that the represented values are no longer uniformly distributed. This can be seen as setting a "soft" radius. The prior radius adjustment method effectively changed the intercept and evaluation point distributions to cover a hyperball of reduced radius. By using the cosine similarity distribution instead, most of these points are inside such a hyperball, but some evaluation points are still allowed to fall outside of such a hard radius cutoff.

To verify the performance, I repeated the benchmarks from Gosmann and Eliasmith (2016). A randomly moving unit-length semantic pointer is fed as input to the set of ensembles and the distribution of resulting root mean square error (RMSE) in the representation is measured in each time step. The first 0.5 s are discarded to allow for the delayed rise of the neural firing rates at the beginning of the simulation. In comparison to Gosmann and Eliasmith (2016), the total simulation time has been halved to 5 s and only 15 instead of 20 independent simulations have been run per condition. These numbers are still sufficient to give highly significant results.

The results for a 64-dimensional representation are shown in Fig. 6.11. All the mean RMSEs for one value of d_q are significantly different ($p < 0.0001$, determined with bootstrapping), with the exception of the radius adjustment method compared to the default parameters when $d_q = d = 64$ ($p > 0.93$). This latter result is expected because for $d_q = d$, the radius adjustment method keeps a unit radius and has no effect. Further, the previous result of the radius method increasingly reducing the error as $d_q \to 1$ is reproduced. The method of setting the intercepts and evaluation points performs better for $d_q = 16$, which is a typical choice in SPA model, and the same improvement is obtained for larger values of d_q in contrast to the radius adjustment. For $d_q = 1$ it is slightly worse and has a long tail, but it is still a much lower error than obtained with the default parameters. Overall, setting the intercepts and evaluation points gives a significant improvement in a wider range of cases and is only slightly worse than the radius adjustment for small d_q.

In addition to pure representation, it is common to compute dot products in the context of the SPA. For the dot product the element-wise products are computed in separate product networks and summed afterwards. In these

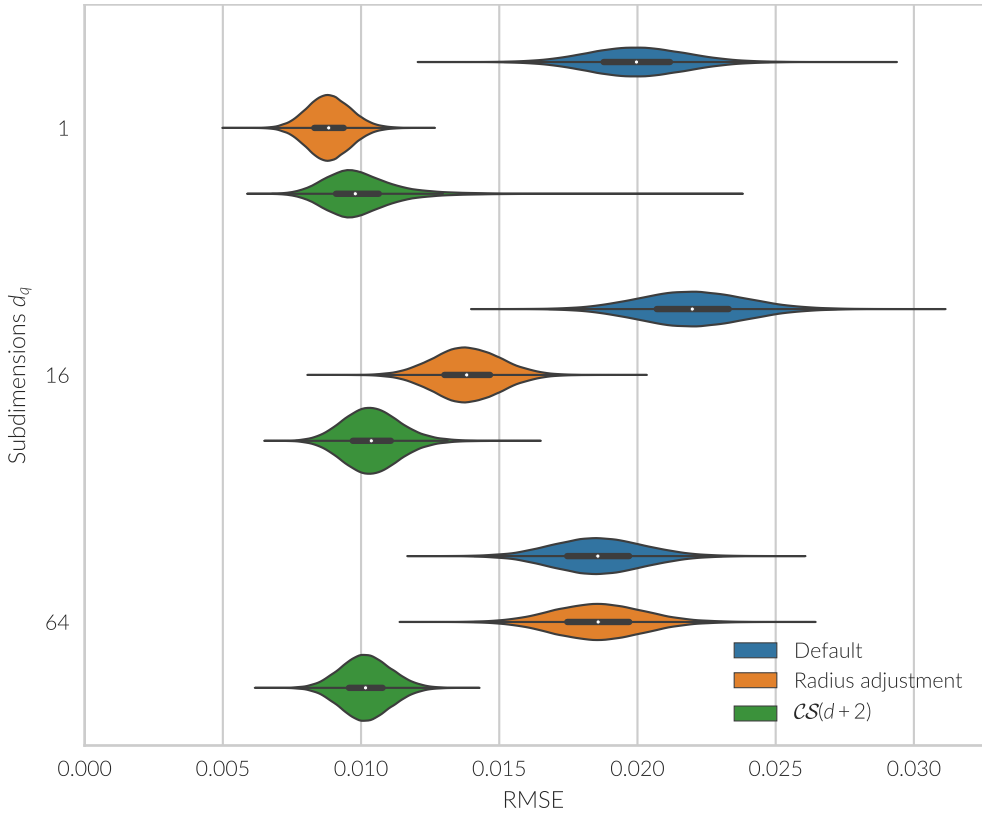


Figure 6.11.: Distribution of the RMSE in the representation of 64-dimensional unit-vectors. The violinplots visualize the distribution with a kernel-density estimate mirrored horizontally. The midlines show box plots with mean (white) and quartiles (thick black). Results obtained with the Nengo default parameters are blue, with the radius adjustment method from Gosmann and Eliasmith (2016) are orange, and using the $\mathcal{CS}(d+2)$ distribution for intercepts and evaluation points are green. Along the y-axis the number of subdimensions d_q represented in a single ensemble are arranged.

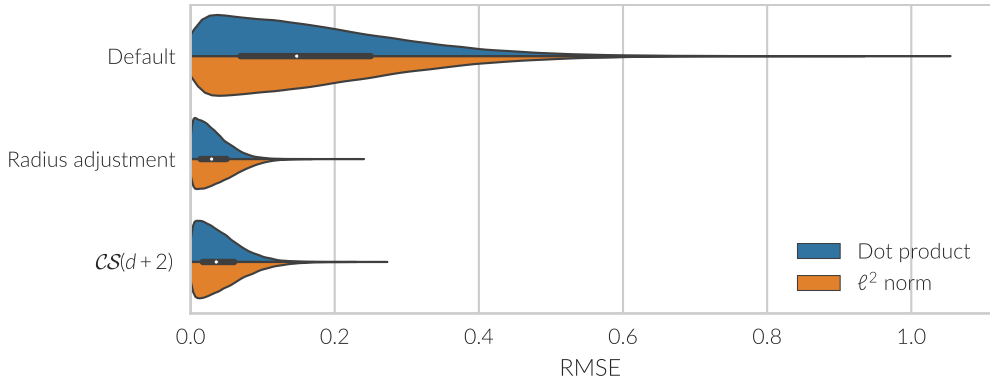


Figure 6.12.: Distribution of the RMSE in the calculation of dot products and vector lengths. The violinplots visualize the distribution with a kernel-density estimate. The midline of shows a box plot with mean (white) and quartiles (thick black).

product networks the sum (and difference) of the two input elements needs to be represented (see Section 4.5), which means that the represented value is distributed according to the sum of two cosine similarity distributions in the general case. However, the dot product is often used to check the similarity of two unit-vectors and a result of one should be obtained for identical vectors. For identical vectors, the two inputs to the element-wise products are the same and the sum follows the distribution of a single input stretched out by a factor of 2. Thus, it is appropriate to use the $\mathcal{CS}(d+2)$ distribution for intercepts and evaluation points in the product networks used for dot products. In the general case of non-identical inputs, this might not be fully optimal, but still performs well (Fig. 6.12). It is also only minimally worse than the previous radius adjustment method.

Furthermore, the benchmark was only run for 64 dimensions with 200 neurons per dimension. However, the same conclusions as in Gosmann and Eliasmith (2016) can be expected to hold for different dimensionalities and neuron numbers. Increasing the dimensionality improves the benefit of adjusting the radius, or the intercept and evaluation point distributions. Similarly, the neuron number can be reduced to achieve faster simulation times without an increase in error.

Part II.

The Context-Unified Encoding memory model

7. The Ordinal Serial Encoding Model

The Ordinal Serial Encoding (OSE) model (Choo 2010) is an NEF and SPA based model of serial recall. It was able to reproduce various effects found in human recall data such as the primacy effect, recency effect, and transposition gradients in serial and delayed forward recall. Within the context-unified encoding (CUE) memory model it provides the basis for the short-term memory component.

In the OSE, m presented items $\boldsymbol{v}_i$ are bound to fixed position vectors $\boldsymbol{p}_i$ and stored in two memory traces

$$\boldsymbol{m}^{\text{stm}} = \sum_{i=1}^{m} \gamma^{m-i} \mathcal{B}(\boldsymbol{v}_i, \boldsymbol{p}_i) \tag{7.1}$$

$$\boldsymbol{m}^{\text{epis}} = \sum_{i=1}^{m} \rho^{m-i} \mathcal{B}(\boldsymbol{v}_i, \boldsymbol{p}_i) \tag{7.2}$$

with decay factor $\gamma < 1$ and scaling factor $\rho > 1$. The memory traces $\boldsymbol{m}^{\text{stm}}$ and $\boldsymbol{m}^{\text{epis}}$ represent the short-term and episodic memory store, respectively. The binding operation $\mathcal{B}$ used here is circular convolution, but it could be worth exploring the effect of other binding operations in the future. The recall of an item is given by unbinding the corresponding position vector as

$$\boldsymbol{v}_i \approx \mathcal{B}^{+}\left(\boldsymbol{m}^{\text{stm}} + \boldsymbol{m}^{\text{epis}}, \boldsymbol{p}_i\right). \tag{7.3}$$

These encoding equations produce the primacy and recency effect due to the differential effect of the decay and scaling factors γ and ρ.

For the neural implementation, each memory trace can be stored in an integrator with some additional processes for updating and unbinding the recalled item. For the integration within the CUE memory model, the episodic memory trace is replaced by a version based on the temporal context model presented in the next chapter. This introduces a more plausible episodic memory, storing experiences in actual synaptic weight changes rather than in the activities of a neural population. In addition, the recall process needs to be adjusted to integrate information from the exchanged episodic memory component (Chapter 10).

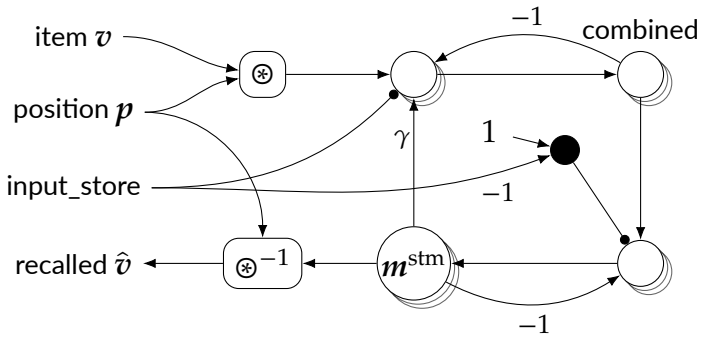


Figure 7.1.: Implementation of the OSE short-term memory trace m^{stm} with the NEF. See text for additional details.

7.1. Neural STM implementation

In the CUE model, the short-term memory buffer of the OSE model is implemented as depicted in Fig. 7.1. The network gets an item and position Semantic Pointer as input which are bound together. The bound result is added into the memory trace stored in combined as long as it does not receive the input_store signal. Once the input_store signal is received, the contents from combined are transferred to the m^{stm} populations. Items are decoded from the memory via the approximate inverse circular convolution. The decay factor of $\gamma = 0.9775$ is taken from the original OSE implementation (Choo 2010) and is implemented on the connection from m^{stm} to combined.

7.2. Neural position counting

For the OSE it is necessary to keep track of the ordinal position of the current item. The CUE model extends the OSE to do this also in neurons. Figure 7.2 shows the network implementing this functionality. All ensembles in this network are implementing a threshold at zero so that represented values are always positive. The state ensemble array has one ensemble for each possible position and only the ensemble for the current position is active. This is ensured by providing a small negative bias (-0.2) to all ensembles to prevent spontaneous activity. Furthermore, a recurrent connection with $\tau_{\text{syn}} = 0.1\,\text{s}$ decoding a constant of 1.2 keeps the current position in a state of stable activity.

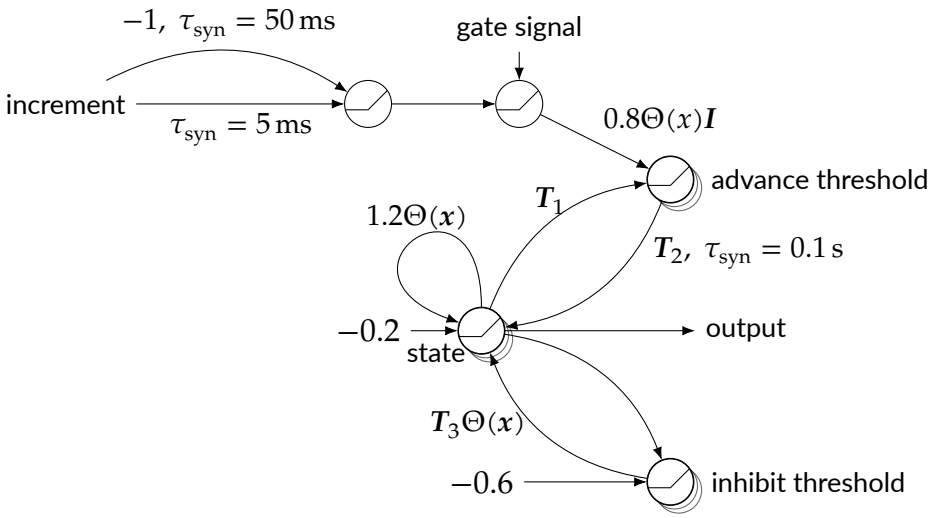


Figure 7.2.: Implementation of position counting with the NEF. See text for details.

To advance to the next position a signal with a rising edge has to be provided to the increment input. To detect the rising edge, a differentiator ensemble is used that receives its input via two connections where one connection has a fast synaptic time constant ($\tau_{syn} = 5\,ms$) and the other connection has a slow synaptic time constant ($\tau_{syn} = 50\,ms$) and a transform of -1 (Section 4.1). The output is fed through a gate ensemble that can be inhibited to prevent position increments.

Then the Heaviside step function is decoded from the gate signal and fed into the advance threshold ensemble array scaled by a factor of 0.8. The output of state is also fed into advance threshold with the transform

$$\boldsymbol{T}_1 = \begin{bmatrix} 0 & -1 & -1 & \cdots \\ -1 & 0 & -1 & \cdots \\ -1 & -1 & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{7.4}$$

which inhibits all ensembles except the one corresponding to the current position. This ensemble only becomes active when a rising edge for the increment input is detected. The advance threshold ensemble projects back to the state

ensembles with a transform of

$$\boldsymbol{T}_2 = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ 2 & 0 & \cdots & 0 & 0 \\ 0 & 2 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 2 & 0 \end{bmatrix} \tag{7.5}$$

to excite the population representing the next position.

When the next position gets active, the old position needs to be inhibited at some point to prevent two positions from being active at the same time. This is done via the inhibit threshold ensemble array. It receives a bias input of -0.6 and input from the decoded constant from state. Once the threshold (decoded as Heaviside step function) is exceeded, the previous and next item are inhibited with the transform given by

$$\boldsymbol{T}_3 = -\begin{bmatrix} 0 & 2 & 0 & \cdots & 0 \\ 0 & 0 & 2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 2 \\ 2 & 0 & 0 & \cdots & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 & \cdots & 2 & 0 \\ 0 & 0 & \cdots & 0 & 2 \\ 2 & 0 & \cdots & 0 & 0 \\ 0 & 2 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \end{bmatrix}. \tag{7.6}$$

An example of how these component interact to advance the represented position is given in Fig. 7.3.

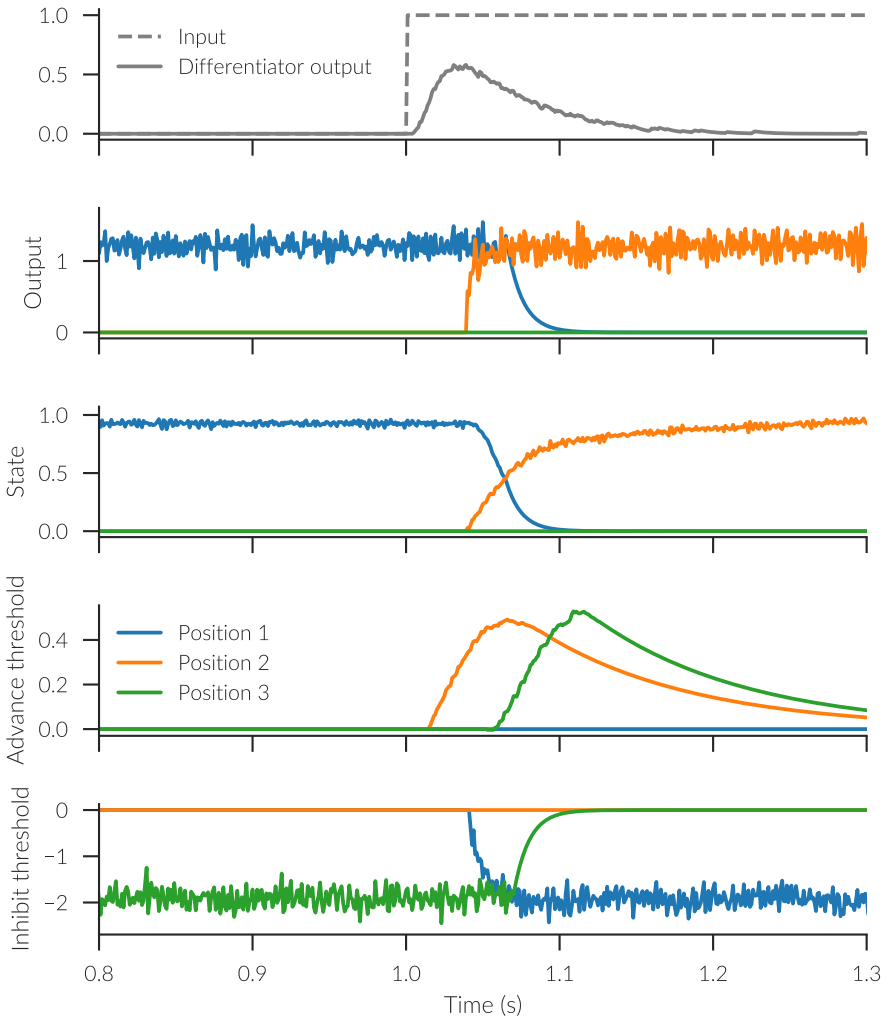


Figure 7.3.: Position increment in the position counting network. The top-most plot shows the input signal and output of the differentiator. The following plots from top to bottom show: Heaviside output of the network, representation in the state ensembles, transformed input to the state ensembles from the advance threshold ensembles, and transformed input from the inhibit threshold ensembles.

8. The Temporal Context Model

The temporal context model (TCM) was proposed by Howard and Kahana (2002) as a model of free recall. It matched data of immediate, delayed, and continuous distractor recall tasks. As the distractor task used in the delayed and continuous distractor condition is designed to prevent active rehearsal, this model is likely to address more long-term, synaptic storage as opposed to the short-term OSE model. Similar to a number of other memory models, the TCM assumes a time varying context signal that items are associated with. But unlike those other models, this context is based on the items themselves rather than being randomly generated.

In particular, items in the TCM are represented as orthogonal vectors $\boldsymbol{f}_i$ and the context signal is also a vector $\boldsymbol{c}$. If we relax the orthogonality constraint on the items to almost (instead of perfectly) orthogonal, we can use Semantic Pointers for these vectors. To associate items and contexts, two association matrices are used. The $\boldsymbol{M}^{\mathrm{CF}}$ matrix represents the associations from a context to an item and is constructed as an outer product matrix as

$$\boldsymbol{M}^{\mathrm{CF}} = \sum_i \boldsymbol{f}_i \boldsymbol{c}_i^{\top}. \tag{8.1}$$

Note that the $\boldsymbol{M}^{\mathrm{CF}}$ matrix can be easily updated by adding another item/context outer product. The $\boldsymbol{M}^{\mathrm{FC}}$ matrix is used to retrieve a context vector $\boldsymbol{c}_i^{\mathrm{IN}} = \boldsymbol{M}^{\mathrm{FC}} \boldsymbol{f}_i$ to update the current context according to the *evolution equation* (also see Fig. 8.1a)

$$\boldsymbol{c}_i = \theta_i \boldsymbol{c}_{i-1} + \beta \boldsymbol{c}_i^{\mathrm{IN}} \tag{8.2}$$

where β is a free parameter controlling how fast the context drifts and $0 < \theta_i \leq 1$ is determined to ensure unit length of $\boldsymbol{c}_i$ in each timestep as

$$\theta_i = \sqrt{1 + \beta^2 \left[\left\langle \boldsymbol{c}_{i-1}, \boldsymbol{c}_i^{\mathrm{IN}} \right\rangle^2 - 1 \right]} - \beta \left\langle \boldsymbol{c}_{i-1}, \boldsymbol{c}_i^{\mathrm{IN}} \right\rangle. \tag{8.3}$$

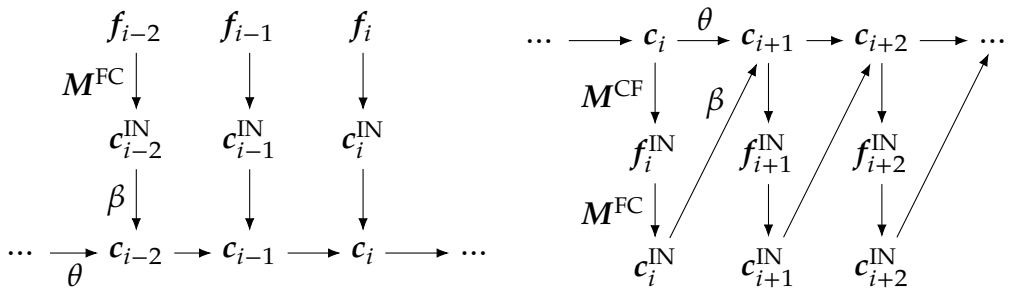


Figure 8.1.: Evolution of the context in the TCM during (a) item presentation and (b) recall.

In the CUE model, however, θ_i is fixed to the asymptotic value for $\langle \boldsymbol{c}_{i-1}, \boldsymbol{c}_i^{\mathrm{IN}} \rangle \to 0$

$$\theta_i = \sqrt{1 - \beta^2}. \tag{8.4}$$

In the TCM this corresponds to the assumption that item i has not been presented for a sufficiently long time which results in a retrieved context $\boldsymbol{c}_i^{\mathrm{IN}}$ that is almost orthogonal to the current context. This change is further motivated by still producing a good match to the data and simplifying the neural implementation as no dynamic scaling of a vector is required. Such scaling would require a product network for each vector dimension in the NEF. Equation (8.2) introduces the asymmetric bias to forward recall into the model (Fig. 8.2). While the similarity of contexts $\langle \boldsymbol{c}_i, \boldsymbol{c}_j \rangle$ is symmetric for the lag $j - i$, $\boldsymbol{c}_i^{\mathrm{IN}}$ is only included in context vectors $\boldsymbol{c}_j$ with $j \geq i$.

Finally, the associations from items to context $\boldsymbol{M}^{\mathrm{FC}}$ need to be updated, so that a recalled item can be used to update and partially restore a previous context to retrieve further items. In the original TCM, this update is given by

$$\boldsymbol{M}_{i+1}^{\mathrm{FC}} = \boldsymbol{M}_i^{\mathrm{FC}} \tilde{\boldsymbol{P}}_{f_i} + a_i \boldsymbol{M}_i^{\mathrm{FC}} \boldsymbol{P}_{f_i} + b_i \boldsymbol{c}_i \boldsymbol{f}_i^{\top} \tag{8.5}$$

$$a_i = \gamma b_i \tag{8.6}$$

$$b_i = \frac{1}{\gamma^2 + 2\gamma \left\langle \boldsymbol{c}^{\mathrm{IN}}, \boldsymbol{c}_i \right\rangle + 1} \tag{8.7}$$

with projection operators $\boldsymbol{P}_v = \boldsymbol{v}\boldsymbol{v}^{\top}/\|\boldsymbol{v}\|^2$, $\tilde{\boldsymbol{P}}_v = \boldsymbol{I} - \boldsymbol{P}_v$, and a free parameter γ specifying the relative contribution of previously associated context $\boldsymbol{c}_i^{\mathrm{IN}}$

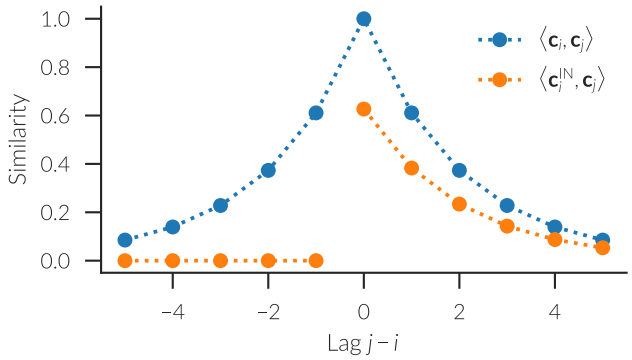


Figure 8.2.: Similarity of the context to itself $\langle \boldsymbol{c}_i, \boldsymbol{c}_j \rangle$ and to the retrieved context $\langle \boldsymbol{c}_i^{\text{IN}}, \boldsymbol{c}_j \rangle$ for different lags $j - i$.

and new context $\boldsymbol{c}_i$. Again, to facilitate the neural implementation, the exact weighting of $\boldsymbol{c}_i^{\text{IN}}$ and $\boldsymbol{c}_i$ is relaxed while still achieving a good match to data. Instead of splitting $\boldsymbol{M}_i^{\text{FC}}$ into components parallel and orthogonal to $\boldsymbol{f}_i$, $\boldsymbol{c}_i \boldsymbol{f}_i^\top$ is added directly into the matrix with a fixed parameter b,

$$\boldsymbol{M}_{i+1}^{\text{FC}} = \boldsymbol{M}_i^{\text{FC}} + b \boldsymbol{c}_i \boldsymbol{f}_i^\top. \tag{8.8}$$

Given a context $\boldsymbol{c}$, a mixture of associated items can be recalled as $\boldsymbol{f}^{\text{IN}} = \boldsymbol{M}^{\text{CF}} \boldsymbol{c}$. To retrieve a single item some form of cleanup has to performed. Once such a single item has been recalled, the item can be used to recall the associated context as $\boldsymbol{M}^{\text{FC}} \boldsymbol{f}^{\text{IN}}$ which in turn can be used to update the current context according to (8.2). The updated context allows recalling further items (Fig. 8.1b).

Different cleanup strategies for the recalled item vector can be used. In the original TCM model, a set of activities $a_i = \boldsymbol{f}_i^\top \boldsymbol{f}^{\text{IN}}$ was obtained and used to make a probabilistic decision according to Luce's choice rule. The probability of retrieving item $\boldsymbol{f}_i$ is given as

$$P(\boldsymbol{f}_i | \boldsymbol{f}^{\text{IN}}) = \frac{\exp\left(\frac{2a_i}{\tau}\right)}{\sum_j \exp\left(\frac{2a_j}{\tau}\right)} \tag{8.9}$$

with a parameter τ that specifies the sensitivity to the activities.

The version of the TCM model presented by Sederberg, Howard, and Kahana (2008) uses a winner-take-all process based on the leaky, competing accumulator model (Usher and McClelland 2001). In this model, for each possible

item a leaky integrator integrates evidence over time. At the same time, the integrators inhibit each other laterally. The dynamics can be described by

$$\boldsymbol{x}_s = \boldsymbol{x}_{s-1} + \frac{1}{\tau}\left(\boldsymbol{u} - \kappa \boldsymbol{x}_{s-1} - \lambda \boldsymbol{L}\boldsymbol{x}_{s-1}\right) + \boldsymbol{\eta} \tag{8.10}$$

where $\boldsymbol{u}$ is the scaled vector of inputs determined from $\boldsymbol{M}^{\mathrm{CF}}\boldsymbol{c}$ with the current context, κ the leak rate, λ the lateral inhibition, $[\boldsymbol{L}]_{ij} = 1 - \delta_{ij}$ the lateral inhibition matrix, and $\boldsymbol{\eta}$ normal distributed random noise. This is a more detailed description of how the brain might actually decide for a single item. However, it is problematic to incorporate within a larger scale neural model under noisy conditions as detailed in Chapter 10. For this reason, a different winner-take-all process described in that chapter is used.

8.1. Neural context update

A neural implementation of the TCM needs to implement the updating of the $\boldsymbol{M}^{\mathrm{FC}}$ and $\boldsymbol{M}^{\mathrm{CF}}$ matrices discussed in Chapter 9, the recall of items discussed in Chapter 10, and updating of the context given by (8.2), discussed in the remainder of this chapter. Despite being a simple equation, a number of different implementation approaches are potentially viable. However, only one of these methods was successful in matching the human data when incorporated into the complete model. It is still instructive to compare these different approaches and consider why they fail.

8.1.1. Bounded integrator

Equation (8.2) assumes discrete steps, but for a neural implementation a continuous formulation is more natural and given by

$$\frac{\mathrm{d}\boldsymbol{c}}{\mathrm{d}t} = \left(\bar{\theta} - 1\right)\boldsymbol{c} + \bar{\beta}\boldsymbol{c}^{\mathrm{IN}}\,. \tag{8.11}$$

This equation is easily implemented with a neural integrator for a constant $\bar{\theta}$ and $\bar{\beta}$. However, there is no limit on the integration of $\boldsymbol{c}^{\mathrm{IN}}$ anymore so that the proportion of $\boldsymbol{c}^{\mathrm{IN}}$ added into $\boldsymbol{c}$ can exceed β. To add at most $\beta\boldsymbol{c}^{\mathrm{IN}}$ to the context $\boldsymbol{c}$, we can gate the input to the integrator and add a network computing the dot product between $\boldsymbol{c}$ and $\boldsymbol{c}^{\mathrm{IN}}$. After thresholding the dot product at

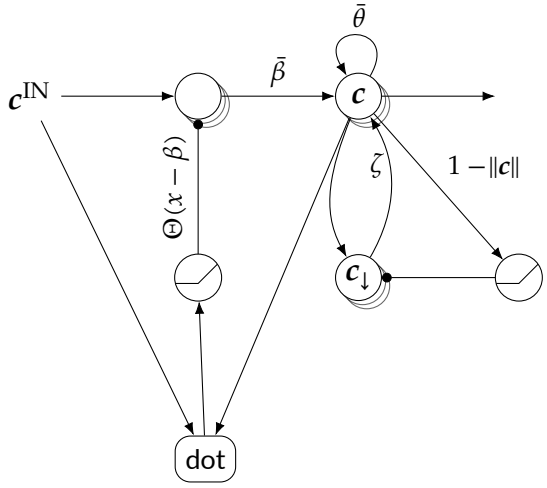


Figure 8.3.: Bounded integrator network.

β, it can be used to suppress the input by inhibiting the gate ensembles (see Fig. 8.3). Furthermore, in the original TCM $\boldsymbol{c}$ was kept at unit length while the integration has no such bound. To keep the unit length, we can project $\boldsymbol{c}$ to another population $\boldsymbol{c}_{\downarrow}$ which projects back to the integrator with a transform of $\zeta = -0.1$. Picking a ζ closer to zero allows the $\boldsymbol{c}$ vector exceed unit length by a larger amount while the integrator receives input and will increase the time required to settle back to unit length, whereas a large magnitude of ζ can lead to oscillatory behaviour. The $\boldsymbol{c}_{\downarrow}$ population needs to be controlled to only provide the inhibitory input to the integrator as long as $\|\boldsymbol{c}\| > 1$. This is achieved by decoding the length of $\boldsymbol{c}$ from the integrator and thresholding it at 1. As long as the threshold is not exceeded $\boldsymbol{c}_{\downarrow}$ is inhibited.

To investigate the behaviour of the network, it was fed with new context vectors $\boldsymbol{c}^{\text{IN}}$ at rate of one vector per second. These vectors were either orthogonal or had a cosine similarity of 0.6 between successive pairs. Figure 8.4 shows the mean similarity of the context vector to itself with given time lag. For (almost) orthogonal vectors $\boldsymbol{c}^{\text{IN}}$, the similarity between the context vectors is close under the target. For non-orthogonal vectors with $\langle \boldsymbol{c}_i^{\text{IN}}, \boldsymbol{c}_{i+1}^{\text{IN}} \rangle \approx 0.6$, however, the similarity of the context vectors is by far larger than the target. This is caused by the input already being similar to the context and thus stopping the update too early. Note that it is not sufficient to simply adjust β as depending on the similarity of the inputs it needs to either be incremented or decremented.

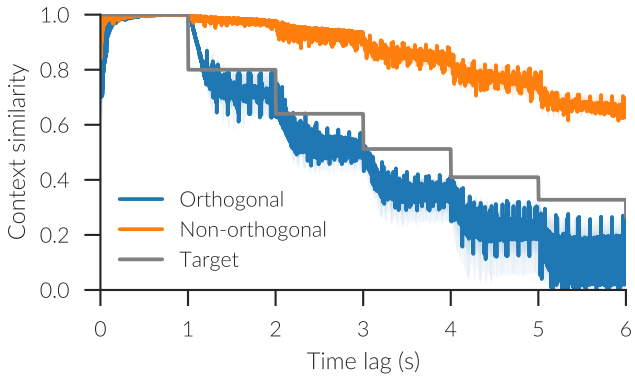


Figure 8.4.: Decay in context similarity with the bounded integrator network given almost orthogonal inputs and inputs with a similarity of approximately 0.6. The desired similarity is given by the gray line. Shaded regions indicate 95 % confidence intervals.

8.1.2. Alternating update of two memories

With a single integrator, we have to rely on the dot product between the input vector and current context as a measure of β. This dot product is biased in different directions if the input vectors have differing similarities. To circumvent this, we need to use two gated memory populations that are updated in alternating fashion. Then the output of the old context and input vector can be combined according to $\theta \boldsymbol{c} + \beta \boldsymbol{c}^{\mathrm{IN}}$ and fed into to the memory buffer for the current context. The completion of that memory update can be detected by the dot product of the updated context and the current context crossing a threshold of one. Such a network is shown in Fig. 8.5.

Unfortunately, this still does not work for non-orthogonal input vectors (Fig. 8.6). In that case the dot product of the updated context and current context are already quite high and the updated context is not completely loaded into the current memory buffer.

8.1.3. Externally controlled alternating memory buffers

All approaches to determine required context updates based on vector similarity will fail because the similarity of $\boldsymbol{c}_i^{\mathrm{IN}}$ and $\boldsymbol{c}_{i-1}$ is not known beforehand and can vary widely depending on what contexts are recalled. Thus, for a

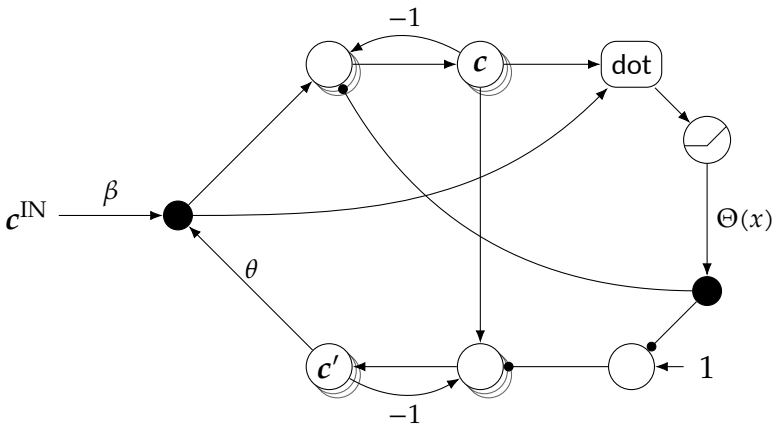


Figure 8.5.: Alternating update of memory buffers.

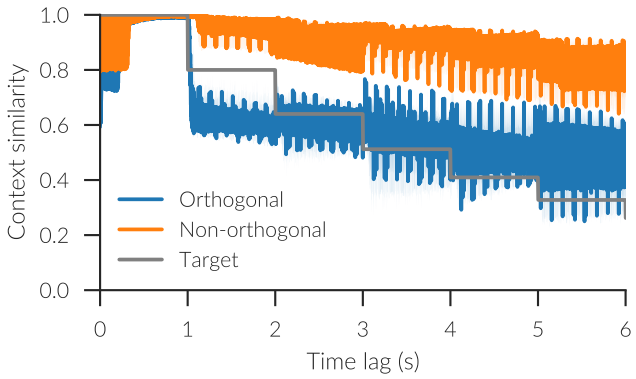


Figure 8.6.: Decay in context similarity with the alternating update of two memories given almost orthogonal inputs and inputs with a similarity of approximately 0.6. The desired similarity is given by the gray line. Shaded regions indicate 95 % confidence intervals.

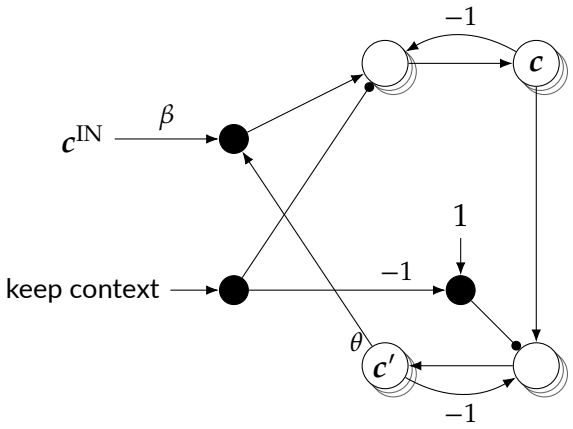


Figure 8.7.: Alternating update of memory buffers with external control.

properly working context update in the TCM model, the update process has to be controlled by an external control signal (see Section 11.1). This control signal indicates when the context signal needs to be updated with the provided input and when it has to be kept stable. If we take the alternating memory buffer network, but control the updating externally (Fig. 8.7), it works for both orthogonal and similar input vectors (Fig. 8.8).

This leads to two predictions. First, the update of the context signal is not directly regulated by the input, but externally controlled. Second, there are neural populations that start representing the current context in succession: first the updated context $\boldsymbol{c}$ is constructed in one memory population before it is transmitted to a secondary population $\boldsymbol{c}'$ to be available as old context for the next update.

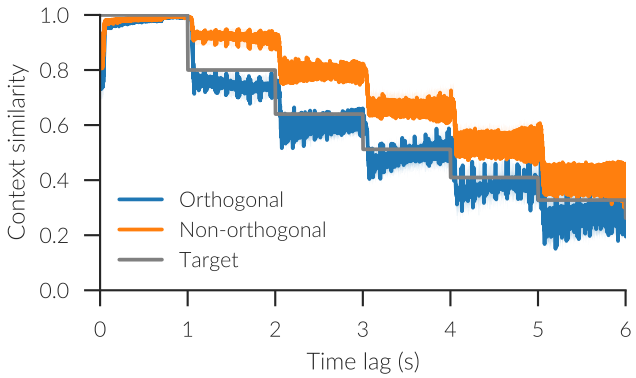


Figure 8.8.: Decay in context similarity with the alternating update of two memories and external control given almost orthogonal inputs and inputs with a similarity of approximately 0.6. The desired similarity is given by the gray line. Shaded regions indicate 95 % confidence intervals.

9. Association Matrix Learning

The TCM requires two association matrices, $\boldsymbol{M}^{\mathrm{CF}}$ and $\boldsymbol{M}^{\mathrm{FC}}$, to be updated. To translate this into neurons, an appropriate learning rule, the association matrix learning rule (AML), has to be derived. The TCM gives the update of such an association matrix as

$$\boldsymbol{M}_{i+1} = \boldsymbol{M}_i + \Delta\boldsymbol{M}_i \tag{9.1}$$
$$\Delta\boldsymbol{M}_i = \boldsymbol{v}_i\boldsymbol{u}_i^\top \tag{9.2}$$

for adding an association from $\boldsymbol{u}_i$ to $\boldsymbol{v}_i$. The association matrix after n updates can be expressed as

$$\boldsymbol{M}_n = \boldsymbol{M}_0 + \sum_{i=1}^{n} \Delta\boldsymbol{M}_i = \boldsymbol{M}_0 + \sum_{i=1}^{n} \boldsymbol{v}_i\boldsymbol{u}_i^\top. \tag{9.3}$$

This allows us to express the neural connection weights after learning n associations as

$$\boldsymbol{W} = \boldsymbol{E}\boldsymbol{M}_n\boldsymbol{D} = \boldsymbol{E}\boldsymbol{M}_0\boldsymbol{D} + \boldsymbol{E}\sum_{i=1}^{n} \boldsymbol{v}_i\boldsymbol{u}_i^\top\boldsymbol{D} \tag{9.4}$$

where $\boldsymbol{E}$ is the post-synaptic encoder matrix and $\boldsymbol{D}$ are the pre-synaptic decoders of the identity function. This equation gives us some important information on how the learning of such association matrices can be implemented. First, preexisting weights can be implemented as a transform on a normal neural connection that is kept constant. Second, all the weight changes can be collapsed into decoder changes. Thus, we need the AML to implement the decoder change given by

$$\tilde{\boldsymbol{D}}_{i+1} = \tilde{\boldsymbol{D}}_i + \Delta\tilde{\boldsymbol{D}}_i \tag{9.5}$$
$$\Delta\tilde{\boldsymbol{D}}_i = \boldsymbol{v}_i\boldsymbol{u}_i^\top\boldsymbol{D} \tag{9.6}$$

where $\tilde{\boldsymbol{D}}$ is the matrix of learned decoders.

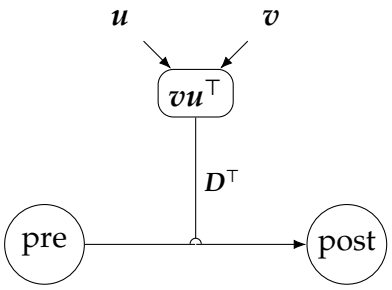


Figure 9.1.: Explicit neural calculation of the AML weight change.

To implement this within a neural network, the discrete equation has to be converted into continuous form:

$$\frac{\mathrm{d}\tilde{\boldsymbol{D}}}{\mathrm{d}t} = \eta \boldsymbol{v}(t)\boldsymbol{u}(t)^\top \boldsymbol{D} \tag{9.7}$$

with learning rate η. Note that while in the discrete formulation all associations are added in with the same strength, in the continuous formulation, the associative strength depends on the learning rate and presentation time. This equation can be directly implemented with the NEF and thus realized with spiking neurons. That alone, however, does not ensure the biological plausibility, as any mathematical formulation of synaptic weight changes could be implemented with the NEF. There are also multiple ways to implement the equation with the NEF that have different implications about a potential biological realization. In the following, several of these possibilities are discussed.

9.1. Explicit calculation of weight change

Both the cue $\boldsymbol{u}(t)$ and target $\boldsymbol{v}(t)$ are available as neural signals. That allows the implementation of the calculation of the outer product $\boldsymbol{v}(t)\boldsymbol{u}(t)^\top$ in a set of neural ensembles. The multiplication of each pair of scalars can be accurately implemented in spiking neurons as demonstrated by Gosmann (2015). Those ensembles can then be connected up to modulate the synaptic strengths from the pre- to the post-populations (see Fig. 9.1) by forming the transpose of the outer product, applying a transform of $\boldsymbol{D}^\top$, and transposing back.

It might be surprising that the change in the connection weights between the pre- and post-ensemble does not depend on their own activity, but is controlled

externally. While this is different from many other common learning rules, there is evidence of such heterosynaptic learning in the brain and specifically the hippocampus (Huilme et al. 2014; Rebola, Carta, and Mulle 2017; Uchida, Fukuda, and Kamiya 2012). Furthermore, this learning rule requires some preexisting structure and connection weights to calculate the signal for the weight modulation. But as most NEF models do not give a developmental account of how such structures come about, I put this question aside and leave it at something that has to be answered in the future and concerns any type of NEF model, not just this learning rule. However, there is one more aspect that can be criticized as being biologically implausible: the connections from the outer product calculation depend on the decoders of the pre-population. It is not clear how the pre-population could transmit this information to this other place.

A similar problem of biological plausibility occurs in classical neural networks and deep learning with back propagation, where the weights for the transmission of the error signal need to be symmetric to the forward weights. Recently, this concern of biological plausibility in deep learning has been slightly alleviated by the discovery that the weight symmetry is not strictly required. It is possible for the network to adjust its forward weights to account for existing, non-symmetric backward weights, a process known as feedback alignment (Lillicrap et al. 2016). Unfortunately, it is not clear whether this can be applied to the situation here.

9.2. Explicit calculation of weight change without weight symmetry

By reformulating $\boldsymbol{v}(t)\boldsymbol{u}(t)^\top \boldsymbol{D}$ as $\boldsymbol{v}(t)[\boldsymbol{D}^\top \boldsymbol{u}(t)]^\top$ it is possible to implement the learning rule without the need for a connection to be based on decoders of a neural ensemble it is not related to. Again there is a set of neural ensembles calculating an outer product (see Fig. 9.2). But in contrast to the previous approach, if the pre-ensemble is also used as the cue input $\boldsymbol{u}(t)$, the transform $\boldsymbol{D}^\top$ applied to $\boldsymbol{u}(t)$ is based on that ensemble's decoders decoding $\boldsymbol{u}(t)$. The transform could even be rolled into the decoder matrix

$$\boldsymbol{D}_* = \boldsymbol{D}^\top \boldsymbol{D}. \tag{9.8}$$

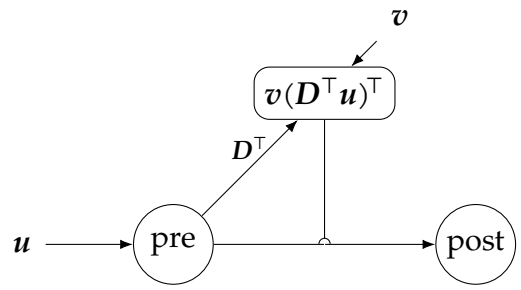


Figure 9.2.: Explicit neural calculation of the AML weight change avoiding weight sharing.

As we generally assume, in the context of the NEF, that there is some mechanism in place to establish the decoders to compute arbitrary given functions, this might already be considered a satisfactory answer to the biological plausibility.

However, it is not possible to state the function to obtain the decoders $\boldsymbol{D}_*$ because the function itself depends on the encoding by the neurons. Moreover, $\boldsymbol{D}_*$ is a symmetric matrix, which is a constraint not respected by the normal decoder optimization process. But in the context of the NEF, it can be assumed that the neural network can decode the identity function, i.e., there is a connection with decoders $\boldsymbol{D}$. This can be used to learn a connection with decoders $\boldsymbol{D}_*$.

Given the vector of neural activities $\boldsymbol{a}$ at time t, we have $\boldsymbol{x} = \boldsymbol{D}\boldsymbol{a}$ and can state

$$\boldsymbol{x}^\top \boldsymbol{x} = \boldsymbol{a}^\top \boldsymbol{D}^\top \boldsymbol{D}\boldsymbol{a} \stackrel{!}{=} \boldsymbol{a}^\top \boldsymbol{D}_* \boldsymbol{a}. \tag{9.9}$$

This gives us an error expression as

$$E^2(\boldsymbol{a}) = \left| \boldsymbol{x}^\top \boldsymbol{x} - \boldsymbol{a}^\top \boldsymbol{D}_* \boldsymbol{a} \right|^2 \stackrel{!}{=} 0 \tag{9.10}$$

with the gradient defined as

$$\frac{\partial E^2(\boldsymbol{a})}{\partial \boldsymbol{D}_*} = 2 \left(\boldsymbol{a}^\top \boldsymbol{D}_* \boldsymbol{a} - \boldsymbol{x}^\top \boldsymbol{x} \right) \left(\boldsymbol{a}\boldsymbol{a}^\top \right). \tag{9.11}$$

The gradient can be used for a weight update rule

$$\frac{\mathrm{d}\boldsymbol{D}_*}{\mathrm{d}t} = -\eta_* \frac{\partial E^2(\boldsymbol{a})}{\partial \boldsymbol{D}_*} \tag{9.12}$$

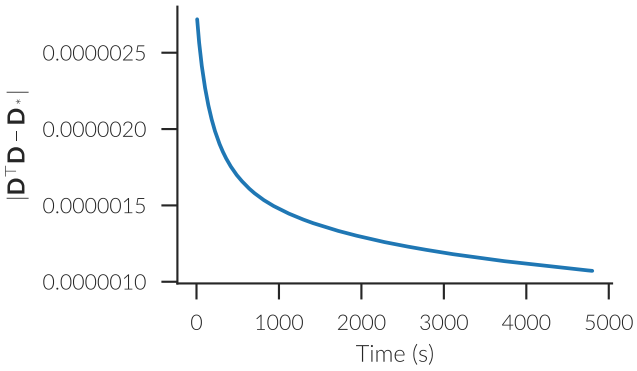


Figure 9.3.: Error $\|\boldsymbol{D}^\top \boldsymbol{D} - \boldsymbol{D}_*\|$.

in a (spiking) neural network to perform stochastic gradient descent.

To demonstrate that this learning rule allows the learning of the desired symmetric matrix, it was applied to a connection where the pre-synaptic ensemble was fed with a randomly varying vector signal. The vector was generated from bandwidth limited Gaussian white noise (upper limit 40 Hz) for each component, normalized, and then multiplied by a bandwidth limited white noise scalar. The learning rate was set to $\eta_* = 1 \times 10^{-13}\,\mathrm{s}^{-1}$. We can see that the Frobenius norm of the difference between the learned matrix $\boldsymbol{D}_*$ and the desired matrix $\boldsymbol{D}^\top \boldsymbol{D}$ continuously decreases (Fig. 9.3). The decoded output for some dimensions quickly aligns with the target output (Fig. 9.4), but that does not happen for all dimensions. This might be due to the random input not sufficiently covering the representational space.

Again, the pure derivation of a learning rule does not ensure its biological plausibility. Thus, let us consider the individual terms in the gradient. The term $\boldsymbol{a}^\top \boldsymbol{D}_* \boldsymbol{a}$ is using the current decoders to decode the ensemble's activity in no way different than usually done in the NEF where the existence of these standard decoders is generally assumed. The decoded value is then correlated with the ensemble's activity. The plausibility of this is somewhat unclear to the direct interaction of decoded values and neural activities, but to my knowledge there is no data excluding this possibility. In particular, the decoded value and activities could be projected to another neural ensembles (see Fig. 9.5) that calculates the inner product. The same, a projection to neural ensemble calculating the inner product, could happen with the decoded value $\boldsymbol{x}$. Alternatively, $\boldsymbol{x}^\top \boldsymbol{x}$ could directly be decoded (as a square of the individual components all projecting

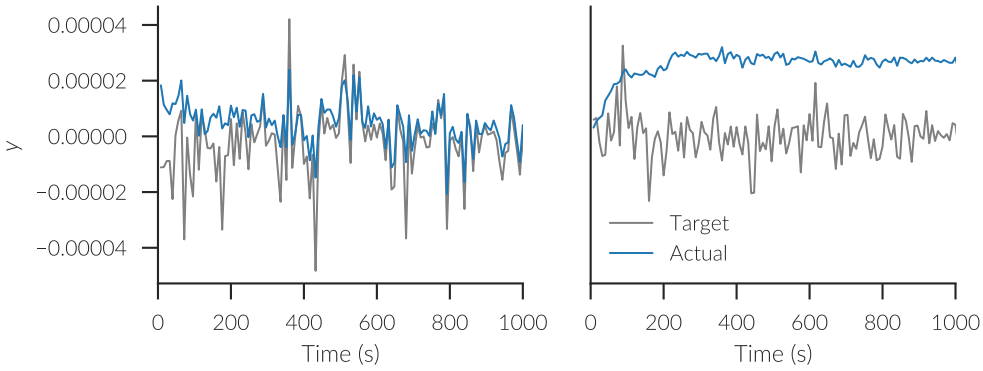


Figure 9.4.: Example of two outputs decoded with the symmetric weights being learned. The output dimensions on the left quickly aligns with the target, while this does not happen (within the shown time frame) for the output dimension on the right.

into the same dimension). Figure 9.6 shows the time course of the error when learning $\boldsymbol{D}_*$ with such a neural gradient computation. The observed decrease demonstrates the basic viability of this approach, but the learning is slower and flattens out earlier due to the spiking noise.

The whole term $\boldsymbol{a}^\top \boldsymbol{D}_* \boldsymbol{a} - \boldsymbol{x}^\top \boldsymbol{x}$ is a scalar that influences all synapses. This could be realized by a broadly acting neuro-modulator or by extensive connectivity of the neural ensembles calculating this error term. Neither option is implausible. This gradient term seems to play a role in weight normalization as it acts equally on all weights and the magnitude depends on the magnitude of

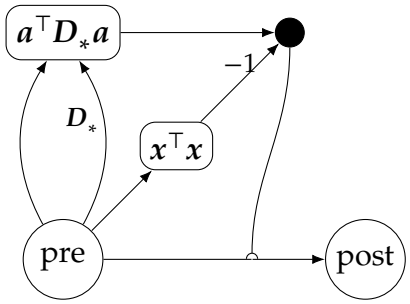


Figure 9.5.: Neural computation of the error signal necessary for learning symmetric decoders with gradient descent.

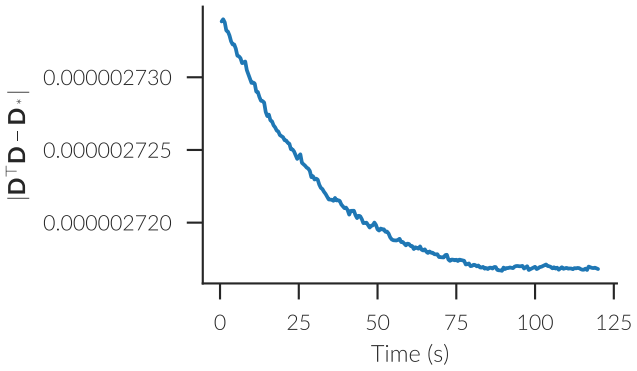


Figure 9.6.: Error $\|\boldsymbol{D}^\top \boldsymbol{D} - \boldsymbol{D}_*\|$ with neural gradient calculation.

the values of $\boldsymbol{D}_*$. Also, without it all weight changes would always be negative. Note that in many learning rules, normalization factors are often criticized as implausible for requiring knowledge of the whole weight matrix at the level of a single synapse. This critique does not apply to this learning rule. The decoding matrix is only used to decode from the activities which require only local knowledge of the weights.

Finally, the term $\boldsymbol{a}\boldsymbol{a}^\top$ correlates neuron activities and appears somewhat Hebbian-like. Hebbian learning is one of the best studied ways that biological systems learn. In this form of learning pre- and post-synaptic neurons become connected when they fire in short succession. In contrast to this usual account, here two pre-synaptic neurons connect to the same post-synaptic neuron if they fire together. Again, the biological plausibility of this is somewhat uncertain, but cannot be outright rejected.

9.3. Implicit error calculation

The previous two implementations of the AML use minimal assumptions about the computational power of synapses. All that is needed are additive weight changes proportional to some error signal. However, it has been proposed that synapses might not simply transmit information, but also perform computations (Abbott and Regehr 2004; Koch 2004, Ch. 5). In particular, Stöckel, Voelker, and Eliasmith (2018) showed that conductance-based synapses can be used in the NEF to compute nonlinear functions like multiplication. That

should allow us to roll the explicit outer product, required in the previous two approaches, into the synapses itself. This requires considerably fewer neurons as no neural populations are required for each product.

In this thesis, I am using an implementation that corresponds to this implicit error calculation, but implements the required synaptic computation in pure math rather than implementing it with actual synaptic models. This is mainly to reduce simulation times and is not meant as a statement that this is likely to correspond to the brain's implementation as there is not enough evidence to make such a strong claim.

9.4. Normative interpretation

While the AML cannot be outright rejected as biologically implausible, there are definitely some open questions concerning it. Nevertheless, it should not be evaluated purely on this fact. Many models in cognitive science, psychology, and computational neuroscience assume the encoding of associations in an outer product matrix (e.g., Kajic et al. 2017; Nowak and Komarova 2001; G. D. A. Brown, Preece, and Hulme 2000). Ultimately, the validity of all those models depends on the possibility that the brain can learn such a matrix. As such, the AML makes it explicit which operations have to be implemented to enable such learning. If these turn out as either not being implemented in the brain or as not being implementable at all, it would follow that the brain has to use some other mechanism to represent associations. Thus, the AML has merit as a normative theory, describing what the brain is ought to do, and directing research to open questions.

That being said, the AML does not make any assumption about the pre-synaptic neurons. With such assumptions, some of the restrictions of the AML might be lessened. For example, assuming orthogonal encoders, the decoder matrix $\boldsymbol{D}$ will be orthogonal too. That in turn means that $\boldsymbol{D}_* = \boldsymbol{D}^\top \boldsymbol{D} = \boldsymbol{I}$ becomes the identity matrix which is independent of the exact decoders. This simplifies connectivity and removes the need to learn the symmetric matrix $\boldsymbol{D}_*$ which alleviates some of the concerns of biological plausibility.

The dentate gyrus in the hippocampus is often assumed to perform pattern separation, which is a form of orthogonalization. Hence, it might be possible that the hippocampus learns associations with a form of the AML where $\boldsymbol{D}_*$ ends up being the identity matrix and thus simplifies the learning rule itself.

9.5. Properties of the AML

So far the considerations about the AML were purely theoretical. Figure 9.7 demonstrates that an implementation of the AML can indeed learn associations in a spiking neural network. Five different cue-target pairs were presented for one second each, before testing the recall with the same cues, but no target vectors. The initially presented target vectors are almost perfectly recalled. Note that no catastrophic forgetting occurred and each association was learned with a single presentation of the cue-target pair (one-shot learning). Most other learning rules exhibit destructive interference between the items in this scenario. As an example, Fig. 9.7 also shows the same experiment using the Prescribed Error Sensitivity (PES) learning rule (Bekolay, Kolbeck, and Eliasmith 2013), which is commonly used in NEF models (e.g., Komer 2015; Rasmussen, Voelker, and Eliasmith 2017). Here all associations except for the last get destroyed. It is still possible to learn associations with PES, but it requires the presentation of each item multiple times in interleaved fashion, i.e., one-shot learning cannot be done to learn associations in a reliable way.

This ability for one-shot learning with the AML is due to the $\boldsymbol{D}_*$ matrix, which accounts for the interference between neurons. In fact, the AML and PES are the same except for that matrix. The PES learning rule can be expressed as

$$\frac{\mathrm{d}\tilde{\boldsymbol{D}}}{\mathrm{d}t} = \eta \boldsymbol{v}(t)\boldsymbol{a}_{\boldsymbol{u}}^{\top}(t), \tag{9.13}$$

while the AML learning rule can be written as

$$\frac{\mathrm{d}\tilde{\boldsymbol{D}}}{\mathrm{d}t} = \eta \boldsymbol{v}(t)\boldsymbol{u}(t)^{\top}\boldsymbol{D} = \eta \boldsymbol{v}(t)\boldsymbol{a}_{\boldsymbol{u}}^{\top}(t)\boldsymbol{D}^{\top}\boldsymbol{D} \tag{9.14}$$

which is the same, except for $\boldsymbol{D}_* = \boldsymbol{D}^{\top}\boldsymbol{D}$.

The one-shot learning ability comes with some limitations, though. The learning rule is restricted to learn transformations that can be expressed as outer product matrices, while learning rules like PES can learn arbitrary transformation matrices. Moreover, the learning rule is essentially trying to learn a look-up table, mapping items to other items. Thus, one should not expect the AML to generalize to unseen mappings.

This might be another reason, besides the stability-plasticity dilemma, why a two-stage memory system is needed. The first stage would learn simple associations with the AML and only in the second step of consolidation with a

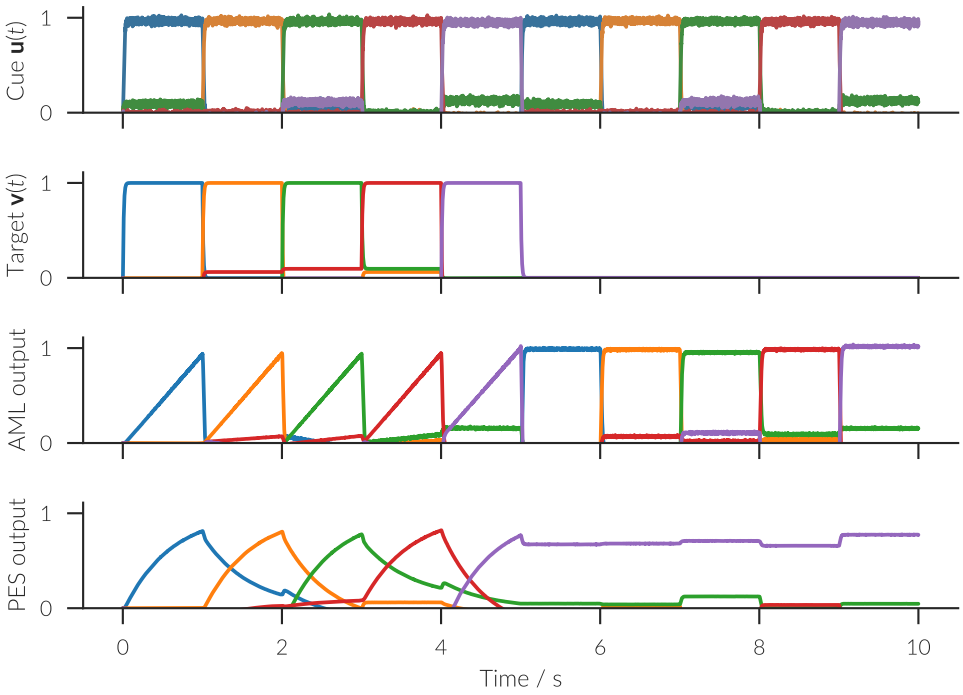


Figure 9.7.: Learning and recall testing of five cue-target pairs with the AML and PES. Each colored trace is the dot product with one of the vectors used. The cue vectors $\boldsymbol{u}(t)$ and target vectors $\boldsymbol{v}(t)$ differ.

different learning rule, commonalities get extracted for generalization. This is consistent with data that shows that generalization improves with sleep. For example, sleep improves transitive inference (Stickgold 2013) and gives insight into implicit rules (Wagner et al. 2004). Experience replay in hippocampus might be responsible for the necessary interleaving of memory traces when using a learning rule that allows for more generalization, but exhibits destructive interference such as PES (McClelland, McNaughton, and O'Reilly 1995; Kumaran, Hassabis, and McClelland 2016). Overall, it is likely that AML only represents a first step in the association learning process and multiple learning rules are at play here. That the brain uses no single general purpose learning rule, but combines different learning rules has been forcefully argued by Gallistel and King (2009).

9.6. AML accounts for neural changes during association learning

Ison, Quian Quiroga, and Fried (2015) reported that individual neurons in hippocampus (and parahippocampal cortex) change their firing rapidly to encode newly learned associations. They recorded from the medial temporal lobe of 14 epilepsy patients that needed to undergo surgery. They identified neurons that responded to specific visual stimuli of pictures of persons and landmarks and recorded the response to this *preferred* (P) stimulus. Then the participants learned associations between pairs of a person and a landmark. Besides multiple tasks to assess the learning, the neuron responses to the different stimuli were recorded again after learning.

Pair-coding units could be identified that showed an elevated firing rate only to the preferred stimulus before learning (BL). After learning (AL), those same units showed an elevated firing rate only to the preferred and associated non-preferred (NP) stimulus. This can also be observed for associations learned with the AML. Figure 9.8 shows a NEF network that implements the learning of associations between persons and landmarks to reproduce the experiment by Ison, Quian Quiroga, and Fried (2015). The connections weights from both pre-ensembles $\boldsymbol{l}$ and $\boldsymbol{p}$ are initialized with the identity transform. To achieve firing rates closer to the recorded data, the maximum firing rates of the ensembles were sampled uniformly from $10\,\mathrm{s}^{-1}$ to $20\,\mathrm{s}^{-1}$ (instead of $200\,\mathrm{s}^{-1}$ to $400\,\mathrm{s}^{-1}$

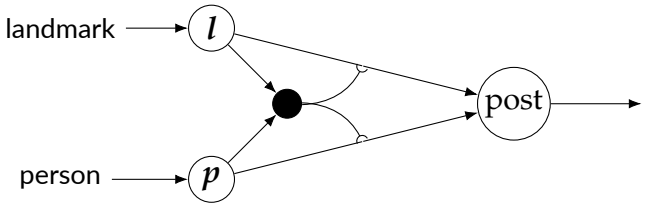


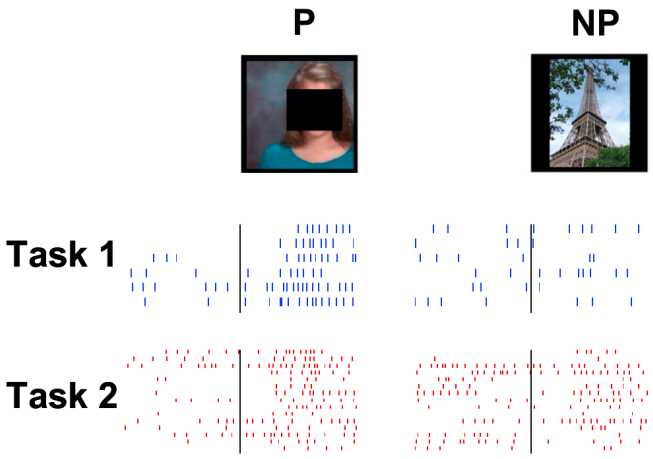
Figure 9.8.: NEF network for associating two stimuli (persons with landmarks here) with the AML.

used in the CUE model) and intercepts were sampled uniformly from 0.1 to 1. The dimensionality of the ensembles and semantic pointers was set to 32. Furthermore, Gaussian white noise with a mean of 0.01 and standard deviation of 0.05, low-pass filtered with a time-constant of 0.1 s, was injected into the neurons to account for neural background firing. The recorded spikes were analyzed analogous to Ison, Quian Quiroga, and Fried (2015).

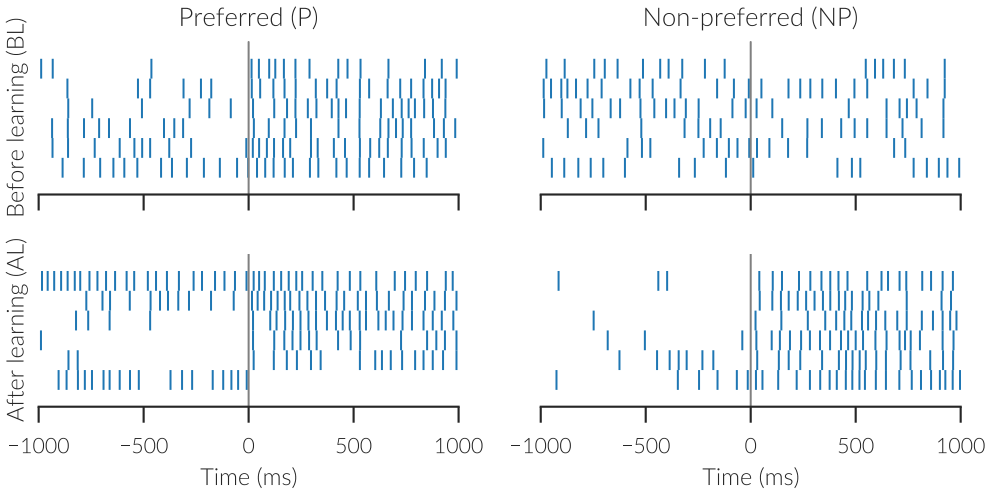
The main results obtained from the experimental and model data are shown in Figs. 9.9 and 9.10. Pairwise coding units, showing elevated firing in response to the preferred stimulus before learning and the non-preferred stimulus after learning without an increased response to any other stimuli, have been selected. The normalized population activity in response to the preferred stimulus shows no or only a minimal change with learning in both the experimental and model data. For the non-preferred stimulus the population activity increases slightly on stimulus onset which is more pronounced in the model data. This might be explained by the model not including any visual processing that could delay and smooth out that response. After learning, this population response is increased in both sets of data. For non-associated stimuli the model shows a minimal decrease in the normalized population activity with learning, while the experimental data shows no significant change. Overall, the model matches the main aspects of the experimental observations. A better quantitative fit might be achievable with more careful parameter selection in the model.

9.7. Weight normalization

Note that the AML allows weights to grow without bound. By introducing a factor of $1 - \boldsymbol{v}(t)^\top \hat{\boldsymbol{v}}(t)$ this can be prevented where $\hat{\boldsymbol{v}}(t)$ is the retrieved, learned association. But similar to other weight normalizations it introduces the need

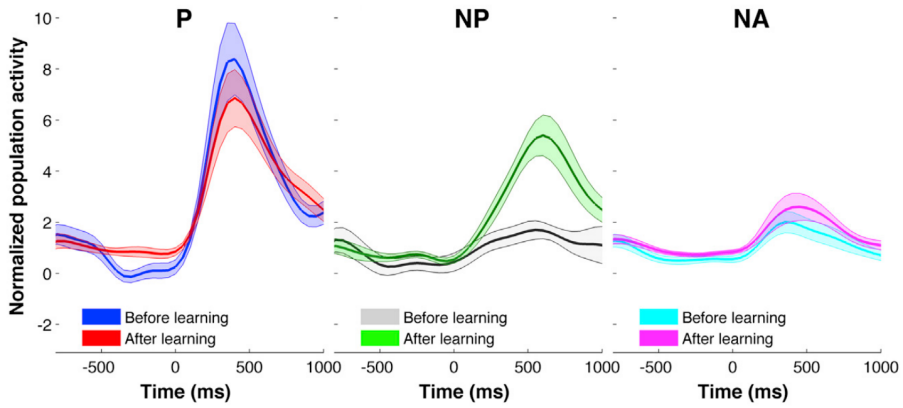


(a) Experimental data

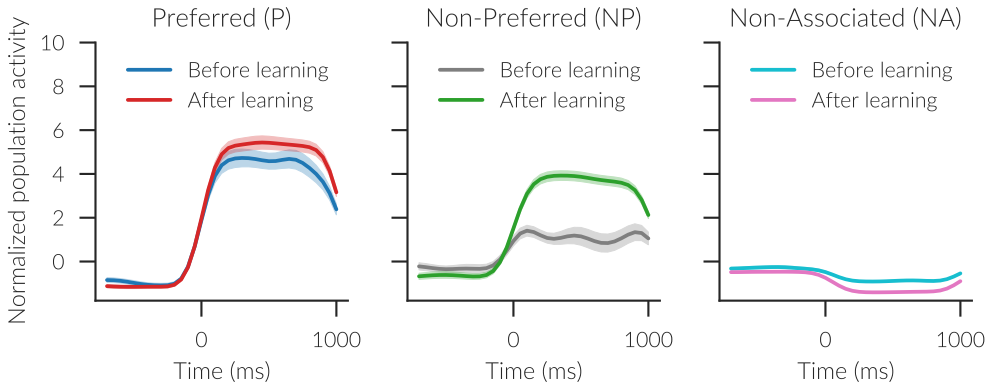


(b) Model data

Figure 9.9.: Change in spiking behaviour when learning associations. (a) Exemplary spikes recorded from human hippocampus for the preferred (P) and non-preferred (NP) stimulus. Task 1 (blue) is before, task 2 (red) after learning the P-NP association. The black line marks the stimulus onset. Figure adopted from Ison, Quian Quiroga, and Fried (2015) under the Creative Commons Attribution 4.0 International license. (b) Spikes recorded from the NEF model learning the P-NP association with the AML.



(a) Experimental data



(b) Model data

Figure 9.10.: Population response for pair-coding units to the preferred (P), non-preferred (NP), and non-associated (NA) stimuli before and after learning. Times are relative to stimulus onset. Shaded regions indicate the standard error of mean. (a) Normalized population activity from (a) experimental and (b) model data. Figure (a) adopted from Ison, Quian Quiroga, and Fried (2015) under the Creative Commons Attribution 4.0 International license.

for each weight to have access to the global population activity and weights as $\hat{\boldsymbol{v}}(t) = \tilde{\boldsymbol{D}}\boldsymbol{a}_{\boldsymbol{u}}(t)$. Such global dependencies are often criticized for not being biologically plausible. As such, I decided to take a slightly different approach with an equivalent effect. Instead of including the dot product $\boldsymbol{v}(t)^{\top}\hat{\boldsymbol{v}}(t)$ in the learning rule, it can be computed by another neural population and the thresholded result can be used to inhibit the population providing $\boldsymbol{v}$. Once fully inhibited, $\boldsymbol{a}_{\boldsymbol{v}}(t)$ will be all-zero and thus prevent further weight changes. In the CUE model, the inhibition threshold is adjusted to follow $1 + \exp(-t)$ for learning the $\boldsymbol{M}^{\mathrm{CF}}$ matrix where t is the time since the trial started. This is intended to account for rehearsal effects that are not explicitly modelled and is analogous to the extended TCM model by Sederberg, Howard, and Kahana (2008).

10. Recalling items

In the original TCM model (Howard and Kahana 2002) the activations a_i of items in the memory is mapped to a recall probability by a softmax function

$$P(\boldsymbol{f}_i|\boldsymbol{c}) = \frac{\exp\left(2a_i/\tau\right)}{\sum_j \exp\left(2a_j/\tau\right)} \tag{10.1}$$

with a free parameter τ controlling for the sensitivity. While this does well in capturing the recall probabilities, it does not provide much insight in how this recall process might be realized neurally. In an extension of the TCM model (Sederberg, Howard, and Kahana 2008) a winner-take-all (WTA) process based on the widely-used leaky, competing accumulator (LCA) model by Usher and McClelland (2001) was used. This works well if the output can be evaluated in a mathematical analysis. However, within the CUE model other parts of the model need to recognize when a single recall is completed to update the context and reset the recall system. As I show, this is difficult to do robustly with the LCA model, but more easily accomplished with an alternate WTA mechanism termed the independent accumulator (IA) model. A comparison of these two networks has also been previously published as Gosmann, Voelker, and Eliasmith (2017).

10.1. Leaky, competing accumulator model

Given D choices, the leaky, competing accumulator (LCA) model proposed by Usher and McClelland (2001) describes the dynamics of D scalar state variables $x_i(t), 1 \leq i \leq D$ as

$$\frac{\mathrm{d}x_i}{\mathrm{d}t} = \frac{1}{\tau}\left(u_i(t) - \kappa x_i - \lambda \sum_{j \neq i} x_j\right), \quad x_i \geq 0 \tag{10.2}$$

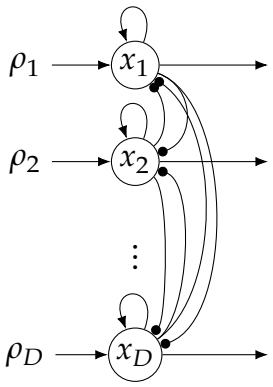


Figure 10.1.: Leaky, competing accumulator (LCA) network.

where $u_i(t)$ are the external inputs, κ is the leak rate, λ the lateral inhibition, and τ the integration time-constant. Each state variable x_i is kept non-negative by setting negative values to zero. Intuitively, each state variable integrates its input with a leak term of $-\kappa x_i$ and provides lateral inhibition to all other state variables. Given one input $u_i > u_j$ for all $j \neq i$, the state variable x_i will converge to u_i while all other state variables x_j, $j \neq i$ will converge to 0 if $\kappa = \lambda = 1$ (Appendix C). In the following analysis, $\kappa = \lambda = 1$ will be fixed. Other choices of λ affect the effective integration time-constant τ and gain on the input, while changing κ results in undesired behaviour where the history of inputs influences the current choice.

By means of principle 3 of the NEF, the prescribed dynamics can be exactly implemented in the NEF. Here, one ensemble per state variable is used (Fig. 10.1) and the gains and biases of the neurons are distributed as described in Section 4.4 to ensure the rectification of the state variables.

10.2. Independent accumulator model

The dynamics of the independent accumulator (IA) model are given by

$$\frac{\mathrm{d}x_i}{\mathrm{d}t} = \frac{1}{\tau_1} u_i(t) + \frac{1}{\tau_2}\left(\bar{x}_i - \bar{\lambda}\sum_{j\neq i}\bar{x}_j\right), \quad x_i \geq 0 \tag{10.3}$$

$$\bar{x}_i = \Theta(x_i - \vartheta) \tag{10.4}$$

where $u_i(t)$ again gives the external input, τ_1 and τ_2 are feedforward and feedback time-constants, $\bar{\lambda}$ is an inhibition constant, Θ is the Heaviside step

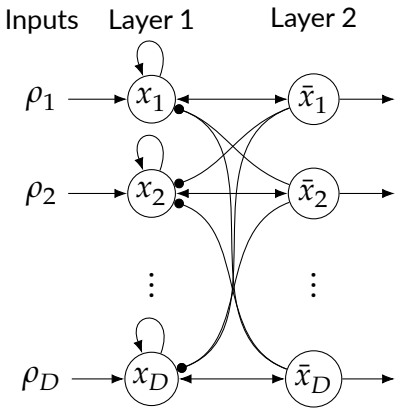


Figure 10.2.: The independent accumulator (IA) network. The second layer is used to compute the function $\bar{x}_i = \Theta(x_i - \vartheta)$.

function, and ϑ is a threshold. The Heaviside non-linearity is the crucial difference to the LCA model as we can reduce this equation to (10.2) by setting $\tau = \tau_1$, $k = -\tau_1/\tau_2$, and $\lambda = \bar{\lambda}\tau_1/\tau_2$ when not considering the Heaviside function. Through the Heaviside function, the accumulators act as perfect (opposed to leaky) and independent integrators. Only when the threshold ϑ is reached, the winning choice is stabilized by feedback while all non-winning choices are inhibited.

These dynamics can again be neurally implemented by means of the NEF (Fig. 10.2). While for the LCA model a single layer was sufficient, it is best to use two layers to implement the IA model. The first layer consists of independent accumulators representing the state variables x_i. This layer projects to the second layer that performs the computation of $\bar{x}_i$ (i.e. the Heaviside function). The second layer projects back to first layer and can be used to read out the output of the network.

10.3. Comparisons of the WTA networks

The pure analytical description does not tell us which network is better suited for certain tasks. To compare these networks, I simulate them with an input of $u_i = u - s\,(1 - \delta_{1i}) + \eta_i$, where u is the magnitude of the largest input, $s > 0$ is the target separation, δ is the Kronecker delta, and η_i is Gaussian white noise with a standard deviation of σ. The first input is always set to be the target and all other inputs are set to be lower by s (without loss of generality as we can reorder the indices). The rationale behind this is that it should present the

hardest case because every choice is close to the largest input. As $s \to 0$, the problem gets more difficult as the separation to the target shrinks. Note, that u determines the general baseline of inputs in addition to the value of the largest input. Furthermore, it is important to consider the influence of noise η_i on the robustness of the decision process.

As the input lists to the CUE model are about ten to twelve items, I compare the networks for $D = 10$ choices. To make a fair comparison 200 LIF neurons are used per choice in either model. In the IA model this means, that the number of neurons is split between the first and second layer. Here I use 150 neurons in the first, and 50 neurons in the second layer. As the first layer is performing the evidence accumulation it needs to be more accurate and requires more neural resources. Due to the lower number of neurons in the output layer, the decoded output has a higher variance. This, however, is not as relevant as variation of index of the highest output. If all other choices do not produce an output, the winning choice can still be clearly identified despite the variance in the decoded output.

Given this basic setup of the comparison, a number of metrics give insight in the performance of the networks. First, we want the network to reach a *clear decision* which I define as exactly one output being above a threshold of 0.15 during the time interval from 1 s to 2 s simulation time, while all other outputs remain below the threshold. The threshold of 0.15 was chosen as it is usually sufficient to tell a zero and non-zero signal apart despite spiking noise (except for very low neuron numbers or firing rates). This metric requires that the output does not change during the interval as this might cause problems in downstream networks that operate on the output. Also, with a change in output it would be unclear which output should be taken as the actual decision. One thing this metric does not take into account is whether the output corresponds to the highest input. This, whether a decision is *correct*, is the second metric. But note, that in some situations within a larger scale network a wrong, but clear decision, can be preferable to a decision that tends to be correct, but is unstable.

Two more metrics are used for all trials that reached a clear decision. The amount of time taken to fulfill the conditions for a clear decision are considered as the *decision time*. It measures how fast the network is obtaining the decision. Finally, the networks can produce transient outputs unrelated to the final decision. A downstream network might consider this output as an actual decision and thus those transient outputs should be as small as possible. The

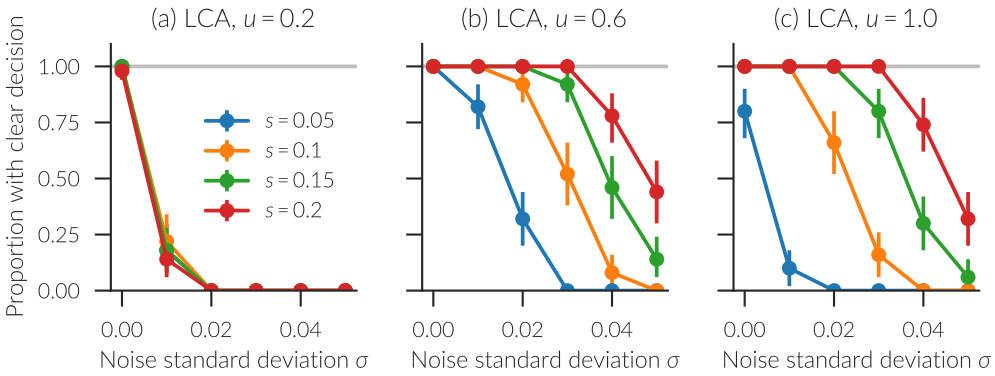


Figure 10.3.: Proportion of trials with a clear decision for the LCA network. The data for correct trials is exactly the same. Each plot shows a different input magnitude u and each curve is for a different target separation s. The optimum is marked with the gray horizontal line and coincides with the performance of the IA network for clear decisions. Error bars show 95 % confidence intervals.

highest output of a losing choice during the whole simulation is taken as a metric for this.

10.3.1. Results

Figure 10.3 shows the proportion of trials that the LCA network reaches a clear decision for different input parameters. The input magnitude u must be large enough to reliably exceed the 0.15 detection threshold under noise. For $u = 0.2$ and even low amounts of noise the network fails to reach a clear decision. However, it seems that a too large input magnitude decreases performance as well when we compare the performance for $u = 0.6$ and $u = 1$. Increasing the noise standard deviation σ or decreasing the target separation s, both decrease the performance as this makes the problem harder. Interestingly, the IA network does not fail to reach a clear decision in any of these conditions.

Moving on to correct decisions, the LCA network always produced the correct output given it produced a clear decision. The IA network, may produce incorrect outputs despite a clear decision (Fig. 10.4). Again, as the problem gets more difficult either by decreased target separation or increased noise, the network performance on this metric decreases. Note that the IA feedforward

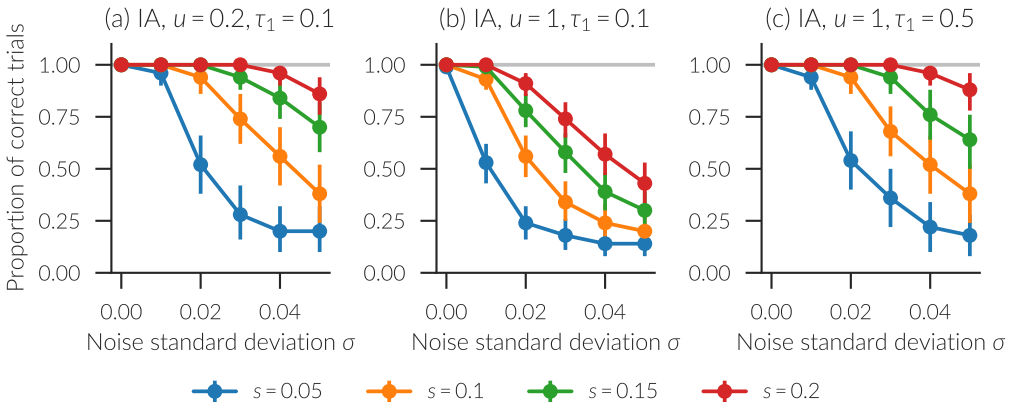


Figure 10.4.: Proportion of correct trials for the IA network. Each plot shows a different combination of input magnitude u and integration time constant τ_1. Each curve shows a different target separation s. The optimum is marked with the gray horizontal line. Error bars show 95 % confidence intervals.

integration time-constant can be increased to integrate evidence over a prolonged time period to increase performance on this metric (compare Fig. 10.4b and c).

This, however, increases the decision times. These tend to be already slower for the IA network than for the LCA network (Fig. 10.5). Additional noise can shorten the decision times in the IA network as it increases the likelihood of an integrator randomly accumulating enough evidence to cross the threshold. For the LCA network the noise level only has a minor influence on the decision time and was averaged over in the plot. In other words, noise in the LCA network influences whether a decision can be reached, but not how long it takes to reach a decision if one is reached.

Finally, both networks might produce a transient response that gets worse with increased noise (Fig. 10.6). This is inherent in the LCA network because the state variables are directly used as output. In the IA network, this transient response is caused when two accumulators cross the threshold in close temporal proximity before the inhibition from the first one can silence the remaining accumulators. By increasing the feedforward time-constant, this transient response gets reduced as the evidence integration gets slowed down, which temporally stretches out when accumulators cross the threshold.

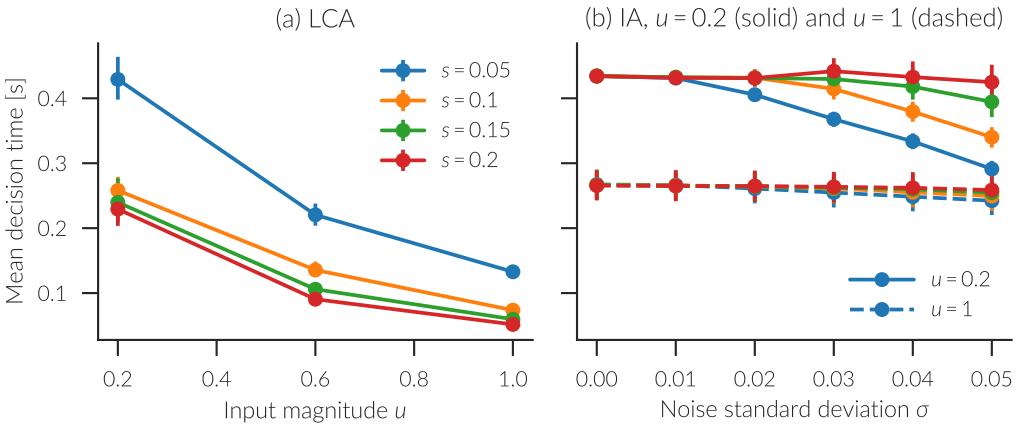


Figure 10.5.: Mean decision times for the (a) LCA and (b) IA network. Data for the LCA network is shown as depending on the input magnitude u and is averaged over all noise levels σ because noise had a minimal effect on decision times. Data from the IA network is shown as depending on the noise standard deviation σ for two input magnitudes of $u = 1$ (dashed lines) and $u = 0.2$ (solid lines). Target separation is indicated by color. Error bars show 95 % confidence intervals.

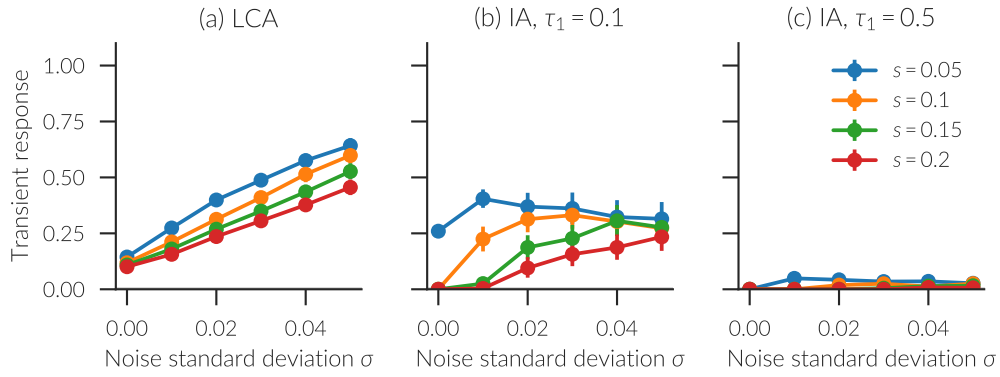


Figure 10.6.: Transient responses (i.e. highest output of a non-winning choice) for the WTA networks as depending on the noise standard deviation σ. Data from the IA network is shown for two integration constants. Each curves shows a different target separation s. Error bars show 95 % confidence intervals.

10.3.2. Discussion

Neither network performs better on all metrics. Thus, each is best suited for a different purpose. In situations where a continuous adjustment of a decision is necessary, the LCA network is most likely the better choice. Under noisy conditions it is not necessarily able to produce a stable, clear output, but if a clear output is obtained, it is always correct. If the input changes, the output adjusts according to the network's time constant and allows for continuous updates. Unfortunately, this also makes the network susceptible to noise.

The IA network is better for a discrete series of decisions. It does not produce a continuous output, but waits until the evidence integration threshold is crossed, at which point the network needs to be inhibited to be reset and obtain another decision. While this produces sequential and discrete decisions, it has the advantage that evidence can be accumulated over a longer time frame to average out noise. Depending on the choice of the integration time-constant, the IA network may either be not as quick or as reliable in identifying the correct winner as the LCA network. However, it will eventually produce a clear decision that a downstream network can act on (as long as at least one input is strictly positive). This can be important in a larger scale model where stalling a decision indefinitely can result in a breakdown of model behaviour. Or in other words, sometimes it is better to act on a wrong decision than to not act at all. For example, in memory recall it might be better to produce a wrong output and continue to recall the next item than indefinitely be trying to recall an item that cannot be recalled. It is also worth pointing out that the IA network allows for dynamic control of the decision speed by adjusting the ϑ threshold through a bias input to the second layer.

As mentioned, it is also important to consider the transient outputs of the network within a larger scale model. Such transient responses are inherent in the LCA model as the state variable is directly used as an output. One could pass the output through a thresholding layer, but the right choice of threshold is not clear, as the output magnitude depends on the input magnitude. If the threshold is too low, a transient response would be produced even with the thresholding layer. If it is too high, small inputs might not produce an output at all. While the IA can also produce transient responses, these can be reduced to almost zero by a proper selection of the integration time-constant according to the input magnitude and target separation.

By increasing $\tau_1 \to \infty$, the IA discrimination ability can be increased without

bound. This is sometimes used as argument to criticize these sort of models (e.g., Usher and McClelland 2001), because there is no sensible stopping criterion. However, this does not consider that there might be a cost to taking more time for a decision. If that cost is included, then there is a trade-off between accurate decisions and the cost incurred by taking more time to decide. Furthermore, this argument also assumes perfect integration accuracy, which an actual neural system cannot have due to neural noise and limited neural resources.

To summarize these findings, for the recall of items in memory experiments the IA network is better suited. Such recall requires discrete decisions and a stable input to downstream motor systems producing the response. Such a stable input cannot be guaranteed with the LCA network and it proves a challenge to detect when a recall is completed. Sederberg, Howard, and Kahana (2008) used the LCA network for modelling the recall, but it is important to highlight that upon reaching the decision threshold, the state variables were immediately set back to zero. This is easy to do in a pure math model, but when transitioning to a full neural model detecting the decision and resetting the state variables is much more difficult, as it cannot be done instantaneously.

Finally, let us consider biological plausibility briefly. Both networks where simulated in spiking neurons, which ensures a certain degree of biological plausibility. Also, accumulation of evidence to a threshold is a well known finding for neurons in LIP (Gold and Shadlen 2007; P. L. Smith and Ratcliff 2004). Often this is assumed to be a gradual integration, but when looking at individual trials instead of the trial-average, a distinct step response becomes evident (Latimer et al. 2015). This matches the output of the second layer of the IA network. However, both networks have an integration layer with gradually increasing firing rates, which implies that such neurons should exist too. Ultimately, it is possible, that the brain employs both networks for different tasks as they have different strengths and weaknesses.

10.4. Recall network

The recall network in the CUE model is based on the independent accumulator network. Each potentially recallable item is regarded as one choice. An additional *null choice* is added to represent a failed recall. This is a stand-in for additional items that might be present in the recall network, but have not

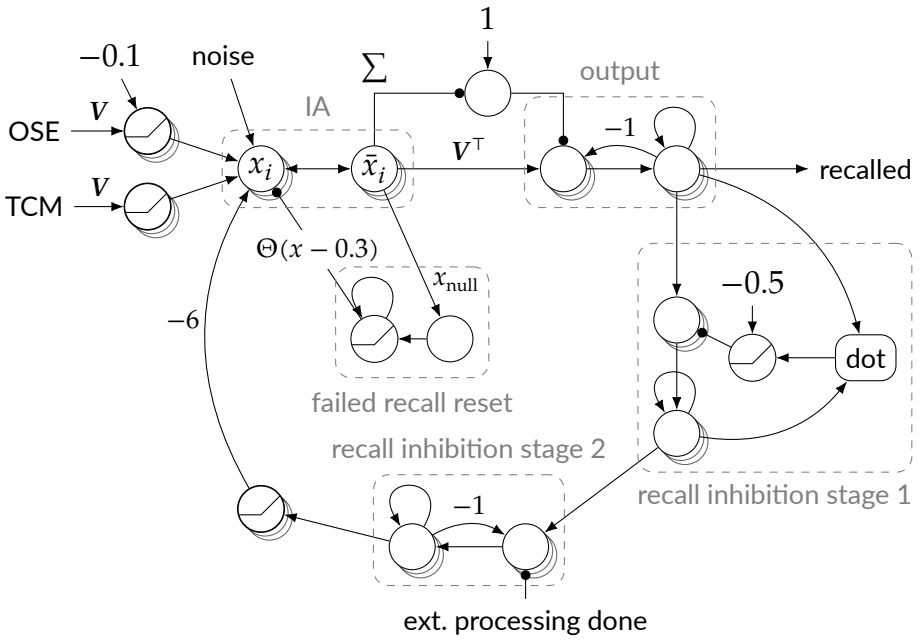


Figure 10.7.: Recall network. See text for details.

occurred in the learned list. It is also a way of providing a time-limit on trying to recall a particular item, and prevents pure noise resulting in a successful recall of one of the learned list items. The additional choice is fed a constant signal of μ. Additive Gaussian white noise is applied to each input with zero mean and a standard deviation of σ to account for additional processes that might interfere with the recall process.

Furthermore, the IA network is embedded into further components (Fig. 10.7). First, items are represented as Semantic Pointers, but the IA network needs a separate utility value for each potential choice. Thus, the incoming connection uses the matrix of the Semantic Pointers of all possible list items as a transform, effectively calculating a dot product between the input signal and each potential item. These utility values are then rectified to only consider positive evidence. By subtracting 0.1 from the input utilities provided by the OSE output, integration of pure noise from a failed short-term memory recall is prevented.

The IA network does not produce an output during the evidence accumulation phase. It is, however, helpful to have a persistent output of the last recalled

item (note that this is still different from the output of the LCA network). This allows for a simpler analysis and can provide potential future downstream networks with a longer lasting input. To achieve this, the output of the IA network is routed through a gated memory that is only updated when the IA network produces an output. As the gated memory stores a Semantic Pointer, a transform matrix needs to be applied to the IA output to project the choice back into the Semantic Pointer space. The updating is controlled by inhibiting the memory gate by default and disinhibiting it (by inhbiting the inhibitory population) when the IA network produces an output.

Repetition errors are rarely made in recall experiments. Thus, it is necessary to inhibit already recalled items. However, this inhibition should not happen immediately as otherwise the recall output would be inhibited too quickly for the downstream network to act upon. Thus, a two-stage process is used. First, an initial working memory population gets immediately updated by feeding in the currently recalled item. This adds the recalled Semantic Pointer into the representation of items to inhibit. From that representation and the current recalled item, a dot product is calculated and thresholded at 0.5. Once the threshold is crossed, the gate to the memory population is inhibited as the new item should be added into the representation, but not completely overwrite it. The output of that memory population is fed to a gated memory that provides the inhibition to recall in the IA. The gate for this memory is controlled externally and disinhibited once the downstream processing has completed. Note that the output of that memory population is projected back into utility values and rectified. This prevents that a negative cosine similarity with one of the vectors stored in the memory population contributes positive evidence.

Finally, the recall network needs to be reset if a recall fails to allow it to continue trying to recall the item for the next position in serial recall. Here the problem is that if the IA output for a failed recall is directly used to inhibit the IA network to reset it, this also immediately inhibits the output used for the inhibition. This does not reliably reset the network as the reset signal is disabling itself. Thus, the signal is fed to an integrator until a threshold is reached and then the signal of the integrator is used to provide an inhibitory pulse to the IA network. The slower decay of the integrator ensures a sufficient pulse length to restart the recall process.

11. The complete model

Now we have all the essential components to construct the complete context-unified encoding (CUE) model.[1] Figure 11.1 gives an overview of the information flow between the different components, and Table 11.1 states the neuron count for different model components. The Semantic Pointers v for the presented items are the input to the model and the recalled items $\hat{v}$ of the item recall network are the model output. The part of the model corresponding to the TCM consists mainly of the $\boldsymbol{M}^{\mathrm{FC}}$, $\boldsymbol{M}^{\mathrm{CF}}$, and context networks, whereas OSE and position correspond to the OSE. The position network (Section 7.2) stores a Semantic Pointer $\boldsymbol{p}$ indicating the current list position. The position is advanced by a control signal discussed in the next section. Both the current list item and position are input to the $\boldsymbol{M}^{\mathrm{FC}}$ and OSE networks.

Within the OSE network, the list item and position Semantic Pointers are bound together and added into the representation of the current list in a neural integrator. The inputted position is also used to unbind an item from the list representation and feed it to the item recall network.

In the $\boldsymbol{M}^{\mathrm{FC}}$ network the superposition of the input item and position (instead of the binding) is created. This superposition is used to recall the context previously associated with the item and position and to update the current context in the context network accordingly. Furthermore, the current context is fed back to $\boldsymbol{M}^{\mathrm{FC}}$ as a modulatory signal to learn the association from the current item and position input to the current context. Via the $\boldsymbol{M}^{\mathrm{CF}}$ network, the context recalls the associated Semantic Pointer and transmits it to the item recall and position recall networks. The current item and position are a modulatory input to the $\boldsymbol{M}^{\mathrm{CF}}$ network to create the association from the current context to these Semantic Pointers.

During the recall phase, recalled items and positions are routed back to the $\boldsymbol{M}^{\mathrm{FC}}$ network to recall further items. The recalled position sets the current

[1] The complete model and evaluation source code is available from https://github.com/ctn-archive/cue-model.

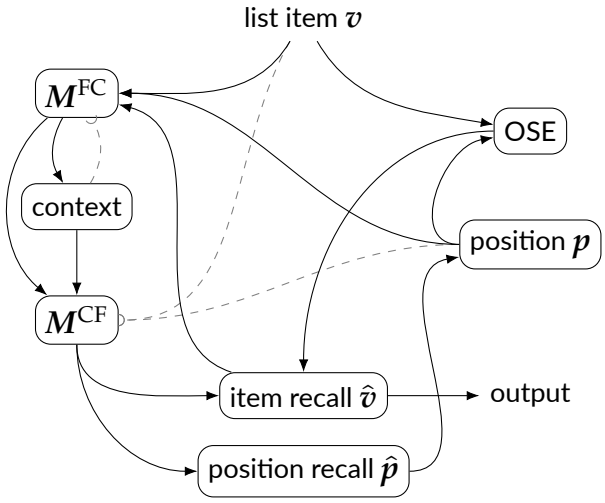


Figure 11.1.: General information flow in the CUE model. The routing of information depending on the task, and task phase is not shown in this figure. Thus, not all shown connections are active at all times.

Table 11.1.: Neuron count in the cue model with up to 20 positions (each additional position adds 120 neurons).

Component	Neuron count
LTM (TCM)	205 125
thereof for context	51 225
STM (OSE)	360 000
Position network	2860
Recall networks	189 350
thereof for items	94 875
thereof for positions	94 475
Other	129 170
Total	886 505

position in the position network to recall that position's item from the OSE short-term memory. The recall networks also store recently recalled items in neural integrators to prevent repetition errors that happen rarely in human experiments.

11.1. Control

While the general structure of the model is important, the desired model behaviour can only be achieved by controlling the flow of information appropriately. This control happens on multiple levels. On the highest levels, the effective connectivity is modified by the general task performed (e.g., an immediate versus a serial recall task) and the task phase (e.g., presentation versus recall phase). Below that, certain information routing is done for each item until it has been stored in memory or for each recall. On the lowest levels, some control and routing happens within the individual networks of the CUE model as described in the corresponding sections. For example, the $\boldsymbol{M}^{\mathrm{FC}}$ and $\boldsymbol{M}^{\mathrm{CF}}$ networks inhibit the modulatory signal once a certain association strength has been reached.

Figure 11.2 shows the information flow during the presentation phase. Parts of the recall networks and the input to them are inhibited. During the recall phase, the routing of information depends in part on the type of recall task as shown in Fig. 11.3. For serial recall, the transmission of the recall network outputs is inhibited because the recalled item is not supposed to be a cue for recalling the next item. Instead, the output of the position network is used as a cue. During free recall, instead, the output of $\boldsymbol{M}^{\mathrm{FC}}$ is used to update the context as usual and the updated context acts as input to the $\boldsymbol{M}^{\mathrm{CF}}$ network.

The main control problem for each item is to regulate the context update because (as discussed in Section 8.1) this cannot be done based on the context signal alone, but requires an external control signal. Before the context can be updated, the new input signal $\boldsymbol{c}^{\mathrm{IN}}$ needs to be present at the network input, which requires a delay after a new list item has been presented. Accordingly, as soon as a new item is presented, the context update should be stopped until that input signal is propagated and the current context has been propagated to the buffer for the old context in the context network. To achieve this delay, the Semantic Pointer of the new item is fed into an integrator with a slow synaptic time constant of $\tau = 0.1\,\mathrm{s}$ (Fig. 11.4). Between the input Semantic Pointer and

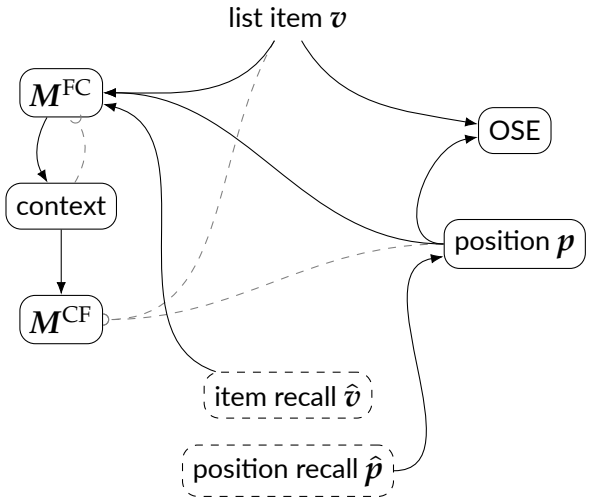
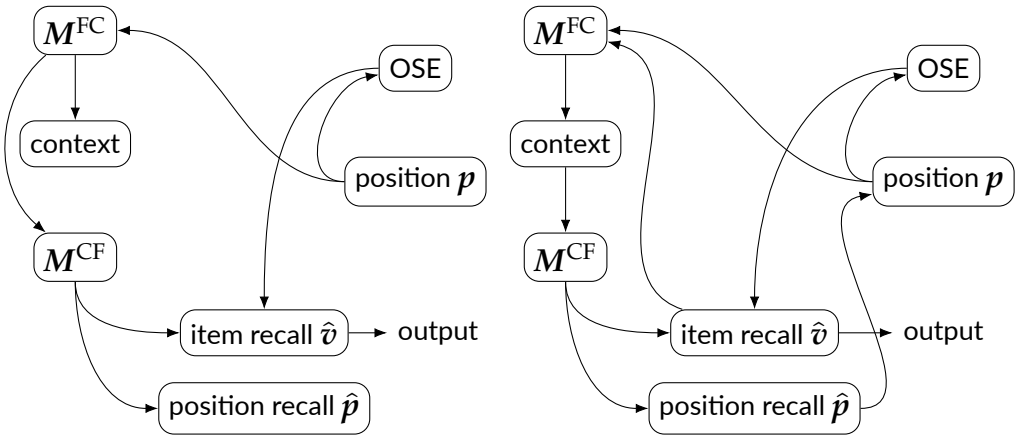


Figure 11.2.: Information flow during the presentation phase.



(a) Serial recall (b) Free recall

Figure 11.3.: Information flow during the recall phase.

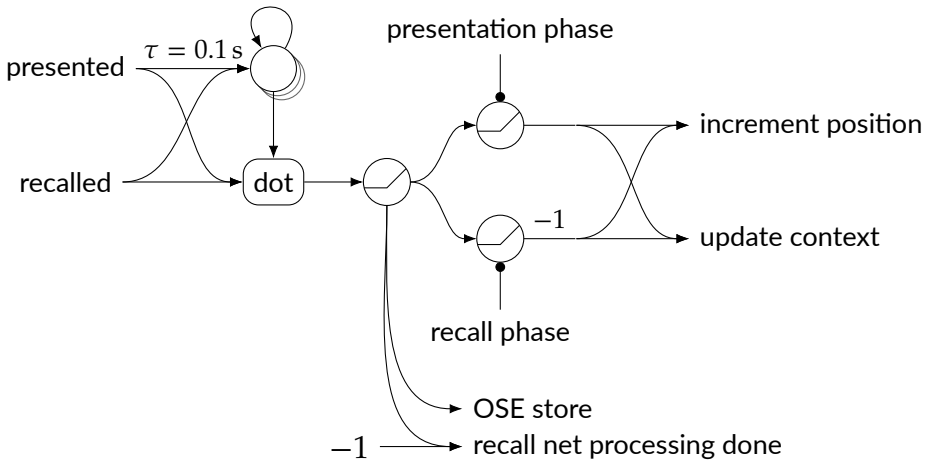


Figure 11.4.: Generation of control signals from the currently presented or recalled item.

the output of the integrator a dot product is calculated that slowly increases towards one. The threshold, obtained with a thresholding ensemble, gives the required control signal for the presentation phase. During the recall phase, the logic is inverted. That enables an immediate update of the current context based on the position provided by the position network and last recalled item. Once an item has been recalled, it is used like a newly presented item and fed to the integrator and dot product. This provides the signal to transfer the current context to the secondary buffer after a delay. The thresholded dot product signal is also used to control three other aspects of the model:

- It is required to enable the learning of associations in the $\boldsymbol{M}^{\mathrm{FC}}$ and $\boldsymbol{M}^{\mathrm{CF}}$ matrices to prevent the creation of associations before the context has been updated.
- It is used to provide the control signal to transfer the updated OSE memory to the secondary memory buffer to allow for the next update.
- The inverted signal is used to gate the transmission of the recalled item in the recall network to the memory of recalled items preventing repetition errors.

Special control and information routing is also necessary during the distrac-

tor tasks or when no list item stimulus is present. Because the distractors are irrelevant to the task, they are assumed to be not encoded in the hippocampal long-term memory. Thus, during the distractor phases the error signals for the association learning are inhibited to prevent the distractors from being learned. Moreover, the distractors are not part of the learned list and thus the advancement of the position counter is inhibited. Instead of the position network output, a different Semantic Pointer indicating an irrelevant position is routed to networks otherwise receiving the position Semantic Pointer.

Switching of task phases also requires some reconfiguration of the network state. The start of the recall phase is detected with thresholded differentiators for serial and free recall (one of them is inhibited). For serial recall the position network is reset to the first position by exciting the neural ensemble for the first position and inhibiting all others. At the same time the Semantic Pointer for the first position is fed to the $\boldsymbol{M}^{\mathrm{FC}}$ network to start of the recall while the position network is still transitioning to representing the first position. Because some subjects may use a serial recall strategy even in free recall, models are configured with probability $\psi = 0.1$ to perform this serial recall routing even in free recall (except in the delayed recall condition). In free recall, the position network is inhibited at the start of recall to base the first recall purely on the currently active context signal. If later during the free recall process a position vector is recalled, it may still set the position network to represent that position.

Finally, the recall networks might fail to recall an item (or position) if the input evidence is too low or too noisy. While these networks will restart the recall process internally, some global actions are necessary for the failed recall of items. The context network is provided with a signal to update the current context to then use the updated context for the next recall attempt. Also, for serial recall, the position network is provided with a signal to increment the current position to attempt recalling the next serial position in the list.

11.2. Extension to the Hebb repetition effect

The CUE model as presented so far is unable to reproduce the Hebb repetition effect. Two different extensions, but both involving an additional type of learned, weight-based associative memory, make it possible to reproduce the effect. First, it is possible to learn the direct association between Semantic Pointers for positions and the presented items with the AML. The associations

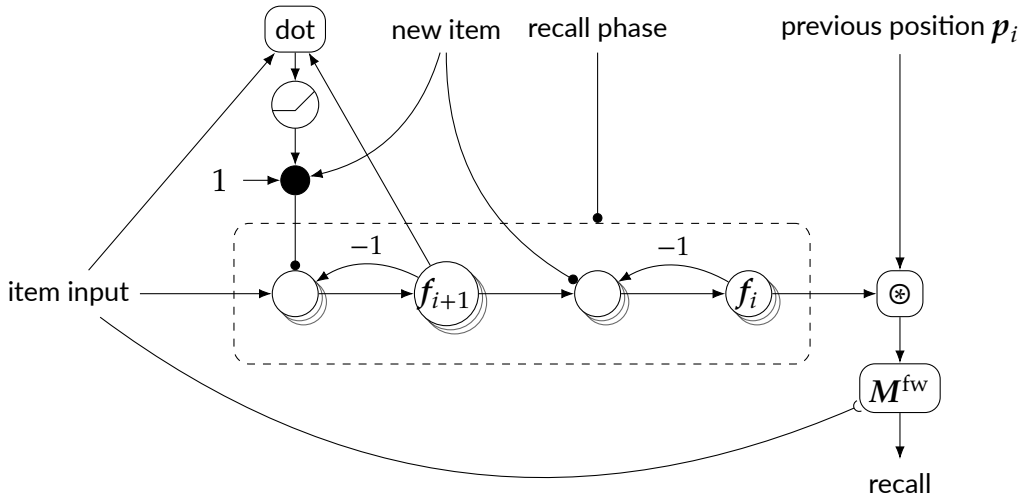


Figure 11.5.: Extending the CUE model with forward associations.

for the repeated list become stronger than the associations for the other lists over time. This will bias the recall towards items in that repeated list.

A second alternative, is to learn the association from $\boldsymbol{f}_i \circledast \boldsymbol{p}_i$ to $\boldsymbol{f}_{i+1}$, where $\boldsymbol{f}_i$ and $\boldsymbol{p}_i$ are the i-th list item and position vector (Fig. 11.5). This is essentially learning of foward associations. The binding to the position vector disambiguates between lists as $\boldsymbol{f}_i \circledast \boldsymbol{p}_i$ is only the same vector if the same item is presented in the same position. In the recall phase, a previously recalled item and its serial position can be used as a cue to support the recall of the next item with this additional learned association matrix. Besides the networks for binding and learning the associations, the previous item needs to be stored. This is done with two chained gated memory networks. When the stimulus switches, the first memory network still stores the previous stimulus and transmits it to the second memory network. Once the update finished, the gate of the second memory network gets inhibited while the first memory network is allowed to receive the current item as input. The required control signal new item is provided by the rectified dot product in Fig. 11.4. To prevent premature overwriting of the first memorized item, a secondary dot product inhibiting that gate is used. The Semantic Pointer for the previous position can be trivially decoded from the position network.

Furthermore, a weight-decay for the $\boldsymbol{M}^{\text{CF}}$ and $\boldsymbol{M}^{\text{FC}}$ association matrices

needs to be introduced for either approach. This prevents the weights in these matrices from growing too large which would induce saturation effects in other parts of the model. It also prevents previously learned associations from providing noise overpowering the associations from the current list.

These extensions should have minimal influence on the single-trial CUE model. The additional learned association matrices use a learning rate much lower than the $\boldsymbol{M}^{\mathrm{CF}}$ and $\boldsymbol{M}^{\mathrm{FC}}$ matrics. Thus, the single-trial performance is dominated by those latter matrices. The added decay requires adjustment of the recall parameters, but will also play a more significant role on longer timescales, i.e., when new associations are learned after a recall phase, but not when items are presented in quick succession.

12. Results

To validate the CUE model, I matched it against human experimental data of serial and free recall experiments. The same model architecture was used in all of these simulations, except for the Hebb repetition effect where the extensions discussed in the previous sections have been used. Parameter values were kept constant across conditions as much as possible, but some small changes were necessary in some instances as noted when results are discussed. The simulations were designed to replicate the experimental paradigms as closely as possible. In particular, the list length, item presentation times, delay times, and recall times were matched to the actual experiments with human subjects.

To model the effect of distractor tasks during delay phases, non-list items where presented a rate of ϕ items per second. These were allowed to influence the STM component, but learning in the LTM component was disabled as these items were irrelevant to the main memorization task. This is similar to how Howard and Kahana (2002) modeled the distractor interval, though in their case they did not define the distractor rate, but the effective length of the distractor interval. As I was aiming to match the experimental paradigms as closely as possible, changing the length of the distractor interval was not an option.

Unless otherwise noted, 100 simulations with different seeds were run per experimental condition. This number is sufficient to get clear results with reasonably small confidence intervals, but still small enough that the simulation and parameter matching is feasible on a high-performance computing cluster. The values used for free parameters in the different settings are summarized in Table 12.1. In addition to these, the context drift parameter was set to $\beta = 0.62676$ and the OSE short term decay was set to $\gamma = 0.9775$ in all simulations. These are the values reported as best fitting by Sederberg, Howard, and Kahana (2008) and Choo (2010) respectively. Some other parameters, like the OSE scaling ρ for episodic memory and the TCM ratio γ for updating $\boldsymbol{M}^{\mathrm{FC}}$, are not present in the CUE model. The extensions for the Hebb repetition effect introduced two additional parameters. The learning rate for the direct

Table 12.1.: Summary of free parameters values for distractor rate ϕ, probability ψ of using the serial recall strategy, bias of the null choice μ in recall, standard deviation of the input noise σ in recall, and the AML learning rate η for $\boldsymbol{M}^{\mathrm{CF}}$ and $\boldsymbol{M}^{\mathrm{FC}}$. See text for discussion of the parameter choices and two additional parameters in the Hebb repetition condition not listed in the table.

Experimental condition	ϕ/s^{-1}	ψ	μ	σ	η
Serial recall					
Immediate	—	1	0.04	0.009	10
w/o STM	—	1	0.04	0.009	10
w/o LTM	—	1	0.03	0.015	10
Free recall					
Immediate	—	0.1	0.04	0.015	10
Delayed	0.4	0	0.0325	0.015	10
Continuous distractor	0.3	0.1	0.03	0.009	10
Scopolamine					
Placebo	—	0.1	0.02	0.015	10
Scopolamine	—	0.1	0.02	0.015	0.1
Hebb repetition	—	1	0.015	0.009	10

or forward associations was set 0.05 or 0.25 respectively. The decay rate for the $\boldsymbol{M}^{\mathrm{CF}}$ and $\boldsymbol{M}^{\mathrm{FC}}$ weights was set to 0.999 973 176 which corresponds to a decay to about 0.2 of the original weight over a period of 60 s with a simulation timestep of 1 ms. This decay required adjusting the bias of the null choice to $\mu = 0.015$.

12.1. Serial recall

In serial recall participants are asked to recall items in the same order as they were presented. In an experiment presented by Jahnke (1968), lists of ten items were presented at the rate of one item per second and recalled immediately by the 96 subjects. Figure 12.1 shows the serial position curve for the experimental and model data and the distribution of transposition errors averaged over all serial positions. In both cases the serial position curve shows a clear primacy

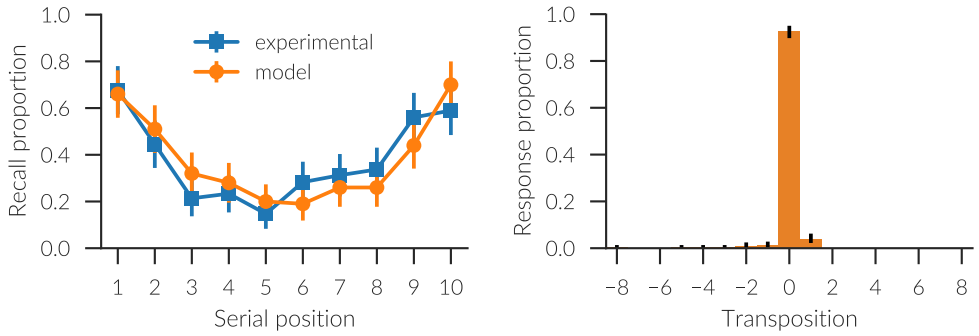


Figure 12.1.: Serial position curve (left) and transpositions (right) for the serial recall of a 10 item list with the CUE model. The experimental data from Jahnke (1968) is shown for comparison in the serial position curve (blue squares). The error bars show 95 % confidence intervals.

and recency effect. The model predictions are statistically indistinguishable from the human data as all confidence intervals overlap. While Jahnke (1968) did not provide the transposition data, the model qualitatively matches the transposition gradients reported elsewhere (e.g., Henson 1996). In general, few transposition errors are made, but for those that occur, transpositions of nearby items are more likely than transpositions of distant items.

The model allows selective disabling of the recall from the STM or LTM component. Doing so, with appropriate adjustment of the recall noise level to account for the reduced evidence input, shows that the primacy effect is mediated by the LTM, while the recency effect depends on the STM (Fig. 12.2). The recall performance without the LTM contribution is also much worse. This might be the case either because the input to the recall network might need further adjustment or because no rehearsal mechanism is modelled, resulting in drift of the OSE integrator.

12.2. Free recall

While the order of recall is predetermined in serial recall, in free recall list items may be recalled in any order. Here, I provide the model match to the data from Howard and Kahana (1999), which has also been used in the original fits of

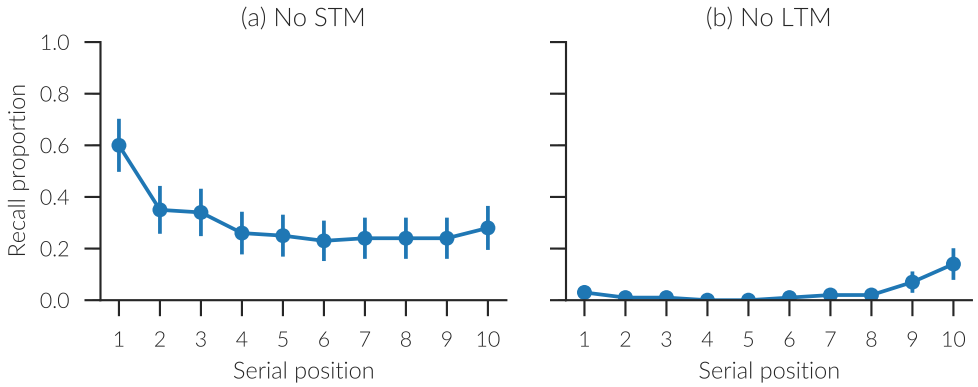


Figure 12.2.: Serial position curves when either (a) STM recall or (b) LTM recall is disabled in the model. The error bars show 95 % confidence intervals.

the TCM model in Howard and Kahana (2002) and Sederberg, Howard, and Kahana (2008). Three experimental conditions are matched: immediate recall, delayed recall, and continuous distractor recall.

In the immediate free recall condition, list items are presented at a rate of one item every second. After the presentation phase, a recall phase of 45 s followed immediately. This protocol is changed to a presentation rate of one item every 1.2 s and a recall phase of 60 s in the delayed and continuous distractor conditions. In both of these latter conditions, the presentation and recall phase are separated by a 16 s distractor task. In addition, in the continuous distractor conditions such a 16 s distractor phase is inserted in between every pair of items. A list length of twelve items is used in all conditions. The experimental data was obtained from 65 subjects presented with 25 lists each for the immediate recall condition, and from 16 subjects presented with 15 lists each for the remaining conditions.

Four resulting metrics from the model and experimental data are shown in Fig. 12.3. First, the distribution of the total number of successful recalls is shown. To my knowledge, this data has not been analyzed for the original TCM model, even though it is arguably the most fundamental comparison. In all conditions, the 95 % confidence intervals of the mean, standard deviation, and kurtosis overlap with the exception of the kurtosis in the immediate recall condition. Thus, no significant difference for the most essential moments is shown, which indicates that the model approximates the experimental distributions well,

even though an equality of the distributions cannot be inferred.

Second, the serial position curves are given. The strong recency effect in immediate recall is attenuated in delayed recall, but reappears to some degree in continuous distractor recall. Interestingly, the recency effect in immediate recall gives the curve an S-like shape that is missing in continuous distractor recall.

Third, these effects also show in the probability of first recall. In immediate recall, the first recall is, with high probability, from the end of the list, whereas in delayed recall the probability is much more uniform. In continuous distractor recall, the probability to start the recall at the end of the list is partially restored.

Finally, the conditional response probability (CRP) gives the probability of how much lag there is between the positions of two recalled items. For example, the asymmetry in immediate recall shows the bias to do forward recall and the peak around zero that nearby items tend to be recalled together. Both of these effects become attenuated in delayed and continuous distractor recall. In delayed free recall, the model predicts a stronger forward bias than the experimental data shows.

The model provides an excellent fit on most measures. The confidence intervals of 101 of the 108 data points overlap. This amounts to less than 7 % confidence intervals that do not overlap, while about 5 % are expected to be non-overlapping by pure chance given a 95 % confidence level. The most salient deviation of the model and experimental data is observed in the CRP curve for the delayed recall condition. The model predicts a slightly higher forward bias than is actually found.

12.3. Scopolamine

A spiking neural network model allows the investigation of the effects of drugs more readily than a pure mathematical model. I demonstrate this here with the acetylcholine antagonist scopolamine. Administered before the presentation phase in an immediate recall experiment, scopolamine is detrimental to recall performance (Ghoneim and Mewaldt 1975). However, scopolamine administered in between the presentation and recall phase does not influence performance. This indicates that scopolamine prevents encoding of new memories in LTM, but does not prevent recall of already encoded memories. More precisely, scopolamine has been shown to attenuate long-term potentiation in

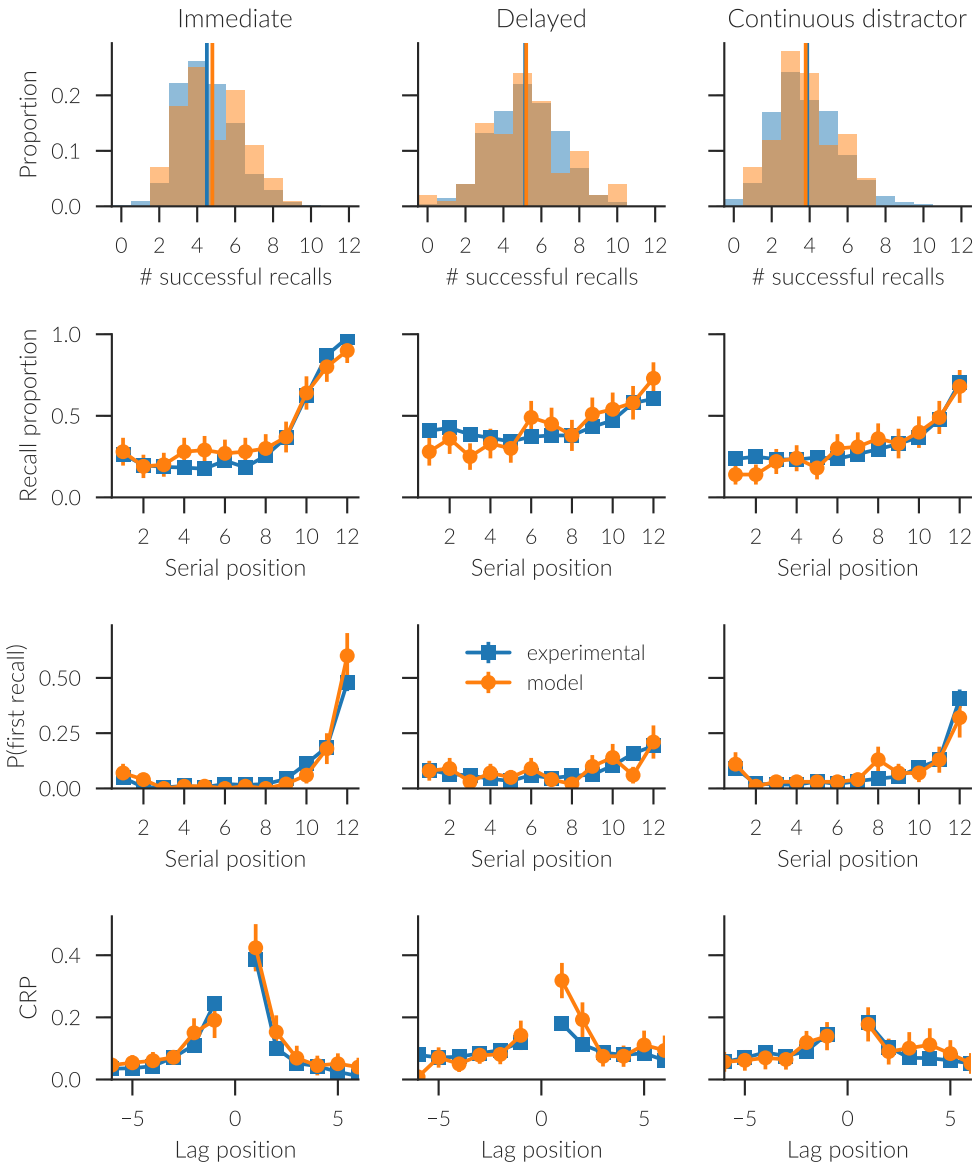


Figure 12.3.: Comparison of experimental and model free recall data. The columns show the immediate, delayed, and continuous distractor conditions. The rows show from top to bottom: distribution of the number of successful recalls (mean marked by vertical line), serial positions curves, probability of first recall, and the conditional response probability. The error bars show 95 % confidence intervals. Experimental data by Howard and Kahana (1999).

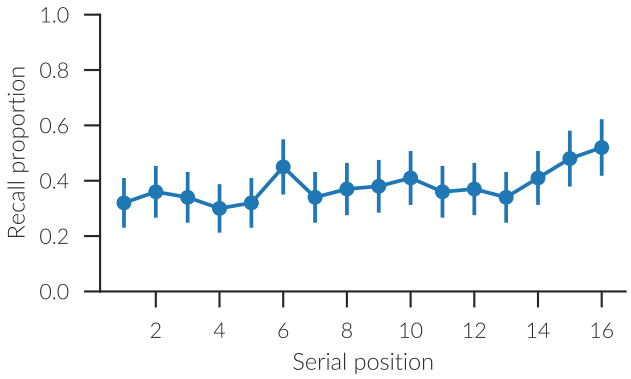


Figure 12.4.: Serial position curve with a scopolamine injection predicted by the CUE model. The error bars show 95 % confidence intervals.

hippocampus (Leung et al. 2003; Ito, Miura, and Kadokawa 1988; Hirotsu et al. 1989).

To model the effect of scopolamine on LTP, I adjusted the AML learning rate for learning the $\boldsymbol{M}^{\mathrm{CF}}$ and $\boldsymbol{M}^{\mathrm{FC}}$ matrices. With this approach the experimental results obtained by Ghoneim and Mewaldt (1975) from 36 subjects (8 trials each) are reproduced. I focus here on the immediate recall experiment with 16 item word lists. The simulation protocols were again modeled to replicate the experimental settings, with a presentation time of two seconds per item. To obtain a similar effect, the normal AML learning rate was used before the time point of scopolamine injection and in placebo trials. After the time point of scopolamine injection it was set to 0.1.

Ghoneim and Mewaldt (1975) reported a recall accuracy of 77.92 ± 4.94 % (mean $\pm$ standard error) in the placebo condition and a recall accuracy of 31.67 ± 2.28 % in the scopolamine condition. The model produces recall accuracies of 74.18 ± 1.08 % and 37.94 ± 0.99 % respectively. Moreover, the serial position curve for the scopolamine condition shows no primacy effect, but a recency effect (Fig. 12.4).

12.4. Hebb repetition effect

To model the Hebb repetition effect, the extensions described in Section 11.2 where required. The simulations for the effect were modelled after Hebb (1961).

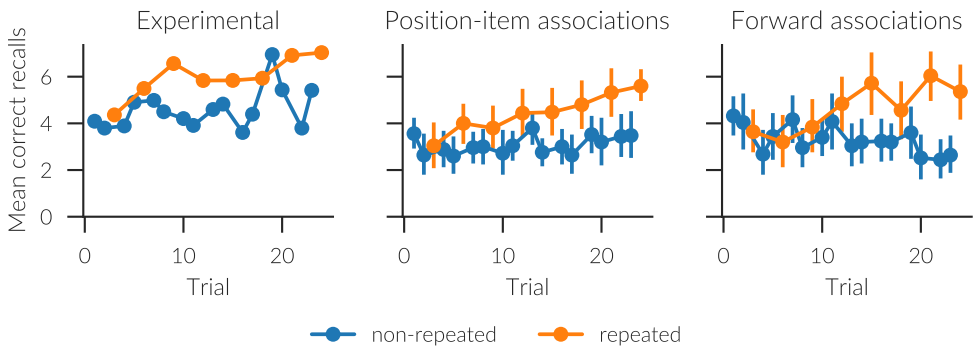


Figure 12.5.: Experimental and model data showing the Hebb repetition effect. Nine item lists were presented and one list was repeated on every third trial. From left to right: experimental data (Hebb 1961), model data with direct learning of position to item associations, and model data with learning of forward associations. The error bars show 95 % confidence intervals (no confidence intervals were provided for the experimental data).

A total of 25 model instances, equivalent to the 25 experimental subjects, were run for 24 consecutive trials each. Each trial consisted of a nine item list (of the digits from one to nine in random order) and starting with the third trial every third list was identical.

While the model performance seems slightly worse overall, the qualitive Hebb repetititon effect is reproduced (Fig. 12.5). In both the experimental and all sets of model data, the number of correct recalls increases by about two to three items on the repeated list.

12.5. Memory encoding

As a spiking neural network model, the CUE model allows the recording of spikes and examination of changes in neural firing (Fig. 12.6). The distribution of active neurons changes for the STM neurons with each item, with a delay of about 250 ms to encode the new item. When no new item is present, persistent neural firing preserves the firing pattern. Similar behaviour is observed for neurons encoding the current context signal that gets updated with about a 100 ms delay. The LTM for $\boldsymbol{M}^{\mathrm{CF}}$ (and $\boldsymbol{M}^{\mathrm{FC}}$, but not shown) is encoded in neural

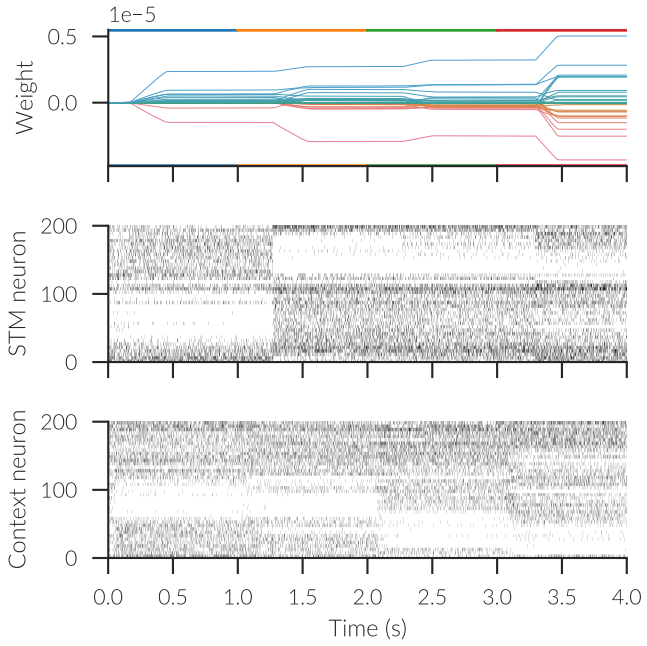


Figure 12.6.: Memory encoding in the CUE model. From top to bottom: weights encoding $\boldsymbol{M}^{\mathrm{CF}}$ in LTM, spiking activity of a subset of STM neurons, and spiking activity of a subset of neurons encoing the context signal $\boldsymbol{c}$. The colored bars at the top mark the presentation of four different list items.

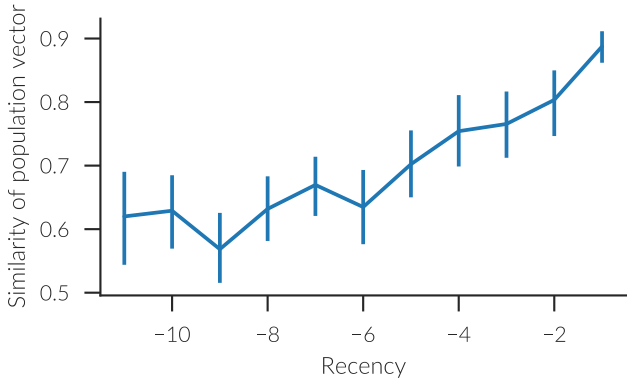


Figure 12.7.: Change in similarity of the context population vector with recency.

weights that change for each list item to encode the newly learned associations.

Folkerts, Rutishauser, and Howard (2018) recorded from the medial temporal lobe of epilepsy patients and found that the population activity vector exhibits a neural recency effect. The similarity of the population vector decreases with lag between presented items. A similar analysis can be done in the CUE model by recording spikes from the context component of the model. The population vector is given by the average firing rates during the presentation window of each item in a list. A decline in similarity qualitatively similar to Folkerts, Rutishauser, and Howard (2018), when comparing to less recent population vectors, can be found as shown in Fig. 12.7. The absolute similarity values are higher than reported in the experimental data which can be attributed to the possibility of specifically recording from the neurons responsible for representing the context, whereas experimentally many unrelated neurons will be included in the analysis.

Furthermore, Ninokura, Mushiake, and Tanji (2003) identified neurons with activity selective to the serial order of items in the lateral prefrontal cortex of monkeys. Using permutations of a three item list, analogous to Ninokura, Mushiake, and Tanji (2003), similar neurons can be found among the STM neurons of the CUE model (Fig. 12.8). Besides neurons selective to a single permutation of items that fire persistently during the delay period, some neurons appear selective for one permutation and either increase or decrease their firing rate during the delay period. Some neurons are selective for multiple sequences. Figure 12.8d shows a neuron that responds to sequences starting with ZY, but also the sequence XYZ. Finally, many neurons are much more

complex in their responses and cannot easily be assigned a specific preferred stimulus sequence (no figure shown).

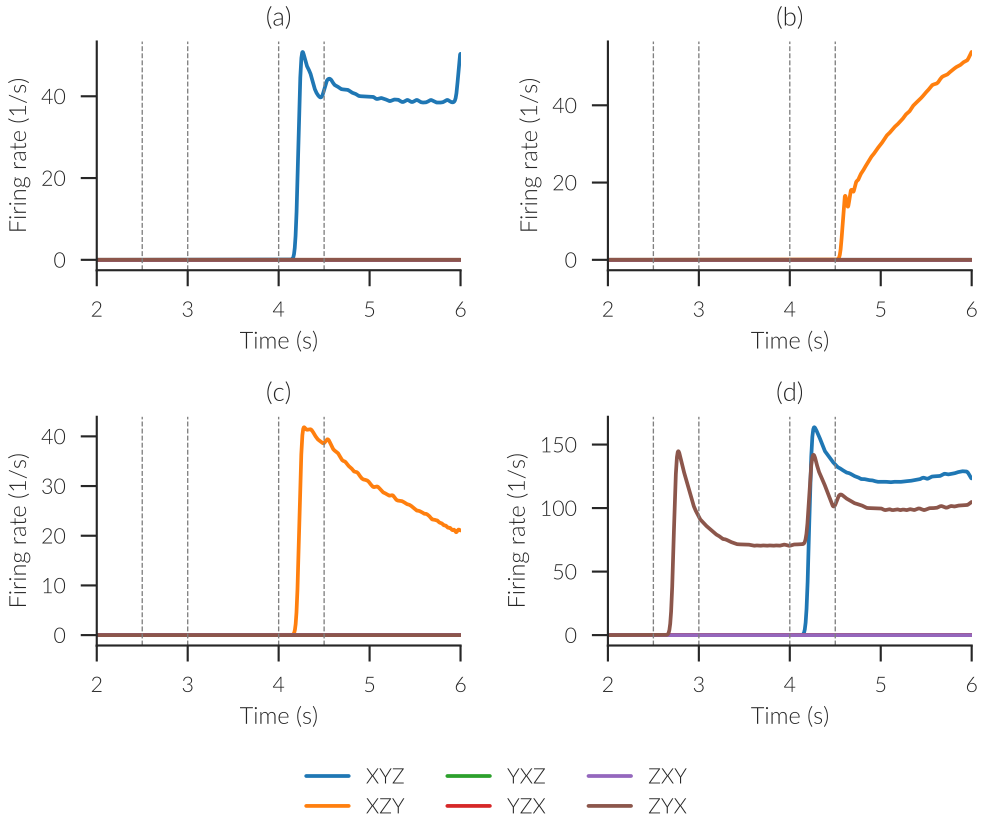


Figure 12.8.: Firing rate of selected STM neurons, smoothed with a Gaussian filter with a standard deviation of 25 ms, in response to permutations of a three item list. Analogous to Ninokura, Mushiake, and Tanji (2003), each list item was presented for 0.5 s preceded by 1 s delays. The presentation of all items was followed by a 1.5 s delay period. The presentation intervals for the second and third item are marked with the vertical lines.

13. Discussion

In this thesis, I presented the context-unified encoding (CUE) model. To my knowledge, it is the first spiking neural network model of human memory that integrates activity-based short-term memory and weight-based long-term memory. The same model matches a variety of behavioural data from serial and free recall experiments, but in contrast to previous models provides a hypothesis of a neural mechanistic explanation.

The CUE model exhibits many of the hallmark findings in memory research. It shows the primacy and recency effect in immediate serial and free recall. These effects get attenuated in delayed free recall, but in continuous distractor free recall the recency effect reappears. Furthermore, the model was found to make very few transposition errors in serial recall, and if it does so nearby items are transposed. In the free recall conditions, the model tends to start with items at the end of the list, recall nearby items together, and favour recall in forward direction. Introducing delays and distractors attenuates these effects. All of these observations match the findings from experiments with human subjects.

Not only are these qualitative effects reproduced, but also the quantitative match to the data is very good. Only few significant differences, close to the number of differences expected by chance, were found. One of these differences is worth considering in more detail: the model predicts a too strong forward bias in delayed free recall with both the lag 1 and 2 values of the CRP curve being significantly above the experimentally found values. Interestingly, this is also highlighted as the least well matched aspect in the original TCM (Howard and Kahana 2002). While in that publication the TCM prediction is closer to the experimental data, the TCM prediction from a more recent paper (Sederberg, Howard, and Kahana 2008) is closer to the CUE model prediction. This makes it likely that the difference is not based on pure statistical chance, but that both the TCM and CUE model do not capture an essential aspect of memory, potentially related to the evolution of the context signal, that leads to reduced forward bias in delayed free recall. It remains for future work,

to precisely identify the reason for this mismatch and to extend the model. Despite Farrell and Lewandowsky (2008) criticizing the TCM for being unable to simultaneously match the probability of first recall and CRP curves, (but see Howard, Sederberg, and Kahana 2009), the CUE model matches both forms of the data simultaneously. It also uses contextual evolution during the recall process in all simulations addressing another criticism.

Extending the CUE model with slow learning of either direct position to item associations or forward associations allowed the reproduction of the Hebb repetition effect qualitatively. The involvement of such secondary learning should be considered a model prediction, as the effect could not be obtained without this extension. Modelling the Hebb repetition effect also extends the model to effects across multiple trials of memory experiments. This is done by only few memory models, in particular, neither the OSE, nor the TCM model attempted to match this kind of data.

As opposed to pure math models, the implementation as a spiking neural network allows the comparison and validation of the model against data from neural recordings in addition to the behavioural data. Neurons sensitive to the serial order of items and a neural recency effect could be found. Both findings have also been reported in experimental data. In Section 9.6, it was demonstrated that the proposed mechanism of association learning is able to explain neural data. Unfortunately, neural data recorded from humans in memory experiments is still scarce, because invasive recordings can only be done when such recordings are required for medical reasons. Nevertheless, implementing models with spiking neurons is worthwhile for several other reasons, despite the more complicated model construction and increased simulation times. Drug effects, like scopolamine, can be more readily modelled, as was done with CUE model (though I note that Hasselmo and Wyble 1997 did address the effects of scopolamine with more detail). Also a higher degree of biological plausibility is achieved as one is forced to consider, for example, spiking noise and synaptic time constants. This prevents common assumptions like arbitrary precision or perfectly orthogonal vectors made in many math models, and forces the consideration of dynamics.

The spiking neural implementation also helps to constrain many parameter values. Synaptic time constants, membrane time constants, and similar cellular physical quantities can be set to biologically plausible values reported in experimental findings. These are fixed parameters that have not been adjusted for matching the behavioural data. Similarly, as in the NEF, most connection

weights are directly determined by least-squares minimization to implement a given function determined by the prescribed model architecture, hence the connection weights are fixed as well. This leaves the model with very few free parameters.

To match the immediate serial recall, only two parameters were adjusted: the bias of the null choice μ and the input noise standard deviation σ in recall. Both account for the fact that the recall network was restricted to recalling the items used within the memory experiment, while in reality a number of other items might interfere with the recall process. For free recall experiments, one additional parameter ψ is added that determines the probability of using the serial recall strategy even for free recall. (For serial recall, a fixed value of $\psi = 1$ is implied as no free recall is allowed.) Furthermore, in experiments with delay periods, a distractor rate ϕ needs to be set. To simulate the effect of scopolamine the AML learning rate η was adjusted. However, in non-scopolamine conditions, it was treated as a fixed parameter and set to a value high enough to learn associations until the threshold for inhibition was achieved within the presentation duration. Even higher values would not have any effect as long as it does not largely exceed the inverse of the synaptic time delay of the inhibition. Lastly, only two additional parameters (weight decay rate and a separate learning rate) were introduced by the model extension to the Hebb repetition effect.

While few free parameters are desirable with respect to model parsimony, they should also be assigned similar values to model-related experimental conditions. This is mostly the case for the CUE model. The bias of the null choice in recall ranges from 0.03 to 0.04 and values get monotonically smaller as the task difficulty increases with additional delays. This corresponds to plausible longer recall attempts in more difficult experimental conditions. Only a small difference is also observed in the distractor rates (0.3 to 0.4) and the probability of using a serial recall strategy (zero for delayed recall and 0.1 in all other free recall conditions). However, the noise standard deviation σ in recall differs by a factor of more than 1.5 without a clear relation to the experimental condition. It is hard to hypothesize potential reasons for this difference as the parameter is accounting for things not explicitly modeled in recall.

It is also of interest how robust the model is against parameter changes. I have not done a formal analysis of this because the model simulation times are prohibitive. However, this also means that only a small set of parameter values without a lot of fine tuning has been tested (less than 200 combinations summed

over all experimental conditions). Given that finding the right parameters with few simulations is less likely if the model were highly sensitive to the parameter choice, a sufficient robustness to the exact choice of parameter values can be expected.

The CUE model is based on prior models of memory, but improves on them in important ways. With regard to the OSE model, two main advancements can be stated. First, the episodic memory buffer has been replaced with a much more plausible long-term memory mechanism that relies on synaptic-weight changes rather than reverberating neural activity. Second, the CUE model also implements the mechanism providing the position tags fully in spiking neurons.

Implementing a long-term memory component based on the TCM in a spiking neural network provides a strong support for the biological plausibility of the TCM that previously was missing. Certain simplifications of the TCM equations in this process to facilitate this implementation highlight which aspects of the TCM are essential and which do not contribute to the explanation of the data. In particular, it also shows that certain assumptions, like perfectly orthogonal vectors, useful in the mathematical analysis, are not essential. In addition, the modified TCM has been extended with a short-term memory component in the CUE model. While the TCM has been posited as a single-store model, this has been criticized (Davelaar et al. 2008). The CUE model demonstrates that treating the TCM as part of a multi-store model is not unreasonable, provides good matches to the free recall data, and in addition allows matches to serial recall data. Finally, the recall process in the TCM has been replaced with a more plausible spiking neural mechanism (Chapter 10). Existing versions of the TCM used either the much more abstract Luce's choice rule with unclear biological plausibility, or a competitive queueing mechanism that has been shown to be not in accordance with experimental data (Davelaar 2007).

In the broader context of memory models, the CUE model is unique as providing a low-level spiking network implementation, but matching high-level behavioural data. This includes the recall process that is not explicitly modeled in many other models. Furthermore, due to the item based context, there is no reinstantiation problem found in most context-based memory models.

A key part of the CUE model is the association matrix learning rule. It provides insight into how one-shot learning without catastrophic forgetting is possible. Several options and their biological plausibility of how this learning rule might be realized in the brain have been discussed in Chapter 9. The

main point there is that either some form of weight-sharing or symmetric decoder matrix is needed, or the input needs to be transformed into a sparse representation. The dentate gyrus of the hippocampus exhibits such sparse firing and is implicated in associative learning, giving support to the latter hypothesis. However, I also provided evidence that a symmetric decoder matrix might be learned in a biological plausible way. Moreover, using the AML for learning simple pairwise associations, allows the reproduction of changes in firing rates that have been observed in recordings from human hippocampus. These results are not only of interest for the CUE and TCM models, but many other cognitive models that assume the storage of associations in a similar association matrix without further explanation of how these associations are learned.

One potential criticism of the CUE model could target the method of position counting (Section 7.2). Only a limited number of positions can be represented, even though this number can be configured as large as permitted by neural resources. If the number of positions is exceeded, the representation can be made to wrap around back to the first position. Thus, the model does not need to fail catastrophically, but a specific pattern of recall errors could be introduced by encoding different sequences of items to the same positions. However, an experimental test of such predictions will be hard, as long lists are likely required, and thus for most positions no item is recalled above chance. It is worth highlighting that the position counting network in the CUE model can be replaced easily to test other hypotheses. But it seems unlikely that the limited number of positions can be eliminated completely, because there will always be a limited number of almost orthogonal Semantic Pointers that fit into a vector space of given dimensionality.

13.1. Anatomical mapping

Given that the CUE model is neural, it is of interest to consider how the parts of the model map to brain areas. Howard, Fotedar, et al. (2005) proposed a mapping of the TCM model to brain areas that applies to a large degree also to the TCM-based part of the CUE model. There are, however, some details to be reconsidered, and the OSE-based STM part of the model has not been discussed.

The medial temporal lobe (MTL) is known to be essential for free recall.

Damage to the MTL is detrimental to free recall performance (Graf, Squire, and Mandler 1984). Thus, we can assume that the TCM-related parts of the model reside in the MTL.

More precisely, the context storage network can be mapped onto parahippocampal areas, in particular the entorhinal cortex (EC). Its properties are consistent with the storage of non-spatial memories for tens of seconds. In delay periods, stimulus dependent persistent activity can be observed (Suzuki, Miller, and Desimone 1997; Young et al. 1997). Quirk et al. (1992) showed EC has a higher mean firing rate than hippocampus, which is not caused by short bursts, and is thus compatible with the sustained maintenance of neural firing. Also, the electrophysiological properties of the EC support integration (Egorov et al. 2002). These findings are, however, based on intrinsic cell properties, whereas the CUE model uses recurrent connectivity for integration instead. Note that the context network also contains integration ensembles that maintain the context signal over the timespan of seconds.

While the EC firing is modulated to some degree by the item position, this coding is more noisy than in the hippocampus (Quirk et al. 1992). This indicates that EC codes for additional information. L. M. Frank, E. N. Brown, and Wilson (2000) have shown that superficial EC employs retrospective coding, i.e., that it differentiates visits to the same position by the history leading up to that visit. This is consistent with a context signal encoding the history of items leading up to the current item.

The other major components to map to brain structures are related to the learning and retrieval of associations in the $\boldsymbol{M}^{\mathrm{FC}}$ and $\boldsymbol{M}^{\mathrm{CF}}$ matrices. Howard, Fotedar, et al. (2005) stated that the $\boldsymbol{M}^{\mathrm{FC}}$ matrix might not be implemented by a single anatomical region due to its complicated structure. However, the updating equation for $\boldsymbol{M}^{\mathrm{FC}}$ in the CUE model has been simplified, which makes the correspondence to a single region more plausible. The learning of new associations in these matrices is attributed to the hippocampus by Howard, Fotedar, et al. (2005). This is consistent with the results about the AML presented in Chapter 9.

The CUE model provides a more detailed description of the updating of the association matrices due to the neural implementation by means of the AML. The learning network has recurrent connectivity to inhibit the learning once an association has been learned with the desired strength. Recurrent connectivity is also found in the CA3 region of hippocampus. Furthermore, CA3 receives input via the dentate gyrus and directly from entorhinal cortex, which could

correspond to separate transmission pathways for the associated cue and target. Interestingly, the AML highlighted the need to incorporate the decoder matrix $\boldsymbol{D}^\top$ into the connections. While I have shown it might be plausible that this connectivity is learned, the matrix could be reduced to the identity if the input ensemble were to provide orthogonalized inputs. The dentate gyrus, providing input to CA3, is commonly assumed to perform such orthogonalization given its large neuron count, sparse firing, and neurogenesis (e.g., Boss et al. 1987; Jung and McNaughton 1993; Piatti, Ewell, and J. K. Leutgeb 2013). Thus, while the CUE model does not explicitly model the dentate gyrus (which is a significant research problem in itself), the learning rule used at least provides a principled reason for its existence.

Further evidence for this neuroanatomical mapping can be obtained from the connectivity between hippocampus and EC. The superficial EC provides input to hippocampus, but does not receive direct input for hippocampus (Witter 2010). In contrast to that, the deep layers of EC receive input from hippocampus and might be relevant for recall, especially the recall of pre-experimental context. This is consistent with the connectivity in the model where the context network projects to the association matrix learning network attributed to hippocampus. The learning network for $\boldsymbol{M}^{\mathrm{FC}}$ also projects back to an ensemble recalling the prior context before it gets combined in a different ensemble.

The short-term memory related components of the CUE model can be assumed to correspond to cortical areas, in particular the prefrontal cortex. The prefrontal cortex has been found to be involved in working memory tasks in many studies (e.g., Goldman-Rakic 1995; Owen 1997).

13.2. Optimally fuzzy temporal memory

Shankar and Howard (2013) proposed a mathematical model of an optimally fuzzy temporal memory. According to them it can replace the context signal in the TCM (Howard, Shankar, et al. 2015). Thus it could also be relevant to the CUE model, but I argue below that a spiking neural version of this memory is unlikely to work without an implausible number of neurons.

The fuzzy temporal memory is constructed with the aim of representing past values of a function $f(t)$ with a scale-free fall off in accuracy. Note that multiple such fuzzy temporal memories could be combined to represent a vector-valued

function as needed for the context signal in the TCM or CUE models. The memory itself consists of a set of independent, leaky accumulators given by the differential equation

$$\frac{\mathrm{d}c_i(t)}{\mathrm{d}t} = -s_i c_i(t) + f(t) \tag{13.1}$$

where s_i are decay constants. This set of integrators is essentially computing a Laplace transform. To readout the memory, the Laplace transform is inverted approximately with a linear operator $\boldsymbol{L}_k^{-1}$ given by

$$\left[\boldsymbol{L}_k^{-1}\right]_{ij} = \frac{(-1)^k}{k!} s_i^{k+1} \left[\boldsymbol{D}_k\right]_{ij} \tag{13.2}$$

where $\boldsymbol{D}_k$ is a square matrix that computes the k-th discrete derivative with respect to s. The reconstructed $\hat{f}_t(t + t_i^*) = \left[\boldsymbol{L}_k^{-1}\boldsymbol{c}(t)\right]_i$ with $t_i^* = -k/s_i < 0$ estimates the values of f at times $t + t_i^*$ in the past. The reconstruction becomes more accurate as $k \to \infty$.

Shankar and Howard (2013) derive a signal to noise ratio, but use the magnitude of a delta impulse input to do so. As the magnitude of a delta impulse is infinite in the limit, the signal to noise ratio also grows without bound for $k \to \infty$. However, it is physically impossible to deliver a perfect delta impulse. Instead it is more appropriate to analyze the steady state response for a fixed input $f(t) = u$. In the context of the NEF, $u = 1$ can be assumed without loss of generality because any change in the magnitude u requires a matched change in the representational radius r which will also scale the absolute error, keeping the relative error the same. The steady state of the leaky integrators is then given by $c_i(t) = s_i^{-1}$. This implies that NEF ensembles representing c_i should use a radius of $r = s_i^{-1}$.

The amplification of the noise standard deviation is given by Shankar and Howard (2013) as

$$g_\eta(s_i, k) = \frac{\sqrt{2^k} s_i^{k+1}}{\delta_{s_i}^k k!} \tag{13.3}$$

where $\delta_{s_i} = s_i - s_{i-1}$. When following their proposal for optimal spacing of the s_i by picking $t_i^* = (1 + \nu)^{i-1} t_1^*$ for some constant $\nu > 0$, it follows that

$\delta_{s_i}/s_i = \nu$ and the formula can be simplified to

$$g_\eta(s_i, k) = \frac{\sqrt{2^k} s_i}{\nu^k k!} = -\frac{\sqrt{2^k}}{\nu^k (k-1)! t_i^*}. \tag{13.4}$$

Note that this amplification is not independent of $t_i^* = -k/s_i$, but increases as $t^* \to 0$. This can be seen easily when adding some Gaussian noise to all c_i before the reconstruction as done in Fig. 13.1a. It also refutes that "the signal to noise ratio will remain constant over all timescales" (Shankar and Howard 2013); a statement based on the assumption of a perfect delta impulse. However, as stated above, NEF ensembles should use a radius of $r = s_i^{-1}$ which scales the noise by the same factor and thus indeed leads to a constant amplification of the noise across timescales (Fig. 13.1b) given by

$$g_{\eta,\mathrm{NEF}}(\nu, k) = \frac{\sqrt{2^k}}{\nu^k k!}. \tag{13.5}$$

Figure 13.2a shows this amplification for different parameter values. When both ν and k are chosen large enough, noise will actually be attenuated. However, this also increases the timescale (Fig. 13.2b) and as such there is a trade-off between the time resolution of the memory and noise amplification. This increase in timescale is given by

$$\tau_i(\nu) = -(1+\nu)^k t_i^* \tag{13.6}$$

and is caused by the discrete approximation of the derivative that relies on unequally spaced s_{i-k} to s_{i+k} for the reconstruction of $\hat{f}_t(t+t_i^*)$ and is dominated by t_{i+k}^* (Appendix D.1).

Based on these equations, the total number of neurons N_{tot} required can be estimated as (Appendix D.2)

$$N_{\mathrm{tot}}(t_1^*, k) = N d\, g_{\eta,\mathrm{NEF}}^2(\nu, k)\,(M + 2k) \tag{13.7}$$

$$M \geq \frac{\log(t_{\max}^*/t_1^*)}{\log(1+\nu)} \tag{13.8}$$

$$\nu \leq \sqrt[k]{-\tau_1/t_1^*} - 1 \tag{13.9}$$

where N is the number of neurons to represent a single dimension with sufficient accuracy (before the effect of the noise amplification), d is the dimensionality of the input signal, M gives the number of required leaky integrators, τ_1

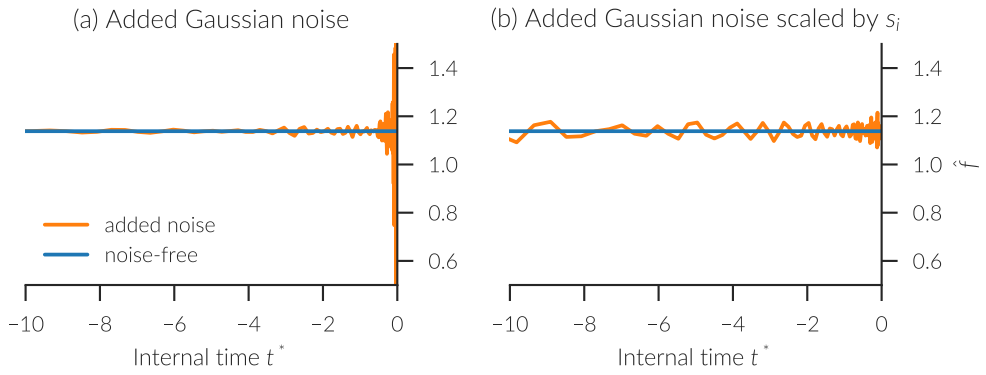


Figure 13.1.: Examples of the effect of noise in the leaky integrators c_i of the fuzzy temporal memory on the reconstruction $\hat{f}$. (a) Gaussian noise with mean zero and a constant standard deviation is added to the output of all integrators. (b) Gaussian noise with mean zero and standard deviation scaled by s_i^{-1} is added to the output of all integrators.

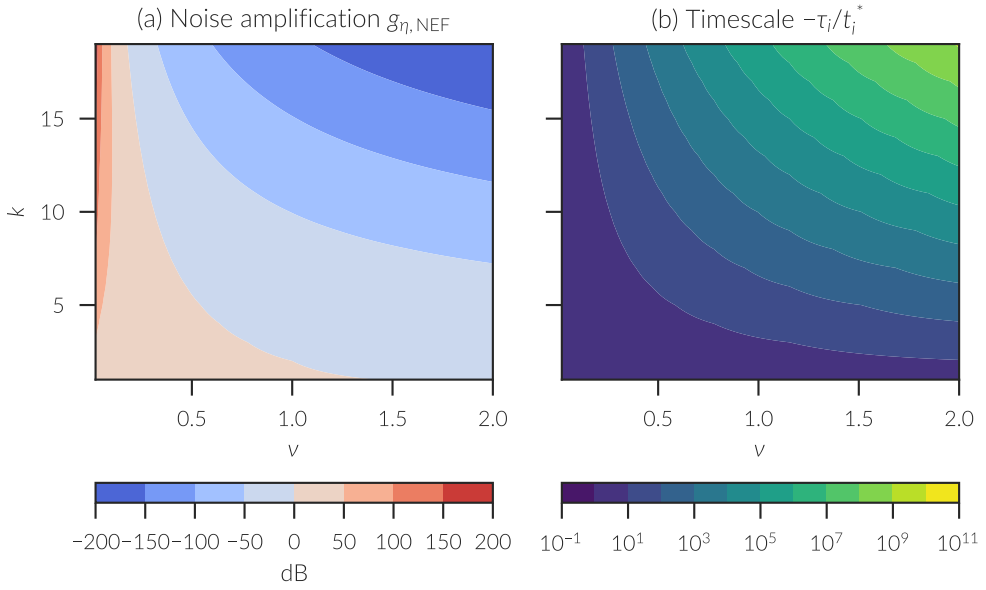


Figure 13.2.: (a) Noise amplification $g_{\eta,\mathrm{NEF}}$ in the fuzzy temporal memory. (b) Timescale τ_1 for the t_1^* reconstruction of the fuzzy temporal memory.

gives the desired smallest timescale, and $t^*_{\max}$ gives the desired timespan of the fuzzy memory. Assuming that the fuzzy temporal memory is to be used for the TCM context signal, $\tau_1 \leq 1\,\mathrm{s}$ and $t^*_{\max} \geq 10\,\mathrm{s}$ are reasonable choices because ten item lists with a presentation rate one item per second are typical for the memory experiments matched in this work. Furthermore, $N = 50$ and $d = 64$ can be used as conservative estimates. Using 50 neurons with maximum firing rates between $200\,\mathrm{s}^{-1}$ to $400\,\mathrm{s}^{-1}$ is sufficient to read out the represented value, but not very precisely. This default range of firing rates is much higher than firing rates typically observed in vivo. Thus, the actual required number of neurons can be expected to be higher. A dimensionality of $d = 64$ is also on the lower end to be able to fit sufficiently many almost orthogonal vectors into the space. The CUE model uses four times as much, $d = 256$, dimensions. Ultimately, the choice of N and d has only a minor influence on the results as they are just linear factors while the required number of neurons grows exponentially for $|t^*_1|$, $k \to \infty$. The results for this choice of parameter values is shown in Fig. 13.3. For most parameter combinations, the number of about 13×10^6 neurons in the entorhinal cortex (West and Slomianka 1998), assumed to be the locus of the CUE/TCM context signal, and even the number of about 86×10^9 neurons in the human brain (Azevedo et al. 2009) is far exceeded. For an implementation with a realistic limit on the neuron number, $|t^*_1|$, but also k, needs to be sufficiently small.

Unfortunately, choosing small $|t^*_1|$ and k is also problematic. The error of the approximate inversion of the Laplace transform with Post's formula converges only at a rate of $1/k$ (Vu Kim Tuan and Dinh Thanh Duc 2000). Shankar and Howard (2013) also comment themselves that t^*_1 needs to be kept sufficiently far from zero as it introduces a relative error in the order of $O\left(k^3\nu^2/96t^{*2}_1\right)$ in the construction of the activity of c_i.

In conclusion, the fuzzy temporal memory has only a small parameter space that allows an implementation with feasible neural resources due to noise sensitivity. This parameter space suffers from large relative errors unrelated to noise in the approximation of the inverse Laplace transform with Post's formula and discretized derivative. Currently, it is not clear whether these non-noise related errors are sufficiently small to allow the fuzzy temporal memory to be used as a context signal in the CUE model. At the same time it would require an increased number of neurons compared to the current implementation as each dimension requires a set of leaky integrators, i.e. additional values have to be stored for each dimension. In comparison, the context network for

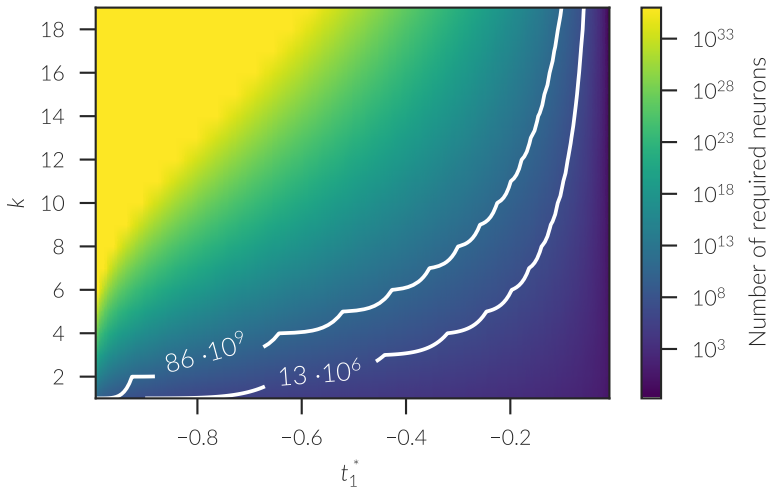


Figure 13.3.: Number of neurons required to implement a fuzzy temporal memory with a lower timescale of $\tau_1 = 1\,\mathrm{s}$ and a timespan of $t^*_{\mathrm{max}} = 10\,\mathrm{s}$. The white contours give the approximate number of neurons in the human entorhinal cortex (13×10^6) and human brain (86×10^9). See text for further details.

256 dimensional vectors uses only 51 225 neurons, and even when adjusting by a factor of 10, this is many fewer neurons than found in the human entorhinal cortex.

Apart from this theoretical argument, it might be possible to differentiate experimentally between the context representation used in the CUE model and the fuzzy temporal memory. With the fuzzy memory, the context representation gets updated continuously, whereas in the CUE model the context signal is subject to discrete updates for each presented item. Such discrete changes in the firing pattern could potentially be detected if such discrete updates are not too fine grained. A related prediction of the fuzzy temporal memory is that the context signal will depend on presentation and inter-stimulus intervals which should have an effect on the contiguity in recall if their length is varied within a trial.

13.3. Advances in large-scale cognitive modeling

The primary objectives of building the CUE model lead to a number of advances in large-scale cognitive modeling that are worth summarizing, even though not all have been presented as part of this thesis. Most importantly, optimizations for high-dimensional representations in neural networks (Chapter 6; Gosmann and Eliasmith 2016). These allow the use of fewer neurons in such models, which ultimately allows more complex networks to be built without prohibitively long simulation times. A similar benefit is the improved product network (Gosmann 2015), as the calculation of products is often required, for example for the implementation of circular convolution. Also a significant improvement to simulation times was achieved by the implementation of an optimization procedure in the Nengo simulator (Gosmann and Eliasmith 2017). Chapter 5 described a new method for binding Semantic Pointers, the vector-derived transformation binding. Even though it has not been used in the CUE model yet, it might provide a more precise binding and unbinding. Finally, the independent accumulator network described in Section 10.2 might not only be useful in recall, but for other large-scale models requiring clear decisions as part of a cognitive process.

14. Conclusion

To summarize, the context-unified encoding model advances our understanding of human memory by matching human behavioural data using an implementation grounded in a spiking neural network. The difficult task of building a spiking neuron model of this scale also led to a number advances in large-scale cognitive modeling in general, such as high-dimensional neural representations with reduced noise. Despite this, the CUE model is only a first step in the integration of neural and behavioural data as well as the understanding the interaction of short- and long-term memory. Much more data from memory experiments exists that could be matched, and can be used to highlight where the CUE model is wrong or needs to be extended. One such aspect that was already evident is the strength of the forward recall bias in delayed free recall. Future work could also focus on mapping the long-term memory components more precisely onto hippocampal structures and cellular properties in the relevant regions. In particular, introducing sparsification with a dentate gyrus model could improve the biological plausibility of the AML.

The sequences that the CUE model can memorize are much simpler than the richness of actual human memory. But they have been proven useful in psychology to investigate basic properties of memory. Also, many forms of memory can be understood as a sequence: episodic memory is essentially a sequence of events or a sequence of left and right turns might be necessary to navigate from one place to another. Thus, the model is relevant to our understanding despite the relative simplicity of modelled tasks.

Finally, the CUE model, in the broader context of large scale cognitive modeling, could provide an excellent extension to the Spaun model. A proper long-term memory component is missing from this model so far, despite being essential for cognition. The CUE model itself could also benefit from such an integration, as Spaun's ability to perform multiple tasks would allow to model experimental delay phases with an actual distractor task.

References

Abbott, L. F. and W. G. Regehr (2004). "Synaptic Computation". In: *Nature* 431.7010, pp. 796–803. DOI: 10.1038/nature03010. pmid: 15483601.

Abraham, W. C. and A. Robins (2005). "Memory retention – the synaptic stability versus plasticity dilemma". In: *Trends in Neurosciences* 28.2, pp. 73–78. DOI: 10.1016/j.tins.2004.12.003.

Anderson, J. A. (1973). "A theory for the recognition of items from short memorized lists". In: *Psychological Review* 80.6, pp. 417–438. DOI: 10.1037/h0035486.

Anderson, J. R. and M. Matessa (1997). "A production system theory of serial memory". In: *Psychological Review* 104.4, pp. 728–748. DOI: 10.1037/0033-295X.104.4.728.

Atkinson, R. C. and R. M. Shiffrin (1968). "Human Memory: A Proposed System and Its Control Processes". In: *Psychology of Learning and Motivation*. Ed. by K. W. Spence and J. T. Spence. Vol. 2. New York, NY, USA: Academic Press, pp. 89–195. DOI: 10.1016/S0079-7421(08)60422-3.

Azevedo, F. A. C. et al. (2009). "Equal Numbers of Neuronal and Nonneuronal Cells Make the Human Brain an Isometrically Scaled-up Primate Brain". In: *The Journal of Comparative Neurology* 513.5, pp. 532–541. DOI: 10.1002/cne.21974. pmid: 19226510.

Baddeley, A. D. (1986). *Working memory*. Oxford Psychology Series 11. Oxford, England: Clarendon Press. 308 pp.

– (2000). "The episodic buffer: a new component of working memory?" In: *Trends in Cognitive Sciences* 4.11, pp. 417–423. DOI: 10.1016/S1364-6613(00)01538-2.

Bahl, A. et al. (2012). "Automated Optimization of a Reduced Layer 5 Pyramidal Cell Model Based on Experimental Data". In: *Journal of Neuroscience Methods*. Special Issue on Computational Neuroscience 210.1, pp. 22–34. DOI: 10.1016/j.jneumeth.2012.04.006.

Baldo, J. V., S. Katseff, and N. F. Dronkers (2012). "Brain Regions Underlying Repetition and Auditory-Verbal Short-Term Memory Deficits in Aphasia:

Evidence from Voxel-Based Lesion Symptom Mapping". In: *Aphasiology* 26.3-4, pp. 338–354. DOI: 10.1080/02687038.2011.602391.

Bekolay, T., J. Bergstra, et al. (2014). "Nengo: A Python Tool for Building Large-Scale Functional Brain Models". In: *Frontiers in Neuroinformatics* 7.48. DOI: 10.3389/fninf.2013.00048. pmid: 24431999.

Bekolay, T., C. Kolbeck, and C. Eliasmith (2013). "Simultaneous Unsupervised and Supervised Learning of Cognitive Functions in Biologically Plausible Spiking Neural Networks". In: *35th Annual Conference of the Cognitive Science Society*. Cognitive Science Society, pp. 169–174.

Bengio, Y. et al. (2015). "Towards Biologically Plausible Deep Learning". In: arXiv: 1502.04156 [cs].

Berger, T. W. et al. (2011). "A cortical neural prosthesis for restoring and enhancing memory". In: *Journal of Neural Engineering* 8.4. DOI: 10.1088/1741-2560/8/4/046017.

Boss, B. D. et al. (1987). "On the Numbers of Neurons on Fields CA1 and CA3 of the Hippocampus of Sprague-Dawley and Wistar Rats". In: *Brain Research* 406.1–2, pp. 280–287. DOI: 10.1016/0006-8993(87)90793-1.

Botvinick, M. M. and D. C. Plaut (2006). "Short-term memory for serial order: A recurrent neural network model". In: *Psychological Review* 113.2, pp. 201–233. DOI: 10.1037/0033-295X.113.2.201.

Brown, G. D. A., T. Preece, and C. Hulme (2000). "Oscillator-based memory for serial order". In: *Psychological Review* 107.1, pp. 127–181. DOI: 10.1037/0033-295X.107.1.127.

Brun, V. H. et al. (2002). "Place cells and place recognition maintained by direct entorhinal-hippocampal circuitry". In: *Science* 296.5576, pp. 2243–2246. DOI: 10.1126/science.1071089. pmid: 12077421.

Burgess, N. and G. J. Hitch (1992). "Toward a network model of the articulatory loop". In: *Journal of Memory and Language* 31.4, pp. 429–460. DOI: 10.1016/0749-596X(92)90022-P.

– (1996). "A connectionist model of STM for serial order". In: *Models of Short-term Memory*. Ed. by S. E. Gathercole. East Sussex, England: Psychology Press, pp. 51–72.

– (2005). "Computational models of working memory: putting long-term memory into context". In: *Trends in Cognitive Sciences* 9.11, pp. 535–541. DOI: 10.1016/j.tics.2005.09.011.

Butts, D. A. and M. S. Goldman (2006). "Tuning Curves, Neuronal Variability, and Sensory Coding". In: *PLOS Biology* 4.4, e92. DOI: 10.1371/journal.pbio.0040092.

Buzsáki, G. (1989). "Two-stage model of memory trace formation: a role for 'noisy' brain states". In: *Neuroscience* 31.3, pp. 551–570.

Buzsáki, G. and E. I. Moser (2013). "Memory, Navigation and Theta Rhythm in the Hippocampal-Entorhinal System". In: *Nature Neuroscience* 16.2, p. 130. DOI: 10.1038/nn.3304.

Cai, T., J. Fan, and T. Jiang (2013). "Distributions of Angles in Random Packing on Spheres". In: *Journal of Machine Learning Research* 14.1, pp. 1837–1864.

Chaudhuri, R. and I. Fiete (2016). "Computational Principles of Memory". In: *Nature Neuroscience* 19.3, pp. 394–403. DOI: 10.1038/nn.4237.

Choo, X. (2010). "The ordinal serial encoding model: serial memory in spiking neurons". Master Thesis. Waterloo, Ontario, Canada: University of Waterloo. URL: https://uwspace.uwaterloo.ca/handle/10012/5385.

Chrobak, J. J. and G. Buzsáki (1994). "Selective Activation of Deep Layer (V-VI) Retrohippocampal Cortical Neurons during Hippocampal Sharp Waves in the Behaving Rat". In: *Journal of Neuroscience* 14.10, pp. 6160–6170. pmid: 7931570.

Claiborne, B. J., D. G. Amaral, and W. M. Cowan (1986). "A light and electron microscopic analysis of the mossy fibers of the rat dentate gyrus". In: *The Journal of Comparative Neurology* 246.4, pp. 435–458. DOI: 10.1002/cne.902460403.

Cohn, H. et al. (2017). "The Sphere Packing Problem in Dimension 24". In: *Annals of Mathematics* 185.3, pp. 1017–1033. DOI: 10.4007/annals.2017.185.3.8. arXiv: 1603.06518.

Crawford, E., M. Gingerich, and C. Eliasmith (2016). "Biologically Plausible, Human-Scale Knowledge Representation". In: *Cognitive Science* 40.4, pp. 782–821. DOI: 10.1111/cogs.12261.

Davelaar, E. J. (2007). "Sequential Retrieval and Inhibition of Parallel (Re)Activated Representations: A Neurocomputational Comparison of Competitive Queuing and Resampling Models". In: *Adaptive Behavior* 15.1, pp. 51–71. DOI: 10.1177/1059712306076250.

Davelaar, E. J. et al. (2008). "Postscript: Through TCM, STM shines bright." In: *Psychological Review* 115.4, pp. 1116–1118. DOI: 10.1037/0033-295X.115.4.1116.

Duggins, P. (2017). "Incorporating Biologically Realistic Neuron Models into the NEF". Master Thesis. Waterloo, Ontario, Canada: University of Waterloo. URL: https://uwspace.uwaterloo.ca/handle/10012/12393.

Egorov, A. V. et al. (2002). "Graded Persistent Activity in Entorhinal Cortex Neurons". In: *Nature* 420.6912, pp. 173–178. DOI: 10.1038/nature01171.

Eichenbaum, H. (2001). "The Hippocampus and Declarative Memory: Cognitive Mechanisms and Neural Codes". In: *Behavioural Brain Research* 127.1-2, pp. 199–207. DOI: 10.1016/S0166-4328(01)00365-5. pmid: 11718892.

Eliasmith, C. (2013). *How to Build a Brain: A Neural Architecture for Biological Cognition*. New York, NY: Oxford University Press.

Eliasmith, C. and C. H. Anderson (2003). *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. Cambridge, MA: MIT Press.

Eliasmith, C., J. Gosmann, and X. Choo (2016). "BioSpaun: A Large-Scale Behaving Brain Model with Complex Neurons". In: arXiv: 1602.05220 [cs, q-bio].

Eliasmith, C., T. C. Stewart, et al. (2012). "A large-scale model of the functioning brain". In: *Science* 338.6111, pp. 1202–1205. DOI: 10.1126/science.1225266.

Estes, W. K. (1955). "Statistical theory of spontaneous recovery and regression". In: *Psychological Review* 62.3, pp. 145–154. DOI: 10.1037/h0048509.

– (1972). "An associative basis for coding and organiation in memory". In: *Coding Processes in Human Memory*. Ed. by A. W. Melton and E. Martin. The Experimental Psychology Series. Washington, D.C.: V. H. Winston & Sons, pp. 161–190.

Farrell, S. and S. Lewandowsky (2008). "Empirical and Theoretical Limits on Lag Recency in Free Recall". In: *Psychonomic Bulletin & Review* 15.6, pp. 1236–1250. DOI: 10.3758/PBR.15.6.1236.

Folkerts, S., U. Rutishauser, and M. W. Howard (2018). "Human Episodic Memory Retrieval Is Accompanied by a Neural Contiguity Effect". In: *Journal of Neuroscience* 38.17, pp. 4200–4211. DOI: 10.1523/JNEUROSCI.2312-17.2018. pmid: 29615486.

Frank, L. M., E. N. Brown, and M. Wilson (2000). "Trajectory Encoding in the Hippocampus and Entorhinal Cortex". In: *Neuron* 27.1, pp. 169–178. DOI: 10.1016/S0896-6273(00)00018-0.

Freund, T. F. and G. Buzsáki (1996). "Interneurons of the hippocampus". In: *Hippocampus* 6.4, pp. 347–470. DOI: 10.1002/(SICI)1098-1063(1996)6:4<347::AID-HIPO1>3.0.CO;2-I.

Gallistel, C. R. and A. P. King (2009). *Memory and the Computational Brain: Why Cognitive Science will Transform Neuroscience*. 1st ed. Chichester, West Sussex, UK; Malden, MA: Wiley-Blackwell. 336 pp.

Gayler, R. W. (2004). "Vector Symbolic Architectures Answer Jackendoff's Challenges for Cognitive Neuroscience". In: arXiv: cs/0412059.

Ghoneim, M. M. and S. P. Mewaldt (1975). "Effects of Diazepam and Scopolamine on Storage, Retrieval and Organizational Processes in Memory". In: *Psychopharmacologia* 44.3, pp. 257–262. DOI: 10.1007/BF00428903.

Girardeau, G. et al. (2009). "Selective Suppression of Hippocampal Ripples Impairs Spatial Memory". In: *Nature Neuroscience* 12.10, pp. 1222–1223. DOI: 10.1038/nn.2384.

Gold, J. I. and M. N. Shadlen (2007). "The Neural Basis of Decision Making". In: *Annual Review of Neuroscience* 30.1, pp. 535–574. DOI: 10.1146/annurev.neuro.29.051605.113038. pmid: 17600525.

Goldman-Rakic, P. S. (1995). "Cellular Basis of Working Memory". In: *Neuron* 14.3, pp. 477–485. DOI: 10.1016/0896-6273(95)90304-6.

Gosmann, J. (2015). *Precise Multiplications with the NEF*. Waterloo, Ontario, Canada: University of Waterloo. DOI: 10.5281/zenodo.35680.

Gosmann, J. and C. Eliasmith (2015). "A Spiking Neural Model of the N-Back Task". In: *37th Annual Meeting of the Cognitive Science Society*, pp. 812–817.

– (2016). "Optimizing Semantic Pointer Representations for Symbol-Like Processing in Spiking Neural Networks". In: *PLoS ONE* 11.2, e0149928. DOI: 10.1371/journal.pone.0149928.

– (2017). "Automatic Optimization of the Computation Graph in the Nengo Neural Network Simulator". In: *Frontiers in Neuroinformatics* 11. DOI: 10.3389/fninf.2017.00033.

Gosmann, J., A. R. Voelker, and C. Eliasmith (2017). "A Spiking Independent Accumulator Model for Winner-Take-All Computation". In: *Proceedings of the 39th Annual Conference of the Cognitive Science Society*. CogSci 2017. Austin, TX: Cognitive Science Society.

Graf, P., L. R. Squire, and G. Mandler (1984). "The Information That Amnesic Patients Do Not Forget". In: *Journal of Experimental Psychology. Learning, Memory, and Cognition* 10.1, pp. 164–178. pmid: 6242734.

Guzowski, J. F., J. J. Knierim, and E. I. Moser (2004). "Ensemble dynamics of hippocampal regions CA3 and CA1". In: *Neuron* 44.4, pp. 581–584. DOI: 10.1016/j.neuron.2004.11.003.

Hafting, T. et al. (2005). "Microstructure of a Spatial Map in the Entorhinal Cortex". In: *Nature* 436.7052, pp. 801–806. DOI: 10.1038/nature03721.

Harman, R. and V. Lacko (2010). "On Decompositional Algorithms for Uniform Sampling from -Spheres and -Balls". In: *Journal of Multivariate Analysis* 101.10, pp. 2297–2304. DOI: 10.1016/j.jmva.2010.06.002.

Hasselmo, M. E. (2012). *How We Remember: Brain Mechanisms of Episodic Memory*. Cambridge, MA: MIT Press. 383 pp.

Hasselmo, M. E. and B. P. Wyble (1997). "Free Recall and Recognition in a Network Model of the Hippocampus: Simulating Effects of Scopolamine on Human Memory Function". In: *Behavioural Brain Research* 89.1–2, pp. 1–34. DOI: 10.1016/S0166-4328(97)00048-X.

Hebb, D. O. (1961). "Distinctive features of learning in the higher animal". In: *Brain mechanisms and learning*. Ed. by J. F. Delafresnaye. Oxford, England: Blackwell, pp. 37–46.

Henson, R. N. A. (1996). "Short-term memory for serial order". unpublished. Dissertation. Cambridge, England: University of Cambridge.

Hintzman, D. L. (1988). "Judgments of frequency and recognition memory in a multiple-trace memory model". In: *Psychological Review* 95.4, pp. 528–551. DOI: 10.1037/0033-295X.95.4.528.

Hirotsu, I. et al. (1989). "Effect of Anticholinergic Drug on Long-Term Potentiation in Rat Hippocampal Slices". In: *Brain Research* 482.1, pp. 194–197. DOI: 10.1016/0006-8993(89)90561-1.

Horwitz, B. and J. F. Smith (2008). "A Link Between Neuroscience and Informatics: Large-Scale Modeling of Memory Processes". In: *Methods* 44.4, pp. 338–347. DOI: 10.1016/j.ymeth.2007.02.007. pmid: 18374277.

Howard, M. W., M. S. Fotedar, et al. (2005). "The Temporal Context Model in Spatial Navigation and Relational Learning: Toward a Common Explanation of Medial Temporal Lobe Function across Domains". In: *Psychological Review* 112.1, pp. 75–116. DOI: 10.1037/0033-295X.112.1.75.

Howard, M. W. and M. J. Kahana (1999). "Contextual variability and serial position effects in free recall". In: *Journal of Experimental Psychology: Learning, Memory, and Cognition* 25.4, pp. 923–941. DOI: 10.1037/0278-7393.25.4.923.

– (2002). "A distributed representation of temporal context". In: *Journal of Mathematical Psychology* 46.3, pp. 269–299. DOI: 10.1006/jmps.2001.1388.

Howard, M. W., C. J. MacDonald, et al. (2014). "A Unified Mathematical Framework for Coding Time, Space, and Sequences in the Hippocampal Region". In: *The Journal of Neuroscience* 34.13, pp. 4692–4707.

Howard, M. W., P. B. Sederberg, and M. J. Kahana (2009). "Reply to Farrell and Lewandowsky: Recency-Contiguity Interactions Predicted by the Temporal

Context Model". In: *Psychonomic Bulletin & Review* 16.5, pp. 973–984. pmid: 19927395.

Howard, M. W., K. H. Shankar, et al. (2015). "A Distributed Representation of Internal Time." In: *Psychological Review* 122.1, p. 24. DOI: 10.1037/a0037840.

Huilme, S. R. et al. (2014). "Mechanisms of Heterosynaptic Metaplasticity". In: *Philosophical Transactions of the Royal Society* 369. DOI: 10.1098/rstb.2013.0148.

Ison, M. J., R. Quian Quiroga, and I. Fried (2015). "Rapid Encoding of New Memories by Individual Neurons in the Human Brain". In: *Neuron* 87.1, pp. 220–230. DOI: 10.1016/j.neuron.2015.06.016. pmid: 26139375.

Ito, T., Y. Miura, and T. Kadokawa (1988). "Effects of Physostigmine and Scopolamine on Long-Term Potentiation of Hippocampal Population Spikes in Rats". In: *Canadian Journal of Physiology and Pharmacology* 66.8, pp. 1010–1016. pmid: 3179833.

Jahnke, J. C. (1968). "Delayed recall and the serial-position effect of short-term memory". In: *Journal of Experimental Psychology* 76.4, pp. 618–622.

Jonas, P., G. Major, and B. Sakmann (1993). "Quantal Components of Unitary EPSCs at the Mossy Fibre Synapse on CA3 Pyramidal Cells of Rat Hippocampus." In: *The Journal of Physiology* 472.1, pp. 615–663. DOI: 10.1113/jphysiol.1993.sp019965.

Jung, M. W. and B. L. McNaughton (1993). "Spatial Selectivity of Unit Activity in the Hippocampal Granular Layer". In: *Hippocampus* 3.2, pp. 165–182. DOI: 10.1002/hipo.450030209.

Kajic, I. et al. (2017). "A Spiking Neuron Model of Word Associations for the Remote Associates Test". In: *Frontiers in Psychology* 8.99. DOI: 10.3389/fpsyg.2017.00099.

Koch, C. (2004). *Biophysics of Computation: Information Processing in Single Neurons*. Oxford University Press, USA. 588 pp.

Komer, B. (2015). "Biologically Inspired Adaptive Control of Quadcopter Flight". Masters Thesis. Waterloo, ON: University of Waterloo. 65 pp. URL: https://uwspace.uwaterloo.ca/handle/10012/9549.

Kumaran, D., D. Hassabis, and J. L. McClelland (2016). "What Learning Systems Do Intelligent Agents Need? Complementary Learning Systems Theory Updated". In: *Trends in Cognitive Sciences* 20.7, pp. 512–534. DOI: 10.1016/j.tics.2016.05.004.

Latimer, K. W. et al. (2015). "Single-Trial Spike Trains in Parietal Cortex Reveal Discrete Steps during Decision-Making". In: *Science* 349.6244, pp. 184–187. DOI: 10.1126/science.aaa4056. pmid: 26160947.

Lega, B. C., J. Jacobs, and M. J. Kahana (2012). "Human Hippocampal Theta Oscillations and the Formation of Episodic Memories". In: *Hippocampus* 22.4, pp. 748–761. DOI: 10.1002/hipo.20937. pmid: 21538660.

Leung, L. S. et al. (2003). "Cholinergic Activity Enhances Hippocampal Long-Term Potentiation in CA1 during Walking in Rats". In: *The Journal of Neuroscience: The Official Journal of the Society for Neuroscience* 23.28, pp. 9297–9304. pmid: 14561856.

Leutgeb, S. and J. K. Leutgeb (2007). "Pattern separation, pattern completion, and new neuronal codes within a continuous CA3 map". In: *Learning & Memory* 14.11, pp. 745–757. DOI: 10.1101/lm.703907. pmid: 18007018.

Levy, W. B., A. B. Hocking, and X. Wu (2005). "Interpreting hippocampal function as recoding and forecasting". In: *Neural Networks*. Computational Theories of the Functions of the Hippocampus 18.9, pp. 1242–1264. DOI: 10.1016/j.neunet.2005.08.005.

Lillicrap, T. P. et al. (2016). "Random Synaptic Feedback Weights Support Error Backpropagation for Deep Learning". In: *Nature Communications* 7.13276. DOI: 10.1038/ncomms13276.

Markram, H. et al. (2015). "Reconstruction and Simulation of Neocortical Microcircuitry". In: *Cell* 163.2, pp. 456–492. DOI: 10.1016/j.cell.2015.09.029.

McClelland, J. L., B. L. McNaughton, and R. C. O'Reilly (1995). "Why There Are Complementary Learning Systems in the Hippocampus and Neocortex: Insights from the Successes and Failures of Connectionist Models of Learning and Memory." In: *Psychological Review* 102.3, pp. 419–457. DOI: 10.1037/0033-295X.102.3.419.

Milford, M., G. Wyeth, and D. Prasser (2004). "RatSLAM: a hippocampal model for simultaneous localization and mapping". In: IEEE International Conference on Robotics and Automation. Vol. 1, pp. 403–408. DOI: 10.1109/ROBOT.2004.1307183.

Milinski, M. and C. Wedekind (1998). "Working Memory Constrains Human Cooperation in the Prisoner's Dilemma". In: *Proceedings of the National Academy of Sciences* 95.23, pp. 13755–13758. DOI: 10.1073/pnas.95.23.13755. pmid: 9811873.

Moreno-Bote, R. and N. Parga (2005). "Simple Model Neurons with AMPA and NMDA Filters: Role of Synaptic Time Scales". In: *Neurocomputing*. Computational Neuroscience: Trends in Research 2005 65-66, pp. 441–448. DOI: 10.1016/j.neucom.2004.10.016.

Murdock, B. B. (1993). "TODAM2: A model for the storage and retrieval of item, associative, and serial-order information". In: *Psychological Review* 100.2, pp. 183–203. DOI: 10.1037/0033-295X.100.2.183.

– (1997). "Context and mediators in a theory of distributed associative memory (TODAM2)". In: *Psychological Review* 104.4, pp. 839–862. DOI: 10.1037/0033-295X.104.4.839.

Ninokura, Y., H. Mushiake, and J. Tanji (2003). "Representation of the Temporal Order of Visual Objects in the Primate Lateral Prefrontal Cortex". In: *Journal of Neurophysiology* 89.5, pp. 2868–2873. DOI: 10.1152/jn.00647.2002.

Norman, K. A. and R. C. O'Reilly (2003). "Modeling hippocampal and neocortical contributions to recognition memory: A complementary-learning-systems approach." In: *Psychological Review* 110.4, pp. 611–646. DOI: 10.1037/0033-295X.110.4.611.

Nowak, M. A. and N. L. Komarova (2001). "Towards an Evolutionary Theory of Language". In: *Trends in Cognitive Sciences* 5.7, pp. 288–295. DOI: 10.1016/S1364-6613(00)01683-1.

O'Keefe, J. and M. L. Recce (1993). "Phase Relationship between Hippocampal Place Units and the EEG Theta Rhythm". In: *Hippocampus* 3.3, pp. 317–330. DOI: 10.1002/hipo.450030307. pmid: 8353611.

O'Reilly, R. C. and M. J. Frank (2006). "Making Working Memory Work: A Computational Model of Learning in the Prefrontal Cortex and Basal Ganglia". In: *Neural Computation* 18.2, pp. 283–328. DOI: 10.1162/089976606775093909.

Oliver Trujillo (2014). "A spiking neural model of episodic memory encoding and replay in hippocampus". Master Thesis. Waterloo, Ontario, Canada: University of Waterloo. URL: https://uwspace.uwaterloo.ca/handle/10012/8970.

Owen, A. M. (1997). "The Functional Organization of Working Memory Processes Within Human Lateral Frontal Cortex: The Contribution of Functional Neuroimaging". In: *European Journal of Neuroscience* 9.7, pp. 1329–1339. DOI: 10.1111/j.1460-9568.1997.tb01487.x.

Page, M. P. A. and D. Norris (1998). "The primacy model: A new model of immediate serial recall". In: *Psychological Review* 105.4, pp. 761–781. DOI: 10.1037/0033-295X.105.4.761-781.

Paxinos, G., ed. (2014). *The Rat Nervous System*. 3rd ed. Amsterdam; Boston: Elsevier Academic Press. 1053 pp.

Penfield, W. and B. Milner (1958). "Memory Deficit Produced by Bilateral Lesions in the Hippocampal Zone". In: *Archives of Neurology And Psychiatry* 79.5, pp. 475–497. DOI: 10.1001/archneurpsyc.1958.02340050003001.

Piatti, V. C., L. A. Ewell, and J. K. Leutgeb (2013). "Neurogenesis in the Dentate Gyrus: Carrying the Message or Dictating the Tone". In: *Frontiers in Neuroscience* 7. DOI: 10.3389/fnins.2013.00050. pmid: 23576950.

Plate, T. A. (1995). "Holographic Reduced Representations". In: *IEEE Transactions on Neural Networks* 6.3, pp. 623–641.

– (2003). *Holographic Reduced Representation: Distributed Representation for Cognitive Structures*. Stanford, CA: CSLI Publications. 300 pp.

Prebble, S. C., D. R. Addis, and L. J. Tippett (2013). "Autobiographical Memory and Sense of Self". In: *Psychological Bulletin* 139.4, pp. 815–840. DOI: 10.1037/a0030146. pmid: 23025923.

Quirk, G. J. et al. (1992). "The Positional Firing Properties of Medial Entorhinal Neurons: Description and Comparison with Hippocampal Place Cells". In: *The Journal of Neuroscience: The Official Journal of the Society for Neuroscience* 12.5, pp. 1945–1963. pmid: 1578279.

Raaijmakers, J. G. and R. M. Shiffrin (1981). "Search of associative memory". In: *Psychological Review* 88.2, pp. 93–134. DOI: 10.1037/0033-295X.88.2.93.

Rasmussen, D., A. R. Voelker, and C. Eliasmith (2017). "A Neural Model of Hierarchical Reinforcement Learning". In: *PLoS ONE* 12.7, pp. 1–39. DOI: 10.1371/journal.pone.0180234.

Rebola, N., M. Carta, and C. Mulle (2017). "Operation and Plasticity of Hippocampal CA3 Circuits: Implications for Memory Encoding". In: *Nature Reviews Neuroscience* 18.4, pp. 208–220. DOI: 10.1038/nrn.2017.10.

Recchia, G. et al. (2015). "Encoding Sequential Information in Semantic Space Models: Comparing Holographic Reduced Representation and Random Permutation". In: *Computational Intelligence and Neuroscience* 2015. DOI: 10.1155/2015/986574.

Robins, S. (2015). "A mechanism for mental time travel? A critical review of Hasselmo's How we remember: Brain mechanisms of episodic memory". In: *Philosophical Psychology* 28.6, pp. 903–915. DOI: 10.1080/09515089.2013.877243.

Robinson, E. S. and M. A. Brown (1926). "Effect of serial position upon memorization". In: *The American Journal of Psychology* 37.4, pp. 538–552. DOI: 10.2307/1414914. JSTOR: 1414914.

Rolls, E. T. (2013). "The mechanisms for pattern completion and pattern separation in the hippocampus". In: *Frontiers in Systems Neuroscience* 7.74. DOI: 10.3389/fnsys.2013.00074. pmid: 24198767.

Rypma, B. et al. (1999). "Load-Dependent Roles of Frontal Brain Regions in the Maintenance of Working Memory". In: *NeuroImage* 9.2, pp. 216–226. DOI: 10.1006/nimg.1998.0404.

Sah, P., S. Hestrin, and R. A. Nicoll (1990). "Properties of Excitatory Postsynaptic Currents Recorded in Vitro from Rat Hippocampal Interneurones." In: *The Journal of Physiology* 430.1, pp. 605–616. DOI: 10.1113/jphysiol.1990.sp018310.

Scoville, W. B. and B. Milner (1957). "Loss of Recent Memory after Bilateral Hippocampal Lesions". In: *Journal of Neurology, Neurosurgery, and Psychiatry* 20.1, pp. 11–21. pmid: 13406589.

Sederberg, P. B., M. W. Howard, and M. J. Kahana (2008). "A context-based theory of recency and contiguity in free recall". In: *Psychological Review* 115.4, pp. 893–912. DOI: 10.1037/a0013396.

Shankar, K. H. and M. W. Howard (2013). "Optimally Fuzzy Temporal Memory." In: *Journal of Machine Learning Research* 14.1, pp. 3785–3812.

Sharma, S., S. Aubin, and C. Eliasmith (2016). "Large-Scale Cognitive Model Design Using the Nengo Neural Simulator". In: *Biologically Inspired Cognitive Architectures* 17, pp. 86–100. DOI: 10.1016/j.bica.2016.05.001.

Shiffrin, R. M. and M. Steyvers (1997). "A model for recognition memory: REM—retrieving effectively from memory". In: *Psychonomic Bulletin & Review* 4.2, pp. 145–166. DOI: 10.3758/BF03209391.

Smith, P. L. and R. Ratcliff (2004). "Psychology and Neurobiology of Simple Decisions". In: *Trends in Neurosciences* 27.3, pp. 161–168. DOI: 10.1016/j.tins.2004.01.006.

Smolensky, P. (1990). "Tensor Product Variable Binding and the Representation of Symbolic Structures in Connectionist Systems". In: *Artificial Intelligence* 46.1, pp. 159–216. DOI: 10.1016/0004-3702(90)90007-M.

Solso, R. L. (1998). *Cognitive Psychology*. 5th ed. Needham Heights, MA: Allyn and Bacon. 632 pp.

Spruston, N., P. Jonas, and B. Sakmann (1995). "Dendritic Glutamate Receptor Channels in Rat Hippocampal CA3 and CA1 Pyramidal Neurons." In: *The Journal of Physiology* 482.2, pp. 325–352. DOI: 10.1113/jphysiol.1995.sp020521.

Squire, L. R. (2009). "The Legacy of Patient H.M. for Neuroscience". In: *Neuron* 61.1, pp. 6–9. DOI: 10.1016/j.neuron.2008.12.023.

Squire, L. R., A. P. Shimamura, and D. G. Amaral (1989). "Memory and the hippocampus". In: *Neural Models of Plasticity: Experimental and Theoretical Approaches*. Ed. by J. H. Byrne and W. O. Berry. San Diego, CA: Academic Press.

Stadler, M. A. (1993). "Implicit serial learning: Questions inspired by Hebb (1961)". In: *Memory & Cognition* 21.6, pp. 819–827. DOI: 10.3758/BF03202749.

Stewart, T. C. and C. Eliasmith (2011). "Neural Cognitive Modelling: A Biologically Constrained Spiking Neuron Model of the Tower of Hanoi Task". In: *Proceedings of the 33rd Annual Meeting of the Cogni6ve Science Society*. CogSci 2011. Austin, TX: Cognitive Science Society, pp. 656–661.

Stewart, T. C., Y. Tang, and C. Eliasmith (2011). "A Biologically Realistic Cleanup Memory: Autoassociation in Spiking Neurons". In: *Cognitive Systems Research* 12.2, pp. 84–92. DOI: 10.1016/j.cogsys.2010.06.006.

Stickgold, R. (2013). "Parsing the Role of Sleep in Memory Processing". In: *Current opinion in neurobiology* 23.5, pp. 847–853. DOI: 10.1016/j.conb.2013.04.002. pmid: 23618558.

Stöckel, A., A. R. Voelker, and C. Eliasmith (2017). "Point Neurons with Conductance-Based Synapses in the Neural Engineering Framework". In: arXiv: 1710.07659 [cs, q-bio].

– (2018). "Nonlinear Synaptic Interaction as a Computational Resource in the Neural Engineering Framework". In: *Cosyne Abstracts*. Cosyne. Denver, CO.

Suzuki, W. A., E. K. Miller, and R. Desimone (1997). "Object and Place Memory in the Macaque Entorhinal Cortex". In: *Journal of Neurophysiology* 78.2, pp. 1062–1081. pmid: 9307135.

Talmi, D. et al. (2005). "Neuroimaging the Serial Position Curve. A Test of Single-Store versus Dual-Store Models". In: *Psychological Science* 16.9, pp. 716–723. DOI: 10.1111/j.1467-9280.2005.01601.x. pmid: 16137258.

Todd, J. J. and R. Marois (2004). "Capacity Limit of Visual Short-Term Memory in Human Posterior Parietal Cortex". In: *Nature* 428.6984, p. 751. DOI: 10.1038/nature02466.

Tort, A. B. L. et al. (2009). "Theta–Gamma Coupling Increases during the Learning of Item–Context Associations". In: *Proceedings of the National Academy of Sciences* 106.49, pp. 20942–20947. DOI: 10.1073/pnas.0911331106. pmid: 19934062.

Uchida, T., S. Fukuda, and H. Kamiya (2012). "Heterosynaptic Enhancement of the Excitability of Hippocampal Mossy Fibers by Long-Range Spill-over of Glutamate". In: *Hippocampus* 22.2, pp. 222–229. DOI: 10.1002/hipo.20885.

Usher, M. and J. L. McClelland (2001). "The time course of perceptual choice: The leaky, competing accumulator model". In: *Psychological Review* 108.3, pp. 550–592. DOI: 10.1037/0033-295X.108.3.550.

Voelker, A. R., B. V. Benjamin, et al. (2017). "Extending the Neural Engineering Framework for Nonideal Silicon Synapses". In: *IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. Baltimore, MD.

Voelker, A. R., J. Gosmann, and T. C. Stewart (2017). *Efficiently Sampling Vectors and Coordinates from the N-Sphere and n-Ball*. Waterloo, ON: Centre for Theoretical Neuroscience. DOI: 10.13140/RG.2.2.15829.01767/1.

Vorhees, C. V. and M. T. Williams (2014). "Assessing Spatial Learning and Memory in Rodents". In: *ILAR Journal* 55.2, pp. 310–332. DOI: 10.1093/ilar/ilu013. pmid: 25225309.

Vu Kim Tuan and Dinh Thanh Duc (2000). "Convergence Rate of Post-Widder Approximate Inversion of the Laplace Transform". In: *Vietnam Journal of Mathematics* 28.1, pp. 93–96.

Wagner, U. et al. (2004). "Sleep Inspires Insight". In: *Nature* 427.6972, pp. 352–355. DOI: 10.1038/nature02223.

West, M. J. and L. Slomianka (1998). "Total Number of Neurons in the Layers of the Human Entorhinal Cortex". In: *Hippocampus* 8.1, pp. 69–82. DOI: 10.1002/(SICI)1098-1063(1998)8:1<69::AID-HIPO7>3.0.CO;2-2. pmid: 9519888.

Witter, M. P. (2010). "Connectivity of the Hippocampus". In: *Hippocampal Microcircuits*. Ed. by V. Cutsuridis et al. Springer Series in Computational Neuroscience 5. Springer New York, pp. 5–26. DOI: 10.1007/978-1-4419-0996-1_1.

Wyner, A. D. (1967). "Random Packings and Coverings of the Unit N-Sphere". In: *The Bell System Technical Journal* 46.9, pp. 2111–2118. DOI: 10.1002/j.1538-7305.1967.tb04246.x.

Yntema, D. B. and F. P. Trask (1963). "Recall as a search process". In: *Journal of Verbal Learning and Verbal Behavior* 2.1, pp. 65–74. DOI: 10.1016/S0022-5371(63)80069-9.

Young, B. J. et al. (1997). "Memory Representation within the Parahippocampal Region". In: *The Journal of Neuroscience: The Official Journal of the Society for Neuroscience* 17.13, pp. 5183–5195. pmid: 9185556.

Yu, Q. et al. (2017). "A Hierarchically Organized Memory Model with Temporal Population Coding". In: *Neuromorphic Cognitive Systems*. Intelligent Systems Reference Library 126. Springer International Publishing, pp. 131–152. DOI: 10.1007/978-3-319-55310-8_7.

Zelano, C. et al. (2009). "A Specialized Odor Memory Buffer in Primary Olfactory Cortex". In: *PLOS ONE* 4.3, e4965. DOI: 10.1371/journal.pone.0004965.

Appendices

A. Derivation of the cosine similarity distribution

Proof. A straight-forward way to derive the cosine similarity distribution is to use a result by Cai, Fan, and Jiang (2013). Theorem 1 states that the probability density of the angles θ between independent d-dimensional vectors is given by

$$h(\theta) = K_d \cdot (\sin\theta)^{d-2}, \quad \theta \in [0, \pi] \tag{A.1}$$

with

$$K_d = \frac{1}{\sqrt{\pi}} \frac{\Gamma\left(\frac{d}{2}\right)}{\Gamma\left(\frac{d-1}{2}\right)} = \frac{\Gamma\left(\frac{d}{2}\right)}{\Gamma\left(\frac{1}{2}\right)\Gamma\left(\frac{d-1}{2}\right)} = \frac{1}{B\left(\frac{1}{2}, \frac{d-1}{2}\right)}. \tag{A.2}$$

To obtain the cosine similarity distribution a change of variables with $\theta = \arccos x,\ x \in [-1, 1]$ has to be performed:

$$\begin{aligned} p_{\mathcal{CS}}(x; d) &= \left|\tfrac{\mathrm{d}}{\mathrm{d}x}\,(\arccos x)\right| \cdot h(\arccos x) && \text{(A.3)} \\ &= \frac{1}{\sqrt{1-x^2}} \cdot K_d \cdot (\sin\arccos x)^{d-2} && \text{(A.4)} \\ &= \frac{1}{\sqrt{1-x^2}} \cdot K_d \cdot \left(\sqrt{1-x^2}\right)^{d-2} && \text{(A.5)} \\ &= K_d \cdot \left(1-x^2\right)^{(d-3)/2}. && \text{(A.6)} \end{aligned}$$

This matches (6.9). □

The cosine similarity distribution can also be obtained as a special case of a more general distribution of the ℓ^2-norm of m components of an $n + m$-dimensional unit vector. The PDF of this distribution is given by Harman and Lacko (2010) and Gosmann and Eliasmith (2016)

$$p_{\mathcal{SB}}(x; n, m) = \frac{2}{B\left(\frac{n}{2}, \frac{m}{2}\right)} \left(x^2\right)^{(m-1)/2} \left(1-x^2\right)^{n/2-1}, \quad x \in [-1, 1]. \tag{A.7}$$

When determining the cosine similarity of two uniformly distributed random vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, we are free to chose any set of basis vectors without loss of generality. Let us chose the basis such that $\boldsymbol{a} = (a_1, 0, 0, \dots)^\top$ is aligned with the first vector of the standard basis. The cosine similarity then becomes

$$\cos(\boldsymbol{a} \angle \boldsymbol{b}) = \frac{\langle \boldsymbol{a}, \boldsymbol{b} \rangle}{\|\boldsymbol{a}\| \cdot \|\boldsymbol{b}\|} = \frac{a_1 \cdot b_1}{a_1 \cdot \|\boldsymbol{b}\|} = \frac{b_1}{\|\boldsymbol{b}\|}. \tag{A.8}$$

The absolute value of this is equal to the length of a one-component subvector of the unit-vector $\boldsymbol{b}/\|\boldsymbol{b}\|$. Thus, we can use (A.7) and divide it by two to account for the symmetry of the absolute value to determine the PDF of the cosine similarity as

$$p_{\mathcal{CS}}(x; d) = \frac{1}{2} p_{\mathcal{SB}}(x; d-1, 1) = \frac{1}{B\left(\frac{1}{2}, \frac{d-1}{2}\right)} \left(1 - x^2\right)^{(d-3)/2}. \tag{A.9}$$

B. Comparisons of the uniform and cosine similarity intercept distributions

Table B.1 summarizes the change in error over a wide range of parameters when switching from a uniform intercept distribution to the $\mathcal{CS}(d+2)$ distribution. In general, the cosine similarity distribution performs better. It performs equal to the uniform distribution for $d = 1$ in which case $\mathcal{CS}(d+2)$ reduces to a uniform distribution and rectified linear (rate) neurons with a regularization of $\lambda = 0.1$. The cosine similarity distribution performs slightly worse for rate neurons (LIF rate and rectified linear) when the regularization is adjusted to $\lambda = 0.01$ to account for the non-existent spiking noise. The only other case with slightly worse performance is when computing pairwise products $y_i = x_{2i-2}x_{2i-1}$, $1 \leq i \leq i/2$, but note that no further optimization for this sort of function has been done and the high dimensionality makes this a hard function to compute. On the simpler squaring, the cosine similarity distribution performs better.

Table B.1.: Change in representational error in the NEF when switching from uniformly distributed intercepts to $\mathcal{CS}(d+2)$ distributed intercepts for different dimensionalities d, neuron numbers n, synaptic time constants τ_{syn}, decoded functions, regularization λ, and neuron types. A negative change in error (highlighted red) means that the cosine similarity distribution performed better. Statistical significance, determined with bootstrapping, is marked with **** for $p < 0.0001$ and * for $p < 0.05$.

							Change in	
d	n/d	τ_{syn}/s	Function	λ	Neuron type	$\langle E_{\text{d}} \rangle$	$\langle E_{\text{n}} \rangle$	$\langle E_{\text{tot}} \rangle$
1	50	0.005	x	0.100	LIF	0.0000	0.0009	0.0010
2						0.0007	−0.0050****	−0.0043****
4						−0.0020	−0.0139****	−0.0137****
8						0.0037	−0.0318****	−0.0268****
64	10					−0.1239****	−0.2131****	−0.2458****
	25					−0.0438****	−0.1743****	−0.1778****
	50			0.005		0.0347****	−0.2836****	−0.2790****
				0.010	Adaptive LIF	0.0821****	−0.2249****	−0.1790****
					LIF	0.0262****	−0.2191****	−0.2142****
					LIF Rate	0.0262****	0.0000*	0.0262****
					Rectified Linear	0.0307****	0.0000	0.0307****
				0.100	Adaptive LIF	0.0311****	−0.1315****	−0.0815****
					LIF	−0.0200****	−0.1313****	−0.1304****
					LIF Rate	−0.0200****	0.0000	−0.0200****
					Rectified Linear	0.0022	0.0000	0.0022
				0.200	LIF	−0.0873****	−0.1004****	−0.1320****
			x^2	0.100		−0.1030****	0.0229****	−0.1016****
			$x_{2i-2}x_{2i-1}$			0.0084****	0.0115****	0.0142****
		0.100	x			−0.0200****	−0.0066****	−0.0207****

C. Leaky, Competing Accumulator Model Analysis

The model dynamics for the leaky, competing accumulator (LCA) are given as follows:

$$\frac{\partial x_i}{\partial t} = \left(u_i - \kappa x_i - \lambda \sum_{j \neq i} x_j \right) \frac{1}{\tau}, \quad x_i \geq 0. \tag{C.1}$$

We now justify our choice of $\kappa = \lambda = 1$ from the text. This guarantees that the winning state will converge to its input value, and each losing state will converge to zero.

C.1. Effect of $\kappa = 1$

To isolate the 'role' of κ, we consider the case where $\lambda \sum_{j \neq i} x_j = 0$. This situation obeys the dynamics:

$$\frac{\partial x_i}{\partial t} = (u_i - \kappa x_i) \frac{1}{\tau}. \tag{C.2}$$

We then take the Laplace transform of both sides, and rearrange to obtain:

$$s\mathcal{L}\{x_i\} = (\mathcal{L}\{u_i\} - \kappa \mathcal{L}\{x_i\}) \frac{1}{\tau} \quad \iff \quad \frac{\mathcal{L}\{x_i\}}{\mathcal{L}\{u_i\}} = \frac{1}{\tau s + \kappa}. \tag{C.3}$$

When $\kappa = 1$, this is commonly referred to as a first-order lowpass filter with time-constant τ. The effect of this filter in the time-domain is a convolution with the exponentially decaying function $h(t) := \tau^{-1} \exp(-t/\tau)$. More generally, when $\kappa > 0$, this has the transfer function:

$$\frac{1}{\tau s + \kappa} = \frac{\kappa^{-1}}{(\tau \kappa^{-1}) s + 1} \tag{C.4}$$

which is the same lowpass with an effective time-constant of $\left(\tau\kappa^{-1}\right)$ and a multiplicative gain of κ^{-1}. In plain words, different values of $\kappa > 0$ do nothing but alter the effective time-constant and the effective gain on the input.

Therefore, by setting $\kappa = 1$, we have $x_i = u_i * h$ provided that $\lambda \sum_{j \neq i} x_j = 0$. Below we prove that the latter assumption eventually holds whenever i is the index of the winner and $\lambda = 1$, and so the winner x_i will converge to the value of its input u_i.

C.2. Effect of $\lambda = 1$

Setting $\lambda = \kappa = 1$, and defining $\chi := \sum_j x_j$, we rewrite the dynamics as:

$$\frac{\partial x_i}{\partial t} = (u_i - \chi)\,\frac{1}{\tau}, \quad x_i \geq 0. \tag{C.5}$$

We conceptualize χ as a single "meta state-variable" that, based on the value of each u_i, will change each corresponding x_i. For instance, if i is the index of the winner, then x_i is necessarily only stable when $\chi = u_i \iff \frac{\partial x_i}{\partial t} = 0$. Assuming stability, $\chi = u_i > u_j$ for all $j \neq i$, and thus each x_j necessarily has a negative derivative. Consequently, all losers x_j will decrease to zero and stabilize there due to rectification (and then $\chi = u_i = x_i$ holds as anticipated).

We also remark that if $\lambda \neq 1$ then the derivatives are no longer sorted by their inputs, and in fact the order will depend on the current state (for instance, it will depend on the previous winner if the inputs were just altered). Consequently, if $\lambda < 1$, then not all losers will necessarily go to zero, and if $\lambda > 1$, then the previous winner may persist.

D. Fuzzy temporal memory

Here, I derive relations used in the analysis of the optimal fuzzy temporal memory (Shankar and Howard 2013) used in the main text.

D.1. Discretized derivative

The first discretized derivative of $\boldsymbol{c}$ with regard to s at s_i is given by (Shankar and Howard 2013)

$$\begin{aligned} c_i^{(1)} &= \frac{c_{i+1} - c_i}{s_{i+1} - s_i} \cdot \frac{s_i - s_{i-1}}{s_{i+1} - s_{i-1}} + \frac{c_i - c_{i-1}}{s_i - s_{i-1}} \cdot \frac{s_{i+1} - s_i}{s_{i+1} - s_{i-1}} \\ &= \frac{1}{K_c}\Big[c_{i-1} \left(s_i - s_{i+1}\right)^2 + c_i \Big(\left(s_i - s_{i-1}\right)^2 - \left(s_i - s_{i+1}\right)^2 \Big) \\ &\quad - c_{i+1} \left(s_i - s_{i-1}\right)^2 \Big] \end{aligned} \tag{D.1}$$

with

$$K_c = s_i^2 s_{i+1} - s_i^2 s_{i-1} - s_i s_{i+1}^2 + s_i s_{i-1}^2 + s_{i+1}^2 s_{i-1} - s_{i+1} s_{i-1}^2.$$

Lemma 1. *When using the optimal fuzzy temporal memory spacing of s_i given by $t^*_{i+1} = (1+\nu) t^*_i \Leftrightarrow s_i = (1+\nu) s_{i+1}$ with $\nu > 0$ and $s_i > s_{i+1}$, we have $s_i - s_{i+1} < s_{i-1} - s_i$.*

Proof.

$$\begin{aligned} & s_i - s_{i+1} \overset{?}{<} s_{i-1} - s_i \\ \Leftrightarrow \quad & (1+\nu) s_{i+1} - s_{i+1} \overset{?}{<} (1+\nu)^2 s_{i+1} - (1+\nu) s_{i+1} \\ \Leftrightarrow \quad & 2 + 2\nu - 1 - 1 - 2\nu - \nu^2 \overset{?}{<} 0 \\ \Leftrightarrow \quad & -\nu^2 < 0 \end{aligned}$$

□

It follows from Lemma 1 that

$$(s_i - s_{i-1})^2 > (s_i - s_{i-1})^2 - (s_i - s_{i+1})^2 \tag{D.2}$$

and

$$(s_i - s_{i-1})^2 > (s_i - s_{i+1})^2 . \tag{D.3}$$

This gives the c_{i+1} coefficient corresponding, to $(s_i - s_{i-1})^2$, the largest contribution in the discrete derivative. Repeated application to obtain a higher-order k-th derivative will yield c_{i+k} with the largest coefficient accordingly. Thus, the k-th derivative for c_i with respect to s will be dominated by the timescale t^*_{i+k}.

D.2. Required neurons

To achieve a lower timescale of τ_1 an appropriate ν for spacing t^*_i can be obtained with (13.6) as

$$-(1+\nu)^k t^*_1 \leq \tau_1 \quad \Leftrightarrow \quad \nu \leq \sqrt[k]{-\tau_1/t^*_1} - 1. \tag{D.4}$$

By spacing M nodes, the earliest reconstructable point $t^*_{\max}$ is given by

$$t^*_{\max} = (1+\nu)^M t^*_1 \tag{D.5}$$

from which the minimum M for given $t^*_{\max}$ follows as

$$M \geq \frac{\log(t^*_{\max}/t^*_1)}{\log(1+\nu)}. \tag{D.6}$$

To be able to construct the k-th discrete derivative an additional $2k$ nodes for a total of $M + 2k$ nodes are required. The noise in the output of the leaky accumulators is amplified by $g_{\eta,\mathrm{NEF}}$. As additional neurons will diminish noise by $O(1/\sqrt{n})$ (Section 6.2), the number of neurons needs to be scaled by $g^2_{\eta,\mathrm{NEF}}$ to achieve the same noise level that would be present without the noise amplification. Multiplying these factors together with a base number of neurons N used to represent a single dimension and the number of dimensions d yields (13.7):

$$N_{\mathrm{tot}}(t^*_1, k) = N d\, g^2_{\eta,\mathrm{NEF}}(\nu, k)\,(M + 2k)\,.$$