In Search of Lost Time

A computational model of the representations and dynamics of episodic memory

by

Yan Wu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2012

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

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

Abstract

In Marcel Proust's most famous novel, *In Search of Lost Time*, a Madeleine cake elicited in him a nostalgic memory of Combray. Here we present a computational hypothesis of how such an episodic memory is represented in a brain area called the hippocampus, and how the dynamics of the hippocampus allow the storage and recall of such past events. Using the Neural Engineering Framework (NEF), we show how different aspects of an event, after compression, are represented together by hippocampal neurons as a vector in a high dimensional memory space. Single neuron simulation results using this representation scheme match well with the observation that hippocampal neurons are tuned to both spatial and non-spatial inputs. We then show that sequences of events represented by high dimensional vectors can be stored as episodic memories in a recurrent neural network (RNN) which is structurally similar to the hippocampus. We use a state-of-the-art Hessian-Free optimization algorithm to efficiently train this RNN. At the behavioural level we also show that, consistent with T-maze experiments on rodents, the storage and retrieval of past experiences facilitate subsequent decision-making tasks.

Acknowledgments

First of all, I would like to thank my supervisors, Chris Eliasmith and Matthijs van der Meer, whose enthusiasm and wisdom make my studies a enjoyable experience, to whose enormous support, within and beyond this thesis, I am deeply indebted. In addition, I appreciate the discussion with James Martens. Without his generous suggestions, my implementation of the Hessian-Free algorithm would not be so smooth. I would also like to thank my reader, Paul Fieguth, for his detailed and helpful comments. Special thanks to Jane Russwurm, who warmly helps me with my English writing and proofread part of this thesis. Many thanks also go to colleagues in the Computational Neuroscience Research Group and the van der Meer Lab, especially to Xuan Choo, who helps me every time when I have problems with my computer, Terry Stewart, who patiently answers all my questions about the NEF, Rob Corss and Sushant Malhotra, who explain animal experiments to me in intuitive and interesting ways.

Dedication

This thesis is dedicated to my parents: Xiao Chunling and Wu Huabin. My gratitude to them is far beyond any word in the “acknowledgement” part could express.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

As one of the most investigated parts of the brain, the hippocampus has been intensely studied in both animal and human experiments. Notably, two seminal observations led to two different hypotheses about the hippocampus [70]. In the early 1970s, O'Keefe and Dostrovsky [62] discovered the famous place cells, neurons in the hippocampus that fire selectively at specific locations. This discovery later inspired the theory that the hippocampus plays the role of a cognitive map [64], by which animals maintain maps of environments. Earlier than that, one of the most famous neurosurgeies, complete removal of both hippocampi, had been performed on a patient called H.M.[1] [78] in order to control his severe seizures. H.M. immediately lost his ability to recall any event that happened after the surgery that removed a large part of his hippocampus. H.M.'s misfortune made important contributions in the development of the theory that the hippocampus supports episodic memory, through which certain experiences are stored in a declarable state.

The dichotomy between the hippcampus' role as a cognitive map or a central player in episodic memory is more likely to be a result of different experiments, rather than reflective of differences in the hippocampus itself [14, 22]. For example, in a typical rat experiment, where neural signals are recorded while a rat is running in a maze, location is one of the most important and easily observable measurements. For this reason, the theory of cognitive maps was first developed through rat experiments, based on the discovery of spatially correlated neural firing patterns [62]. Conversely, restricted by either ethical concerns or the subject's natural behaviours, primate experiments, especially human experiments, seldom involve controlled location change[2]. As a result, the study of spatial behaviors in primate experiments is severely limited. However, higher level memory tasks such as episodic recall are more often performed [81], resulting in the development of a theory of episodic memory in the context of primate experiments [84].

With the development of experimental techniques, however, more recent evidence from rat and primate experiments suggests an intimate relationship between cognitive maps and episodic memory. For instance, recent data show that rat hippocampal cells encode both spatial and non-spatial variables [20, 53, 54, 60, 66], in favor of multimodal representations in a memory space [22]. Moreover, the reactivation of hippocam-

[1] His real name is Henry Gustav Molaison.

[2] However, more recent experiments using virtual reality also found place cells in human hippocampus[23]

pal cells' spatially correlated patterns, or replay, is believed to be the neural correlate of the retrieval of episodic memory [13, 17, 29, 86, 92]. In addition, neural activities resembling one of the signature features of cognitive maps, remapping, in which the selective firing patterns of hippocampal cells are re-organized when an animal is exposed to a different environment, were also observed in the primate hippocampus [43].

Such converging experimental results motivate researchers to combine the two hypothesized roles of the hippocampus. In this thesis, we first introduce how the hippocampus can represent multi-modal information through vectors in a multi-dimensional memory space. Under this scheme, the cognitive map and episodic memory theory are unified naturally at the representation level. Further, we demonstrate how sequences of vectors can be stored and retrieved in recurrent neural networks (RNNs) structurally similar to the hippocampus, resembling the storage and retrieval of episodic memory. Combining vector representations and temporal sequence learning, we simulated both forward [85] and reverse replay [29] observed in rodent experiments. From a computational perspective, our model supports the previously hypothesized role of awake replay in reinforcement learning [13, 29, 45]. Based on the simulation results, we propose an experimentally testable prediction that forward and reverse replay rely on synaptic change and persistent neural activity respectively.

Chapter 2

Background

Supported by a massive amount of experimental data from both humans and animals (e.g., [3, 6, 9, 12, 14]), the hippocampus is considered the as the place where episodic memory, the memory of sequential events, is encoded. However, how episodic memory is stored and retrieved in the hippocampus is still unclear, despite the existence of various theories (e.g., [9, 12, 14, 37, 84]). In this chapter, we start our investigation of how the hippocampus supports episodic memory by reviewing essential background materials.

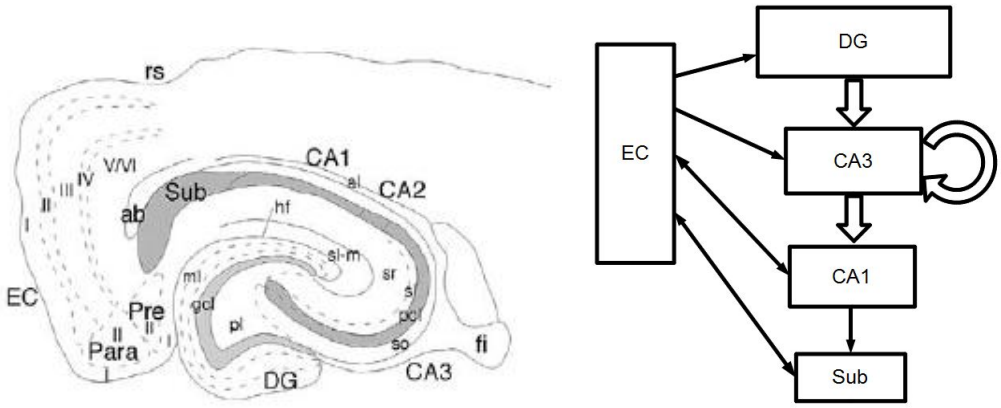
2.1 Anatomy of the hippocampus

The hippocampus is part of the forebrain, sitting beneath the neocortex. Figure 2.1 illustrates the anatomical structure of a rat's hippocampus. It is characterized by feedback loops at different levels: the local recurrent connections at the CA3[1] and the global feedback connections passing through all components of the hippocampal formation (EC $\rightarrow$ DG $\rightarrow$ CA1 $\rightarrow$ CA3 $\rightarrow$ Subiculum $\rightarrow$ EC) [4].

The hippocampus receives information from different sources mainly through the entorhinal cortex (EC), which closely interacts with a diverse range of other cortical areas, permitting the encoding of multi-modal information in episodic memory. The dentate gyrus (DG) is one of the only two places in the brain where adult neurogenesis happens [1] (the other is the olfactory bulb). The EC is connected to DG through the perforant pathway, and it sends direct projections to the CA3, CA1 and the subiculum. The DG granule cells send their output to the pyramidal cells in the CA3 through mossy fibers, whose strong connections enable the CA3 neurons to be activated by very sparse activities in the DG. Compared with the typical mostly-local connections in the neocortex, the dense recurrent connections across the whole CA3 region are unique in the brain, allowing globally synchronized activities of the CA3 pyramidal cells. Pyramidal cells in the CA1 receive input from the CA3 through Schaffer collaterals. Traditionally, the DG and the CA are considered to be the central part of the hippocampus, called the "hippocampus proper"; the hippocampus proper and its surrounding area, including the EC, the subiculum, and the parasubiculum, are together called the hippocampal formation[2].

[1] CA is the acronym for cornu ammonis, or Ammon's horns. However, this full name is seldom used.

[2] However, there are other opinions concerning which parts should be considered as the hippocampal



(a) (b)

Figure 2.1: The anatomical structure of the hippocampus and its cartoon illustration. Figure (a) is a diagram of a rat's hippocampus, showing the anatomical structure of the hippocampal formation (EC: entorhinal cortex, DG: dentate gyrus, Sub: subiculum) [4]. (b) illustrates main parts of the hippocampal formation and connections between them.

2.2 Hippocampal cells

The most famous neurons in the hippocampus are probably the place cells, which are known for their selective spatial firing patterns. Place cells were first discovered in the CA1 region in the hippocampus, and later were also found in the CA3, the DG and the subiculumn [4, 62]. Initially, place cells were only thought to support the neural representation of the spatial properties of the environment, forming cognitive maps [63] for spatially related behaviours such as navigation. A place cell fires only when the rat is close enough to that cell's place field[3]. Figure 2.2 illustrates the place fields of some place cells. The contours of these place fields can be approximated by Gaussian functions. When the firing of place cells are arranged topographically according to their place fields (so that place cells representing similar places are together), a population level Gaussian-shaped pattern, the activity packet, can be observed (figure 2.3). Arising from the Gaussian tuning curves of individual hippocampal cells, activity packets can be used to estimate the location of the animal. For example, the activity packet in figure 2.3 indicates that the rat is at the center of an environment.

More recently, temporally selective activities of the hippocampal cells were observed at different laboratories, from recordings during delay periods of tasks [66, 53]. Specifically, they show that strong temporal modulation in their recoded hippocampal cells is preserved even after statistically removing the influence of location and behaviour. For this reason, they call these cells "time cells"[4]to contrast with the famous place cells

formation [4].

[3]Exceptions are that place cells also fire in hippocampal replay events, which may happen when the rat is in REM sleep, slow wave sleep or staying stationary. Besides, a place cell may have multiple place fields [27].

[4]Although the concept of "time cells" may be controversial, for simplicity, later we use neurons "rep-

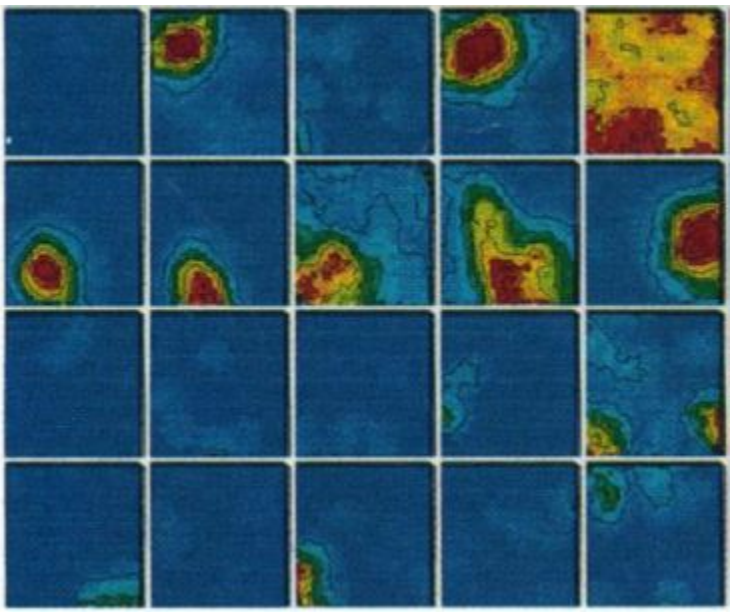


Figure 2.2: Spatial tuning of single hippocampal cells [91]. The color map, in which red represents high firing rates and blue represents low firing rates, showing single cell recording from hippocampal neurons in the CA1 region of the hippocampus when a rat was freely foraging in an open field. In this specific environment, while some cells showed clear location selective firing, other cells either fire thoughout the whole environment (e.g., upper-right corner) or keep silent all the time (e.g., upper-left corner).

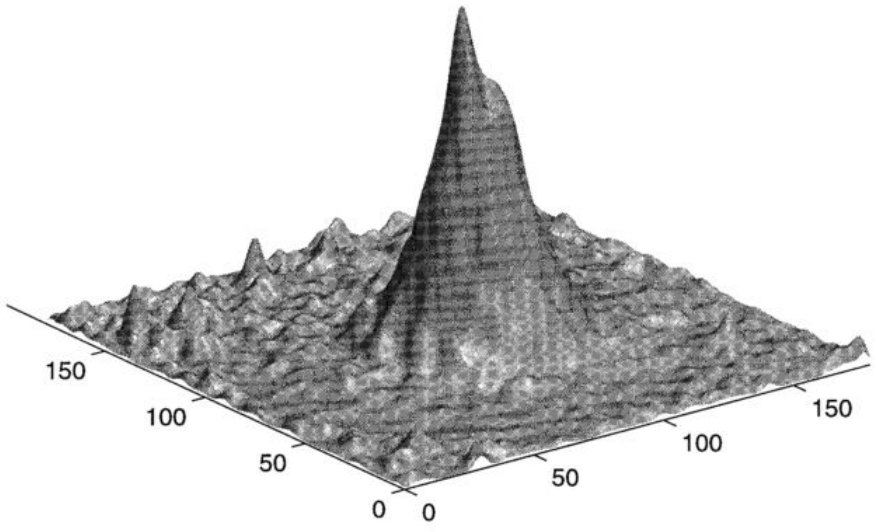


Figure 2.3: Activity packet visualizing the firing rate of a population of place cells [75]. Neurons are arranged according to their place fields, so that their positions on the figure are roughly the same as their preferred locations in the environment. In this figure, neurons at the center fire most strongly, implying the rat was near the center of the testing environment.

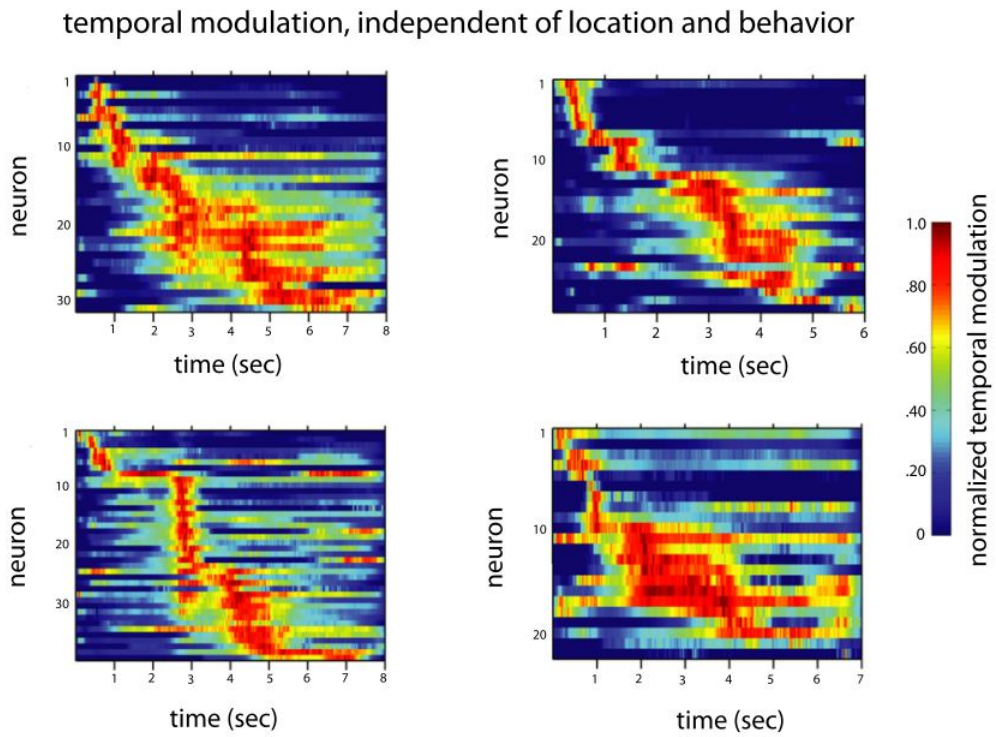


Figure 2.4: Hippocampal cells are tuned to specific locations in a temporal sequence [53]. The four panels show the record from four different CA1 cells. After statistically excluding the influence of spatial location and behavior, these cells still show strong temporal modulation.

(figure 2.4). Figure 2.5 further shows that during delay, the activities of these cells are modulated by both spatial and temporal inputs.

With the discovery that hippocampal lesions impair various memory related tasks [14, 22, 12, 54], researchers started to realize that information represented by these place cells may be part of episodic memory. This more recent view is consistent with early human clinical cases indicating that the function of the hippocampus is related to episodic memory [14].

Later in this thesis, we examine forward and reverse replay as examples of storing and retrieving episodic memory, as introduced in section 2.4, 2.9, and simulated in section 4.3, 4.4. Restricted by the scope of this thesis, we only note here that hippocampal replay is not a simple function of experience [33]. Correspondingly, the role of hippocampus in constructive episodic memory (imagining future experiences, in addition to recalling past experiences) has been confirmed in human studies [76, 36, 35].

---

resent temporal signal" and "represent time" interchangeably.

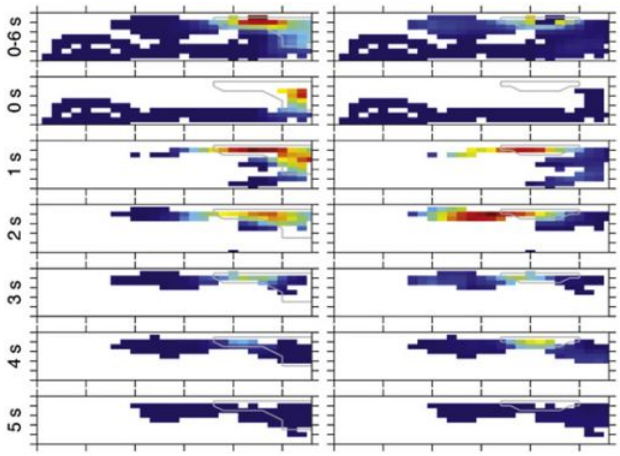


Figure 2.5: Neural activity during the delay depends on both space and time. In the experiment in [53], a delay is introduced between two phases. The spatial firing rate maps for 2 hippocampal cells are shown. For each cell the top map is for the entire delay, and maps below focus on each individual second of the delay.

2.3 Remapping

In order to support episodic memory, a problem faced by the hippocampus is the possible ambiguity in representation. An example commonly used by researchers studying episodic memory is the scenario of parking a car: although both the car and the parking lot are the same each day, ones needs to remember today's parking location (or path) as distinguished from that of yesterday or before. In this example, the time (today or other days) provides the context based on which different memories of parking can be retrieved. In rodent maze experiments, this problem is usually presented as requiring a rat to behavior differently based on different context [15].

Given the presence of ambiguity in various tasks, it is not surprising that the brain, especially the hippocampus, has different ways to disambiguate inputs at the neural level. First, the Gaussian tuning curve (figure 2.2) leads to sparse firing patterns (figure 2.3, where only cells proximal to the center are firing). Sparse firing patterns maintain low interference because of low overlapping between different patterns. Although the functionality of neurogenesis in the DG is still not entirely clear, it is believed to facilitate the orthogonalization of input into the hippocampus [1, 7, 83]. Another way to disambiguate is remapping, as if a animal uses different maps for different contexts.

Two kinds of remapping are observed in rodent experiments[5]: rate remapping and global remapping [15]. In rate remapping, a hippocampal neuron fires differentially in different environments only through its firing rate changing. In global remapping, in addition to the change of firing rate, a neuron's place field may either change or disappear in different environments (figure 2.6). Through these changes of spatial correlated firing, different "maps" are formed for different environments. For example, the same location at the center of two different environments will be represented by different neurons.

[5]Phenomena similar to remapping are also observed in human fMRI studies [9].

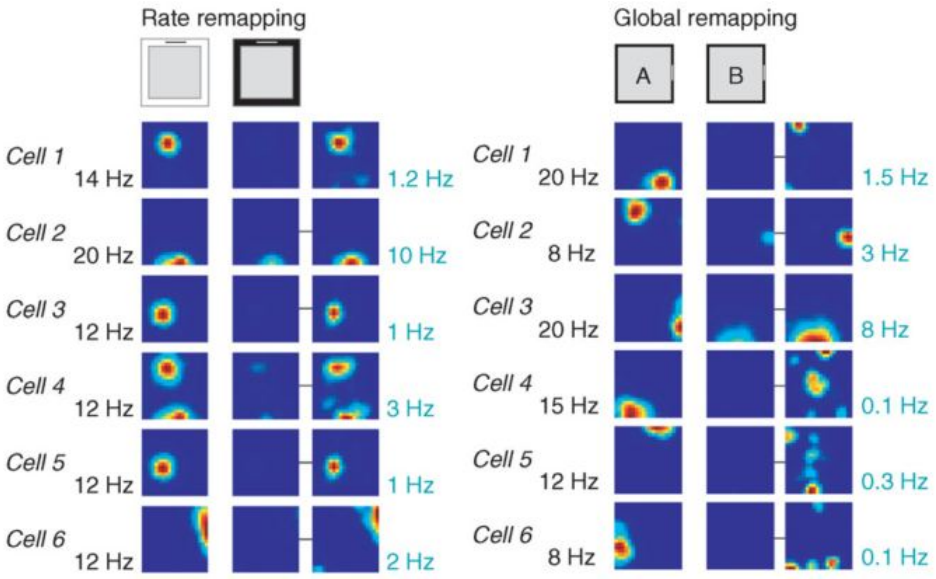


Figure 2.6: Rate remapping (left) and global remapping (right) [15]. The color map is used to indicate firing rate as with previous figures. On each side, the first column illustrates the firing of neurons in one environment, while as the last two columns illustrate that in the other environment. The mapping from firing rates to colors are the same in the first two columns, but are rescaled in the third column. For rate remapping, the activity of neurons looks the same after rescaling, indicating the firing in two different environments is only distinguished by firing rates. On the contrary, for global remapping, the neural activity is different even after rescaling the color map, showing the preferred locations of these cells are also changed.

Similarly, traces sharing similar relative potions in two environments will be represented by distinct neurons. Thus ambiguity cased by sharing representations (sharing "maps") can be largely reduced.

2.4 Forward replay

We mainly consider the forward replay observed in rodent T-maze experiment [46, 86][6]. A T-maze (figure 2.7) has three connected arms (left, central, right), with possible feeders providing food or drink as reward at the left or right arms. When a rat is at the central arm of the T-maze, it needs to choose which side to go to, as only one arm of the T-maze gives the desired reward. The location right before the intersection of the "T" point (indicated by the white circle in figure 2.8) is called the decision point. At early learning stages of the T-maze experiments, the rat pauses at the decision point before proceeding, as if it is pondering which side to go to [85].

[6]In this thesis, we use the term "forward replay" and "sweep" interchangeably to refer to the reactivation of hippocampal neural activities at the decision point, although the exact relationship between these terms are not entirely clear in neuroscience literature.

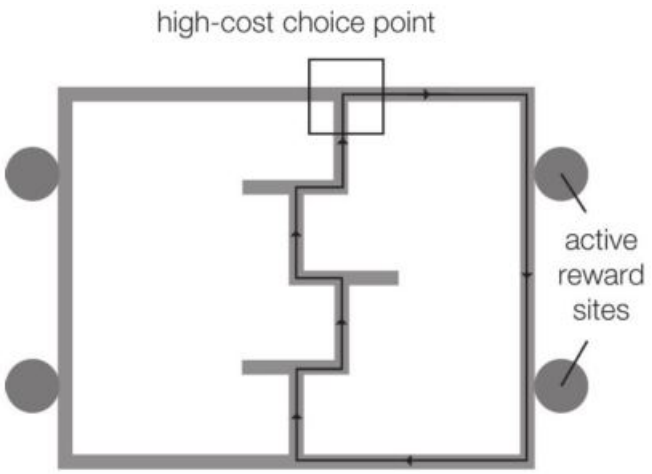


Figure 2.7: The T-maze used in [85].

This intuitive hypothesis is supported by multi-cell recordings from the rat hippocampus, which indicates that previously visited routes including both arms are replayed at the decision point through the reactivation of hippocampal cells (figure 2.8). Interestingly, at the about same time as the replay (or "sweep"), cells in ventral striatum that are activated when reward is presented are also firing, implying that forward replay may be linked to the evaluation of future outcome[86].

2.5 Reverse replay

As suggested by its name, reverse replay also involves the reactivation of past neural activities representing past experiences, however, in a reversed order. Although the replayed sequences can be either remote [17, 33] or recent in the past [29], in this thesis we mainly consider the reverse replay of recent experience[7]. This kind of replay happens immediately after a rat reaches a goal or obtains a reward [29]. For reasons we will discuss in more detail in section 4.4, remote reverse replay may be similar to forward replay, while as the reverse replay of recent experience is likely to be based on a different computational mechanism. Figure 2.9 shows the reverse replay recorded during a single lap on a linear track.

[7]Therefore, without specification, later in this thesis "reverse replay" means the replay of recent experience.

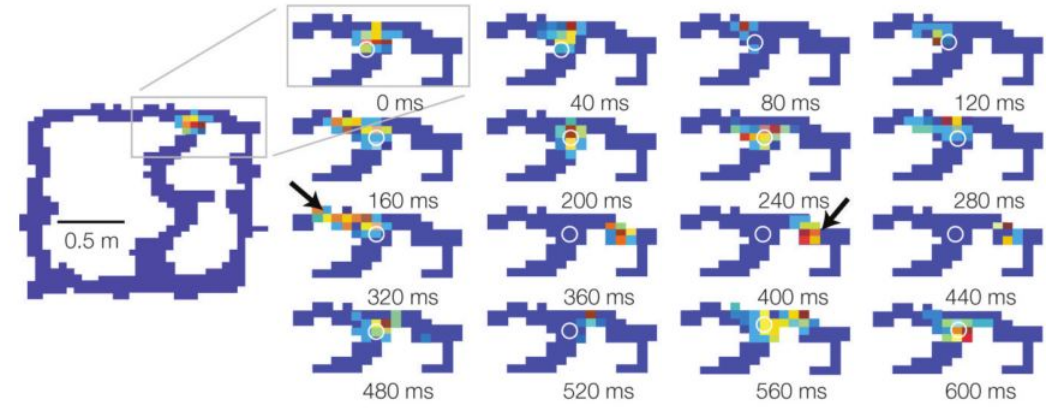


Figure 2.8: Replay at the T-maze decision point [86]. Each sub-plot shows decoded representation at different time during the rat's being at the decision point (white circle); the color code indicates the probabilities of the represented location, where blue to red means low to high probabilities. From the decoded spatial representation in these sub-plots, although the rat was keeping still, the locations represented in its hippocampus drifted towards each possible future direction, indicating the replay of previous experiences at different arms.

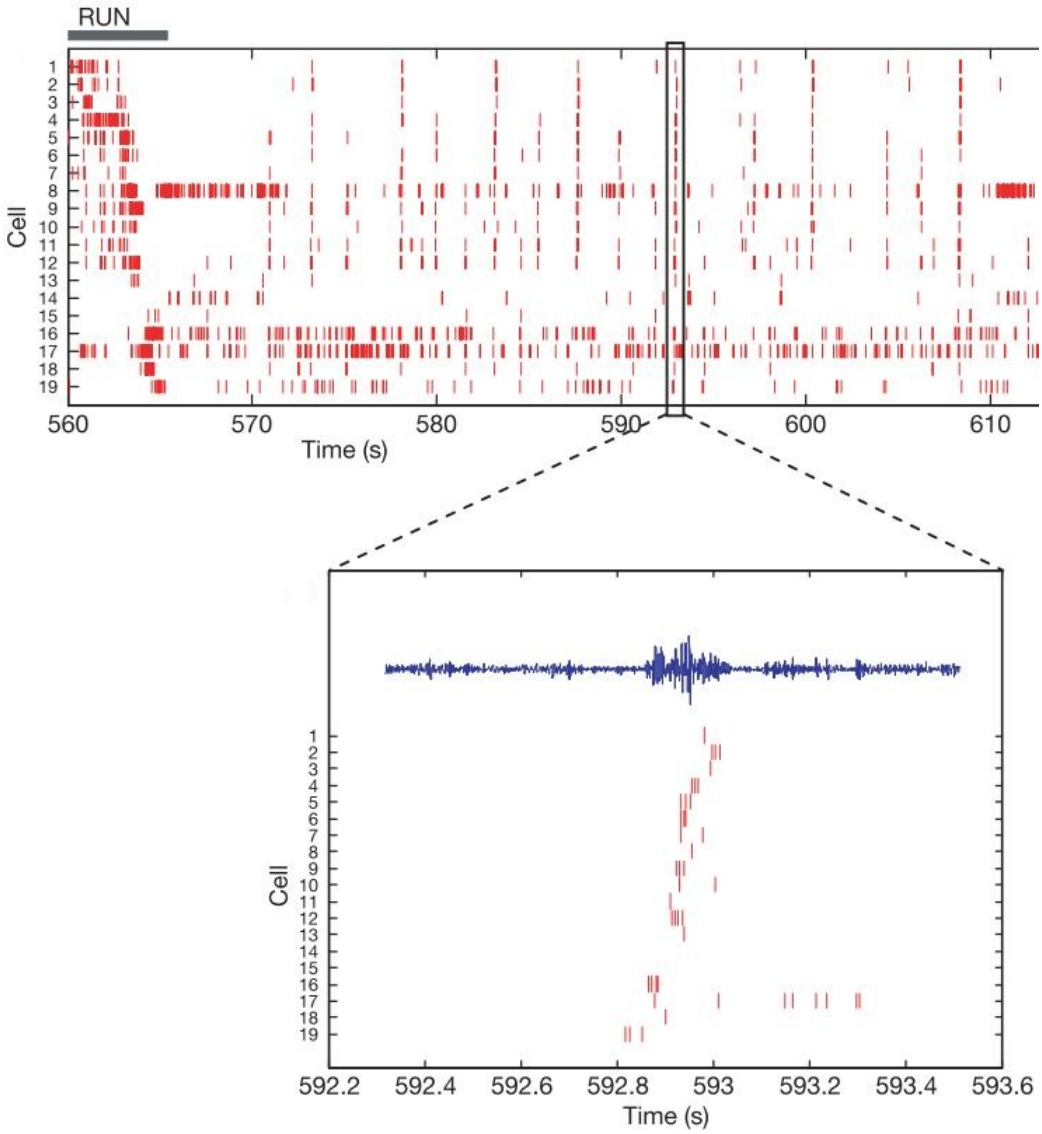


Figure 2.9: Reverse replay events [29]. The upper panel shows all the spikes recorded from 19 hippocampal neurons during a single lap on a linear track. The rat was running during the first 5 seconds (560 - 565, under the gray mark and "RUN" at the top-left corner). The magnified view in the lower panel shows the reversely reactivated spiking trains during the rat was resting after reaching the goal. It is also shown from the two panel that the reverse replay happened at a time scale about 10 times faster than the original experience.

Chapter 3

Multi-dimensional neural representations

We can understand episodic memory as temporal sequences in a high dimensional memory space, in which different kinds of (multi-modal) information are bound together through experiences [22]. Mathematically, temporal sequences can be represented as sequences of vectors in this memory space. In this chapter, we introduce how vectors can be represented in spiking neurons and how we can use such vectors to represent variables potentially forming episodic memory. In the next chapter, we describe how sequences of these vector can be remembered as traces of episodic memory.

While a vector may have multiple elements, or multiple dimensions, we can divide a vector into subgroups that represent different information from different sources. For example, a six dimensional vector, $t = (1,0,3,7,5,9)^\top$, can devote its first two dimensions to represent hours, the middle two dimensions to represent minutes and the last two dimensions to represent seconds. Together, t represents a specific time. In a similar manner, for the vector v used in our hippocampal representation, we can divide it into dimensions representing space, dimensions representing task, etc.

3.1 How spiking neurons represent vectors

The Neural Engineering Framework (NEF) [25] provides a theoretical framework for characterizing neural representations, neural transformation and neural dynamics across different levels of description. This section introduces the first principle in the NEF, neural representation, by which vectors can be represented in biologically plausible spiking neurons. Although the NEF supports simulation of neurons with different levels of physiological details, for simplicity, this thesis uses leaky-integrated-and-fire (LIF) neurons [25].

The sub-threshold dynamics of a LIF neuron is described by the differential equation:

$$\frac{dV}{dt} = -\frac{1}{\tau^{RC}}(V - J \cdot R) \tag{3.1}$$

where both τ^{RC}, the membrane time constant, and R, the leak resistance, are parame-

ters. This equation describes how the membrane potential V of a LIF neuron change with time t given the input current J when V is below a threshold $V_{threshold}$. Once the membrane potential reaches a threshold $V_{threshold}$ (at which time the current also reaches its threshold $J_{threshold}$), the LIF neuron generates a spike. A spike, or action potential, is a transient and dramatic increase of its membrane potential followed by a refractory period. During the refractory period, the membrane potential keeps at a low level for a short amount of time specified by the parameter τ^{ref}. When the input current is constant, we can analytically solve for the firing rates of a LIF neuron:

$$a(J) = G(J) = \frac{1}{\tau^{ref} - \tau^{RC} ln(1 - \frac{J_{threshold}}{J})} \tag{3.2}$$

The NEF suggests that, for a given neuron, its input current is regulated by a vector x in its input space through

$$J = \gamma \cdot (e \cdot x) + b \tag{3.3}$$

where e is the normalized unit encoder of this neuron, γ is gain factor, and b is the input bias. The parameters γ, e, b as well as τ^{RC} and τ^{ref} together specify the tuning curve of a LIF neuron, as a function of vector x in its input space. The encoder e is also called the preferred direction of a neuron, because for input vector x with fixed lengths, the dot product $e \cdot x$ reaches a maximum when x has the same direction as e, which in turn gives the maximum firing rate (equation 3.3 and 3.2).

The tuning curve of 20 LIF neurons with one dimensional input is shown in figure 3.1, and the tuning curve of a neuron with two dimensional input is shown in figure 3.2. Given a population of heterogeneous neurons, through activation functions defined by equation 3.2 and 3.3, a vector x can be encoded into the firing of these neurons. Since activation functions are non-linear, this is a process of non-linear encoding.

The encoding process need to be paired with a corresponding decoding process that reads out the encoded vector x. Given the complexity of possible non-linear decoding, the NEF suggests that linear decoding is a general and more biologically plausible mechanism. With a decoder d for each neuron in a population, we can decode the estimation of x from the activity of this population:

$$\hat{x} = ad \tag{3.4}$$

For a population representing x in an input space $\mathbb{S}$, equation 3.4 should give a good estimation for all x in $\mathbb{S}$. Therefore, we extend equation 3.4 for a set of representative[1] samples in the input space $\mathbb{S}$ by substituting each lower case variable with capitalized letters representing samples in the whole input space, giving a system of linear equations:

$$\hat{X} = A^{\top} D \tag{3.5}$$

Now each column represents one dimension of space $\mathbb{S}$, and each row of X represents a sample in this space. Each column of A represents the firing rate of a neuron in this

[1] whether a set of sample is representative is determined by the space $\mathbb{S}$. In general, more samples are needed for higher dimensional spaces with more complex structure.

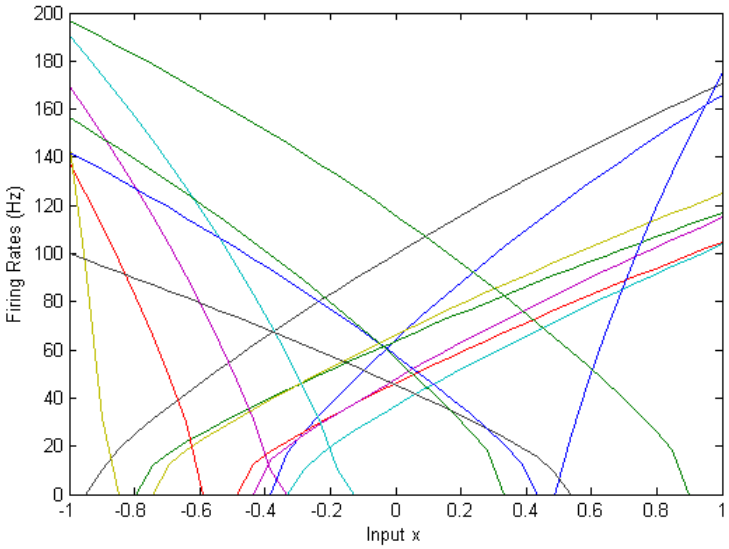


Figure 3.1: The tuning curves of 20 LIF neurons. Different neurons are represented by different colors. These heterogeneous neurons encode a scalar variable from -1 to 1. Using a set of linear decoders, the scalar can be reconstructed from the activity of this neuron population. We use the representation of scalars as a base case, which will be expanded into function representations of the hippocampal cells.

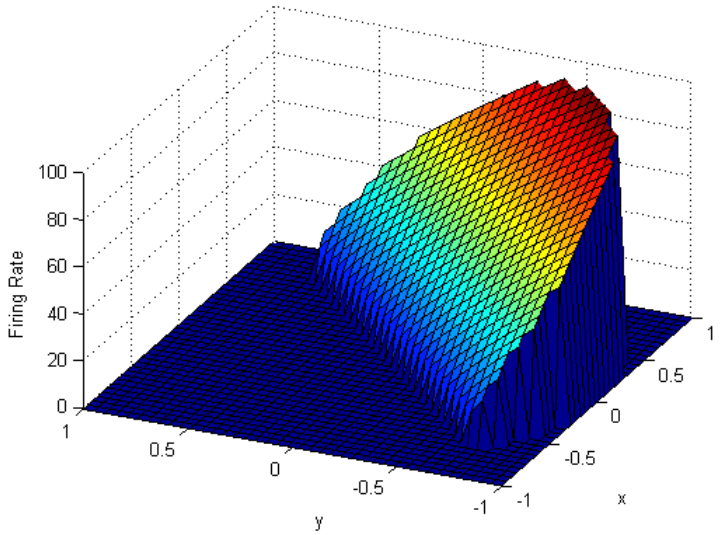


Figure 3.2: The two dimensional tuning curve of a LIF neuron. The firing rate of this neuron is a function of both input x and input y. A slice of this figure, in which the the firing rate is subject to only one variable, is similar to curves in figure 3.1

population, while each row represents the activity of the population given one sample input. Similarly, each column of D represents the input dimension corresponding to X, and each row of D represent the decoder for one neuron.

We can understand equation 3.5 as projecting vectors from the neural space, where the tuning curves of neurons serves as basis, back to the original Cartesian space. Therefore, finding the decoders is a regression problem and the decoders D can be solved analytically using the conventional normal equation:

$$D = (A^\top \cdot A)^{-1} \cdot A^\top \cdot X \tag{3.6}$$

As one of the most well studied equations in numerical analysis, equation 3.6 can be evaluated efficiently through a singular value decomposition (SVD).

Now we turn to the temporal dynamics of LIF neurons by allowing the input current to change. Although equation 3.2 is no longer true, the effect of spikes can still be accumulated though the post-synaptic current (PSC):

$$h_{psc}(t) = e^{-\frac{t}{\tau_{syn}}} \tag{3.7}$$

As illustrated in figure 3.3, the PSC smooths the spikes; the smoothness is controlled by the parameter τ^{syn}, the synaptic time constant. The PSC bridges rate coding and spike coding by providing a diminishing time window, in which the effect of spikes is accumulated as the strength of the current. Filtering a spike train by the PSC can recover the original signal, albeit with additional errors introduced the the fluctuation of the spikes (figure 3.4). This filtering process can be embedded at the synapse as

$$\frac{dJ_{PSC}}{dt} = -\frac{\tau_{PSC}}{J_{PSC}} \tag{3.8}$$

Thus, we have now established a representation scheme for spiking neurons. Our way of deriving the optimal linear decoders supports the smooth transactions between rate coding and spike coding, two neural coding schemes that may not be fundamentally different [25]. Nevertheless, supporting spikes makes our model easier to compare with experimental data. For more detailed description, refer to [25].

In the context of episodic memory, for a population of neurons representing events in memory as vectors x, the input space $\mathbb{S}$ becomes the memory space. In the following sections, we discuss properties of this memory space, and further characterize vectors in this memory space.

3.2 Spatial representation

Using the vector representation scheme introduced in the last section, many variables can be encoded and decoded in spiking neurons. However, more structured variables are required to represent information essential in episodic memory, such as locations. This section extends the vector representation scheme by showing how more structured functions can be represented. The basic idea of function representation in NEF is decomposition (dimension reduction) of discrete functions.

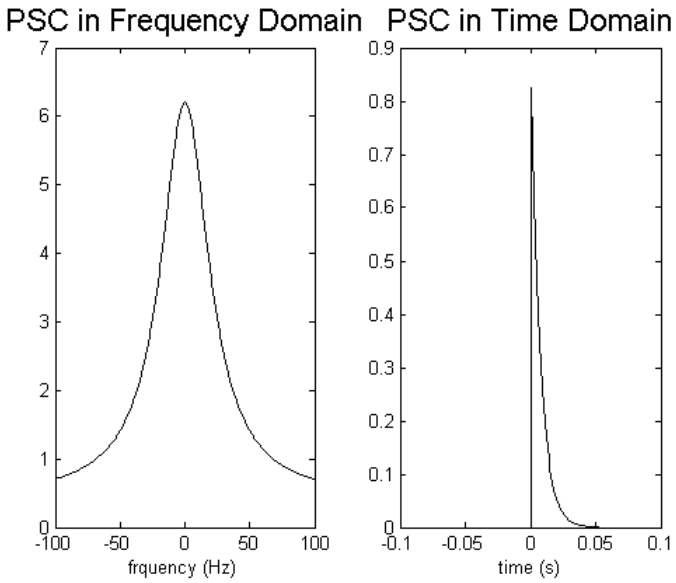


Figure 3.3: The PSC in the frequency and temporal domains. As we can see from the frequency domain illustration, high frequency components are suppressed. Therefore, the PSC smooths spikes.

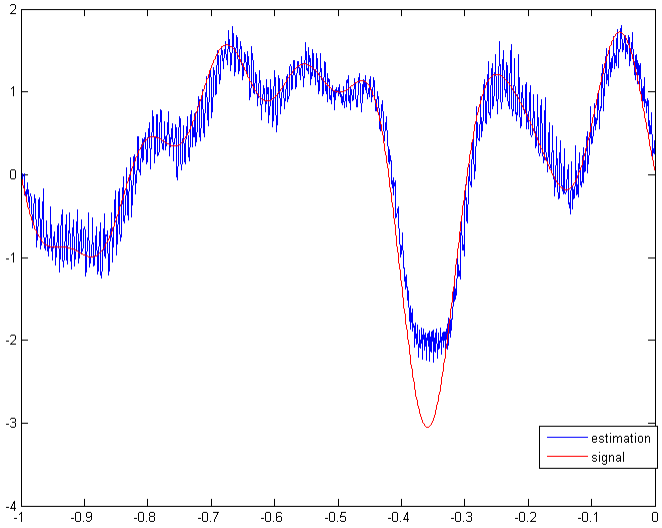
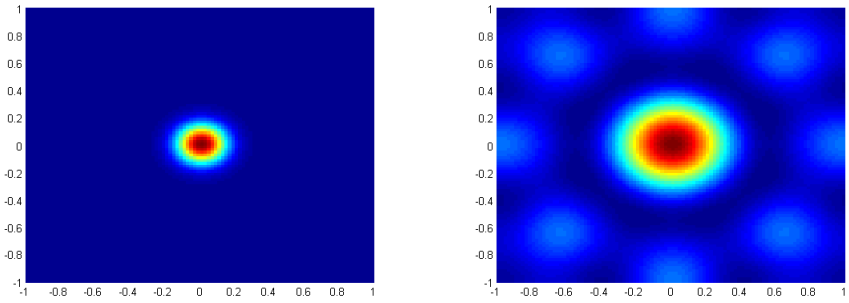


Figure 3.4: Estimation of the original signal. The estimation is obtained from filtering spikes generated by each LIF neuron in a population of 20 LIF neurons, then sum up the filtered signals weighted by the decoders (equation 3.6). The fluctuation is introduced by the spikes and the high-frequency components of the PSC.



(a) (b)

Figure 3.5: Reconstruction of the Gaussian function with different maximum frequency components. (a) Reconstruction using components with a maximum frequency of 10 Hz. (b) Reconstruction with components with a maximum frequency of 3 Hz. Although (a) is closer to the original Gaussian function, (b) also gives a reasonably good approximation for our qualitative simulations.

Inspired by the Gaussian shaped spatial tuning curves of the hippocampal cells (section 2.2), Conklin and Eliasmith [16] simulated Gaussian firing patterns of place cells from a population of neurons representing two dimensional Gaussian functions.

A Gaussian function (figure 3.5 (a))

$$f(x,y) = exp(-\frac{(x-\mu_x)^2 + (y-\mu_y)^2}{2\sigma_2}) \tag{3.9}$$

can be discretized into a vector with a finite (large) number of dimensions. In order to guarantee an acceptable representation precision, the number of neurons in a population usually increases as the number of dimensions represented increases.[2] Therefore, dimension reduction is required to represent functions in our vector representation scheme. For any function, one conventional way of performing dimension reduction is using the Fourier transformation, decomposing the function into basis and coefficients

$$F_{kl} = \sum_m \sum_n f(x_m, y_n) \cdot e^{-2\pi i(\frac{k}{M}m + \frac{l}{N}n)} \tag{3.10}$$

where f's are values of the function in spatial domain and F's are coefficients in frequency domain. Preserving a finite number of coefficients results in a reduced approximation to the full Fourier space. Using the inverse Fourier transformation, we can restore the spatial domain function from coefficients F's and the oscillatory basis $e^{2\pi i(\frac{k}{M}m + \frac{l}{N}n)}$ (e.g., figure 3.8)

$$f_{mn} = \sum_k \sum_l F_{kl} \cdot e^{2\pi i(\frac{k}{M}m + \frac{l}{N}n)} \tag{3.11}$$

Recall that the Fourier transformation of a Gaussian function is still a Gaussian function, so the coefficients of a Gaussian function diminish quickly as the frequency in-

[2]This statement is not strictly true, since different dimensions may be correlated.

creases. This property implies that we can discard high frequency components and still maintain a good representation of the Gaussian function. Figure 3.5 shows the reconstruction using components with frequencies less than 3Hz, compared with the reconstructed Gaussian function with more frequency components.

In general, function decomposition (not necessarily Fourier decomposition) provides a way to simplify the representation of an otherwise continuous space. This also simplifies the calculation of similarity between an input and the preferred direction vector of a give neuron[3]. Mathematically, since the firing of a neuron is only affected by the dot product between its encoder and the input vector obtained from function decomposition, the basis is not directly involved in neural computation. Providing that the basis is chosen properly (i.e., it is able to give a good reconstruction of the original function) and the coefficients are represented well, the firing pattern is only affected by the function being represented. In fact, both a Fourier basis and a basis obtained from principal component analysis (PCA) can reproduce the same Gaussian firing pattern reported in [16]. Although in theory PCA does a better job at dimension reduction, a Fourier basis may be realized in the brain as the observed sub-threshold oscillations at different frequencies (this issue is further discussed in section 3.2.1).

Using this spatial representation scheme, an activity packet resembling that in figure 2.3 is simulated in a population of 4,900 neurons (figure 3.6), where each neuron represent a Gaussian function with width (δ in equation 3.9) 0.1, centralized at locations evenly sampled in a 2D square. In order to illustrate the activity packet, these neurons are organized topographically. Note neurons in this population are not interconnected, although they can be connected in the same way as in [16], forming an attractor. Although the activity packet is able to indicate the location of the animal from the distribution of neural activity (red areas in 3.7 a), using the full NEF neural representation gives a much better estimations (3.7 b). With the multi-dimensional decoders, instead of estimating the compound information from the 1D firing rate only, different dimensions of the represented vectors can be decoded separately. Therefore, the advantage of using neural coding is even more obvious with the presence of non-spatially encoded information and noise.

3.2.1 Fourier basis and grid cells

As a digression from the vector representations in episodic memory, here we briefly give a justification of choosing the Fourier basis for function decomposition. A Fourier basis function $e^{2\pi i(\frac{k}{M}m+\frac{l}{N}n)}$, which is sometimes written as sine and cosine functions, is illustrated in figure 3.8. The interference of three such waves separated by 60 degrees produces the hexagonal pattern reminding us the firing patterns of grid cells (figure 3.9).

Such an interference mechanism is used in various models of grid cells [11, 31, 80]. Physiologically, the Fourier basis (waves) may be realized in neural networks as subthreshold oscillations which are found in the hippocampal formation. More specifically, these oscillations may be originated from the intrinsic oscillations in the entorhinal cor-

[3]Therefore, although only vectors are represented by the neurons, we sometimes call it function representation in this situation

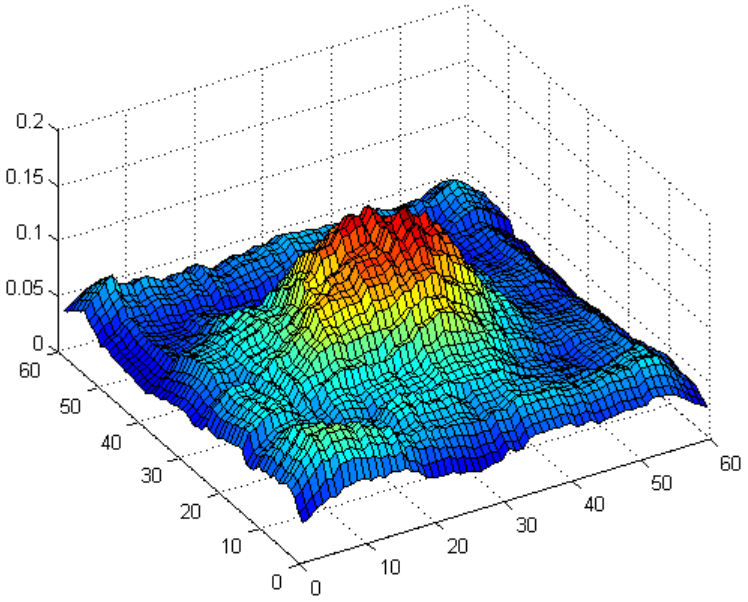
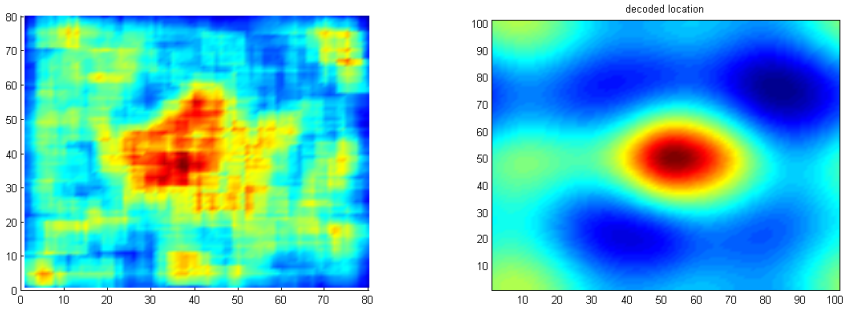


Figure 3.6: Simulation of an activity packet using 4900 LIF neurons arranged according to their place fields. Each cell represent a Gaussian function through their coefficients obtained from Fourier decomposition. These Gaussian functions have with fixed width, sampled evenly from the 2D square. The two horizontal axes index each cell and the vertical axis is proportional to the firing rates.



(a) (b)

Figure 3.7: The firing rate of a population compared with decoded location from the same population. Figure (a) is a top view of figure 3.6, indicating the firing rate of neurons in this population. Figure (b) shows the decoded spatial representation using the optimal linear decoder. This 2D map is obtained from taking the dimensions representing space from the decoded vector (equation 3.5) as coefficients and reconstructing the 2D surface together with the Fourier basis. While it is possible to (roughly) determine the encoded location from the firing rates only (a), neural decoding give a much better estimation of the represented location (b).

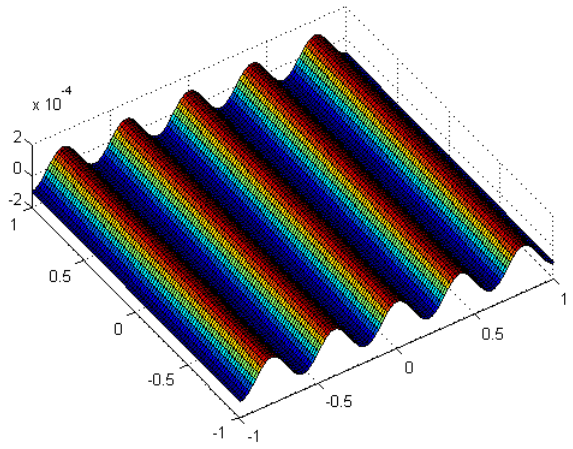


Figure 3.8: An example of a Fourier basis function.

tex [31, 38]. The frequencies of these oscillation are proportional to the velocity of the animal. Specifically, the direction information comes from the head direction cells in the parahippocampus. These oscillations are summed, forming new sub-threshold oscillation patterns like that in figure 3.9 A. When receiving spiking input with certain frequencies, the grid cells are more likely to reach the threshold and fire when the sub-threshold oscillation near the peaks (places in figure 3.9 (a) where the color close to red). Thresholding the pattern in figure 3.9 (a) gives figure 3.9 (b) that is similar to the experimentally observed grid firing patterns [30, 34].

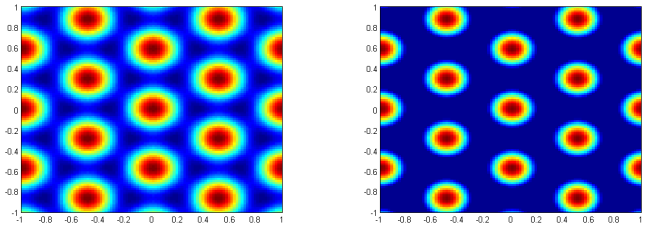
Since both $e^{-2\pi i(\frac{k}{M}m+\frac{l}{N}n)}$ in equation 3.10 and $e^{2\pi i(\frac{k}{M}m+\frac{l}{N}n)}$ in equation 3.11 can be represented as oscillations, the combination of them can form the hexagonal patterns of the grid cells. Given the fact that the entorhinal cortex, where these grid cells reside, serves as both input and (indirect) output of the hippocampus proper, our representation scheme is consistent with the view that the grid cells also support decoding of spatial information, in addition to supporting encoding as an upstream input to the hippocampus[4] [28].

3.3 Temporal representation

In order to form a unified and systematic representation scheme, we specify that hippocampal cells are also tuned to temporal sequences directly. This design choice doesn't violate the assumption that the "time cells" are resulted from intrinsic temporal dynamics [26, 66], since these temporal sequences can be generated internally from an integrator like structure.

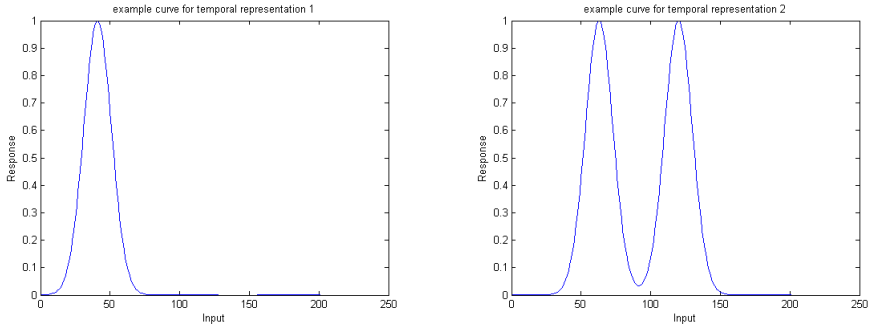
From a modeler's view, by analogy to how space is represented, temporal signals can be represented in our model as vectors obtained from decomposing one dimensional

[4]When grid cells are used as basis directly, the the linear relationship between basis and Gaussian functions is still held, although the coefficients for these basis are computed in a different way [80].



(a) (b)

Figure 3.9: The interference pattern obtained from summing 3 sinusoid waves separated by 60 degrees (a) and the grid pattern obtained after thresholding the interference pattern (b).

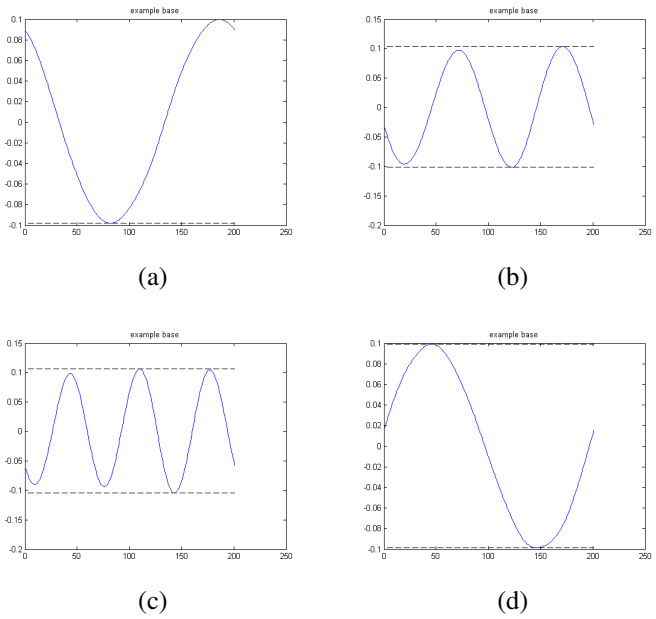


(a) (b)

Figure 3.10: Tuning curves of "time cells" in the simulation. The horizontal axis indicates time steps, while the vertical axis indicates the strength of neural response. These tuning curves enable hippocampal neurons to fire selectively at locations (time steps) in a sequences.

Gaussian functions (time is considered one dimensional). In addition, since some neurons fire in multiple time steps, we account for this phenomenon by allowing some neurons to have tuning curves as summations of two or more Gaussian functions (figure 3.10).

As mentioned before, the choice of basis doesn't affect the firing pattern for given preferred functions. In this section, for temporal representation, we use PCA basis instead the Fourier basis used before. An example of the basis used in our model is shown in figure 3.11. Despite being obtained from a different procedure, they are similar to the Fourier basis. Since the PCA basis can be considered as the optimal orthogonal basis, this similarity indicates that the biologically more realistic and mathematically more convenient Fourier basis is close to optimal. Reconstruction using PCA is illustrated in figure 3.13. The magnitude of these principal components are plotted in figure 3.12.



(a) (b)

(c) (d)

Figure 3.11: Examples of PCA basis. They may look like Fourier basis, but a closer look at the horizontal lines (black dash lines) reveals that they are not. Therefore, the fourier bases are close to optimal.

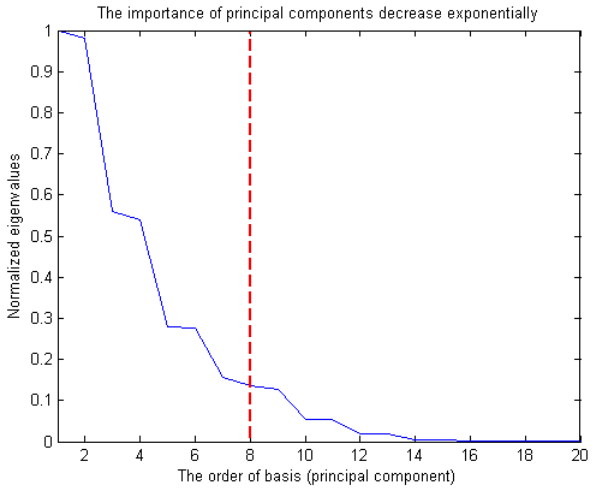


Figure 3.12: Magnitudes (singular values) of the principal components. The red line indicates the number of components used in our experiments.

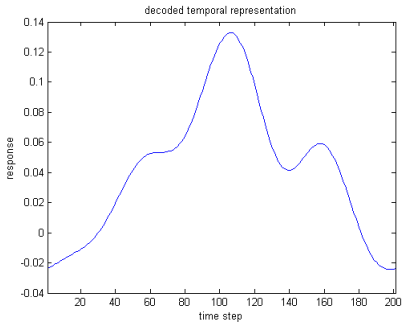


Figure 3.13: Reconstructed temporal representation (a one dimensional Gaussian function with mean at 100) from decoded coefficients (as vectors) and PCA basis. Due to both the limited basis used and the induced noise, the reconstructed temporal representation is not perfectly smooth. Nevertheless, the time step it represents (near 100), as is very clear, as the position that gives the highest value.

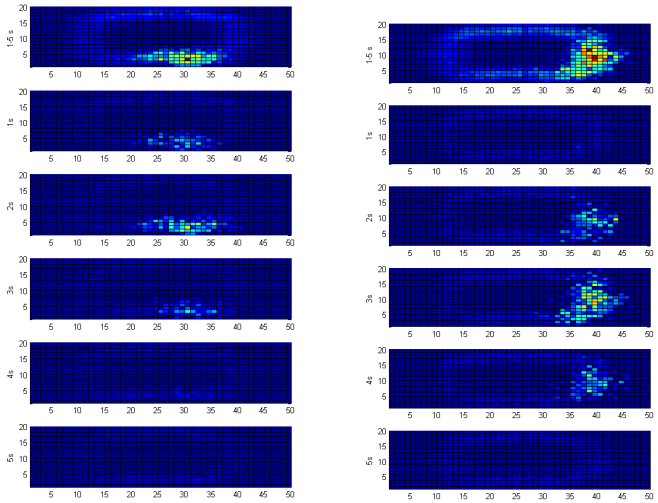
3.4 Combining multi-modal representations

Combining vectors representing multi-modal information gives the multi-modal representations used in our hippocampus model. In addition to spatial and temporal inputs being represented in the way described in the previous two sections, other variables can be represented either similarly through decomposition of functions or directly as vectors.

When a neuron is assigned a preferred direction representing a combination of multi-modal information, this neuron will be activated with its highest firing rate only when the input match with all dimensions of its preferred direction. However, partial matching, such as when the neuron receives inputs matching only the dimensions representing spatial information, will still partly activate the neuron as long as the input is enough to drive the neuron to reach its threshold potential.

Based on this general representation scheme, in addition to spatial and temporal information, we added a one-dimensional variable representing task and two special dimensions representing the context that distinguishes different environments (discussed in detail in section 3.5). The degree with which a neurons is tuned to specific input variables can be controlled through the direction of its encoder. For example, if the preferred direction of a neuron has projections with same length in the subspace representing space and the subspace representing time, this neuron is tuned equally to space and time. On the other hand, if the preferred direction is orthogonal to the subspace representing time, this neurons will not response to temporal input at all. This control of response enables the simulation of "response gradient" across the dorsal and ventral hippocampus [72].

In a population of 4900 neurons, we simulated how representing multi-modal information gives rise to neurons tuned to both spatial and temporal inputs (figure 3.14). In this experiment, spatial input comes from pre-defined paths in a simulated environment, and the temporal input comes from a controlled neural integrator [24]. This controlled integrator consists of a population of 1500 neurons representing temporal information only. Its dynamics is specified through connection weights using NEF, so that the represented temporal information will be advanced automatically at each time step.



(a) (b)

Figure 3.14: Simulated hippocampal neurons firing selectively at a specific place and a specific time. Figures (a) and (b) illustrate the activity of two neurons with different preferred location and time through a 5-second period while the simulated rat is running around a circular track. Neural activity is color coded such that red represents the highest firing rate and blue represents no firing at all. In each graph, the first row represents the activity of this neuron across the whole time period. Each following row shows the neural activity in an individual second, revealing that the neuron is modulated by both space and time.

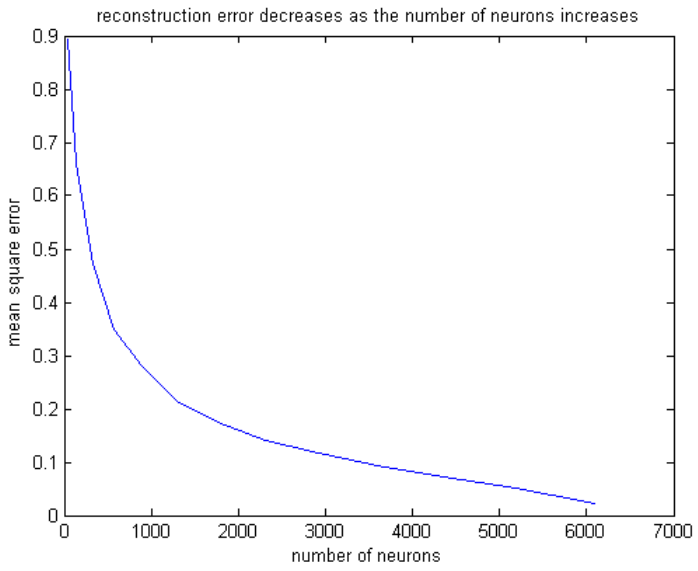


Figure 3.15: Representation error (the mean square error) decreases with the number of neurons .From this figure, our choice of 4900 neurons gives a balanced trade-off between the quality of representation and the number of neurons.

As in our previous discussion in section 3.2, the representation accuracy is restricted by the amount of information (dimensionality of vectors) and the number of neurons in a population. As in figure 3.15, the representation error measured as mean-square-error (MSE) decreases as the number of neurons increases. Conversely, given the information need to be represented and the precision required, we can calculate the number of neurons needed in a neural population.

3.5 Simulations of Remapping

Based on the vector representation in NEF, we proposed two plausible computational mechanisms underlying remapping (section 2.3). First, we use specialized dimensions to provide systematic inhibition on neurons not associated with the current context. Second, we use a transformation matrix to shift place fields of neurons.

3.5.1 Rate modulation

To illustrate the first mechanism, consider a k dimensional unit vector $v = (a_1, a_2 \cdots a_k)^\top$ in the memory space as the preferred direction of a neuron. Now extend v into a $k+p$ dimensional vector $v_e = (a_1, a_2 \cdots a_k, b_1, b_2, \cdots b_p)^\top$. In a similar way, we can extend a k dimensional input vector i into a $k+p$ dimensional vector $i_e = (c_1, c_2 \cdots c_k, d_1, d_2, \cdots d_p)^\top$. We call the last p dimensions the environment code. In preferred directions, the environment code dictates the preferred environment of each neuron; in input vectors, the environment code keeps track of the current environment by staying the same in one environment.

We can constrain environment codes so that the dot product between the environment codes in the preferred direction of a neuron and the input vector (the last p dimensions of v_e and i_e) is zero when the input indicates the neuron's preferred environment and negative when it indicates other environments. Under this constraint, a neuron will fire as specified in the previous sections, unaffected by the extended dimensions, in its preferred environment. On the other hand, the neuron will be suppressed when it is in other environments, because the dot product between the environment codes are negative, lowering the firing rate of the neuron and producing rate remapping.

The environment code is different from other dimensions in several important ways. First, an environment code is only used to modulate neural activity, instead of being decoded. Since environment codes do not need to be decoded, they only increase little computational burden for neurons representing other information. In addition, as specified by the constraint, the dot product between the preferred environment code and the input environment code can not be positive (ignoring numerical errors and noises). Actually, this simple constraint can be satisfied by many different choices of the environment code.

3.5.2 Place field shifting

The shifting of place fields can be implemented through linear operations at different levels, which give different biologically plausible implementations (derived from equation 3.14 or equation 3.15). Here, we consider two implementations. First, assume f_M is the two dimensional function representing the preferred Gaussian function of a neuron. We directly discretize f_M to obtain the high dimensional vector M. From basic linear algebra, we can easily shift M to $\tilde{M}$ using a permutation matrix P:

$$PM = \tilde{M} \tag{3.12}$$

which can be rewritten by decomposing M into basis B and coefficients C and $\tilde{C}$:

$$PBC = \tilde{M} = B\tilde{C} \tag{3.13}$$

where $\hat{C}$ is the input vector giving the same activation of this neuron after shifting. Due to the combination rule of matrix products, the same shifted place field of a neuron can be obtained from either of the following transformations:

$$(PB)C = B\hat{C} \tag{3.14}$$

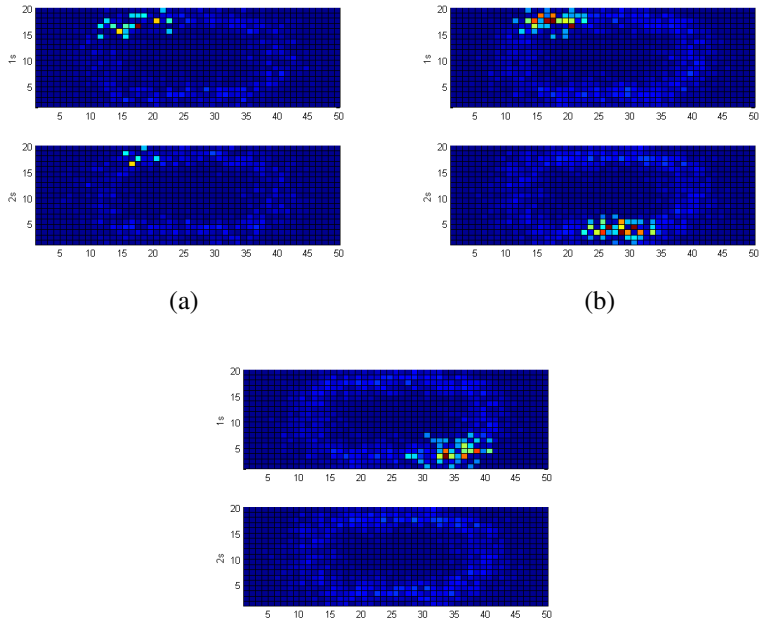
or

$$P(BC) = B\hat{C} \tag{3.15}$$

When B contains a Fourier basis, (PB) in equation 3.14 resembles shifting these oscillations, thus shifting the grid cell's firing patterns (section 3.2.1). On the other hand, rearranging equation 3.15 gives a transformation matrix T:

$$\hat{C} = (B^{-1}PB)C$$

$$T = B^{-1}PB \tag{3.16}$$



(a) (b)

(c)

Figure 3.16: Simulation of rate remapping (a) and global remapping (b, c). In each sub-figure, the two panels show the recordings from one simulated neuron in two different environments. The change of environments is controlled through the change of the environment codes (not plotted) as inputs into this population of cells.

$$\hat{C} = TC \tag{3.17}$$

From equation 3.17, multiplying input vectors by T, before feeding them to a population of neurons, gives the same effect as shifting the place fields. From equation 3.16, T has low dimensionality (the same as the number of basis used for spatial representation), so it is plausible to both store T and to compute the matrix product using T.

However, a problem in our proposed remapping mechanism is that they only partially simulate the effect of global remapping, since correlations between place fields of neurons in a population within one context are preserved after applying the transformation matrix. As a result, the place fields of all neurons are shifted in the same direction. Experimental studies suggest that the dentate gyrus may play an important role in remapping, especially in dissociating representation through different contexts [83]. Therefore, additional non-linear computations from regions such as the dentate gyrus may be essential for global remapping.

Using this two method, we simulated both rate and global remapping as shown in figure 3.16. The basic simulation settings are the same as in the experiment in section 3.4, with additional dimensions representing the environment codes. In the simulation of global remapping, transformation matrices specific to each environments are (equation 3.16) applied on the input vector, depending on the environment codes.

Chapter 4

Hippocampal replay

In this chapter, we explore the computational potential of the dense recurrent connections in the CA3 region of the hippocampus. Recurrent Neural Networks (RNN) initialized with random weights are trained by the state-of-the-art Hessian Free optimization algorithm (HF) to simulate both forward and backward replay. While this chapter focuses on replay itself, the relationship between replay and reinforcement learning is discussed in section 5.2.

4.1 The hippocampus as an information processing system

If we consider episodic memory as sequences of vectors (as described in the sections before), the anatomical structure (section 2.1) of the hippocampus gives some hints from an engineering view. When seen as an information processing system, this structure provides an ideal setup for time series processing, such as continuously predicting: while the local recurrent connections at the CA3 provide control for the whole system, the global feedback loops are able to continually feed the output of the system back as the input. In fact, researchers studying the hippocampus as a dynamic system have proposed that the CA3 with recurrent connections implements an attractor [71, 24]. However, these attractor models more often emphasize on the auto-completion of spatial patterns rather than temporal associations between patterns through time.

The simplified anatomical structure of the CA3 resembles a recurrent neural network (RNN, figure 4.1), governed by the following equations:

$$y_t = W_{hi} \cdot x_t + W_{hh} \cdot h_{t-1} + b_h \tag{4.1}$$

$$h_t = \tanh(y_t) \tag{4.2}$$

$$z_t = W_{oh} \cdot h_t + b_o \tag{4.3}$$

where x is the input, W_{hi} is the weight matrix connecting the input to hidden layer, W_{hh} is the weight matrix for the recurrent connections at the hidden layer, b's are biases, and

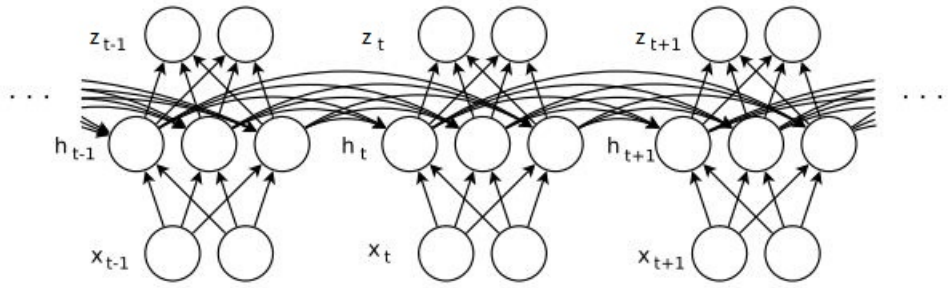


Figure 4.1: A recurrent neural network. *x* represent input units, *h* represents hidden units and *y* represents output units. The states of the hidden units depend on both current input and the last state of the hidden units.

z is the output. In theory, being hidden Markov models [69], RNNs are able to model complex dynamics of temporal sequences. Elman [26] compares RNNs and other neural network models that are used to model temporal sequences. He concludes from both theoretical reasoning and experimental simulations that RNNs can model complex temporal dynamics by representing the effect of time in their hidden layers, providing context information discovered from the structure of temporal sequences. Therefore, given that episodic memory can be seen as temporal sequences, it is reasonable to explore how such sequences can be stored and retrieved in RNNs, seeking insight into the computational potential of the recurrent connections in the hippocampus.

4.2 Training the RNN

Although the potential of RNNs is promising from the above discussion, the training of RNNs is notoriously difficult. The most basic algorithm is back-propagation though time (BPTT) [68, 74, 88, 89]. In general, BPTT unrolls a RNN, treating it as a multilayer neural network with the same number of layers as the number of time steps in the training sequences, while keeping weight matrices connecting layers the same. Unfortunately, even worse than the unsatisfactory performance of back-propagation in deep networks [40] (see also appendix B), training RNNs for temporal sequences with long temporal dependencies (more than 10 time steps) using BPTT is usually disappointing [8]. However, long temporal dependance is important given the length of a usual memory episode. In the analysis by [42], they conclude that the difficulties in training RNNs using gradient based methods mainly come from the "diminishing gradients" - the gradients used as a training signal usually either explode or diminish when propagating. Since step sizes in gradient based (first-order) methods are proportional to the gradients, it is thus very difficult to choose suitable step sizes (appendix A). Because of this fundamental problem, many previous algorithms for training RNNs are hardly better than random guessing [41].

A breakthrough came from a method called long short-term memory (LSTM)[42], which introduces additional structures to regulate gradients (figure 4.2). Specifically, local unit recurrent connections are used to ensure that the gradientwill neither explode

nor diminish. With additional structural complexity, the LSTM is able to learn very long temporal dependancies, and produces the state-of-the-art result in applications such as phoneme classification.

Another way to regularize gradients, without introducing any additional structure, is to use the second-order (curvature) information. The basic idea behind all second-order algorithms (e.g., Newton's method, quasi-Newton methods, the conjugate gradient method [10] and the Hessian free optimization [56] used in this thesis), is to calculate step sizes based on information from both first-order and second-order information. In general, when optimizing the parameters of a model in its parameter space, second-order algorithms take small steps when the curvature is large, where the gradient changes fast, and take large steps when the curvature is small, where the gradient changes slowly. Therefore, despite that gradients alone may still explode or diminish, the step sizes modulated both by gradients and curvatures usually remain stable.

Due to the broad range of techniques used in the Hessian free optimization algorithm (HF), here we only treat HF as a black box, optimizing the neural network based on given training inputs and targets. A self-contained introduction of HF covering most mathematical details is presented in appendix E.

4.3 Simulations of forward replay

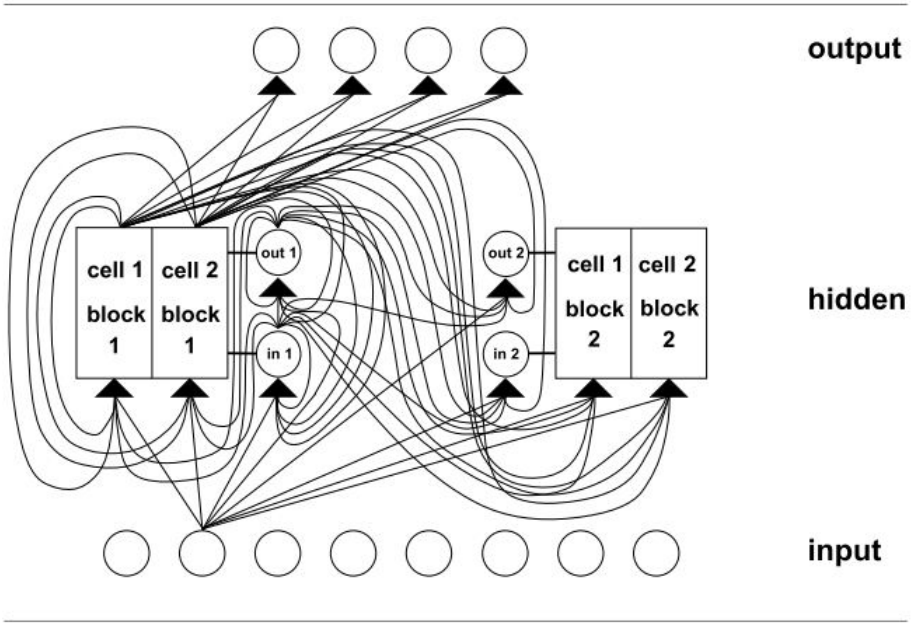
Our first goal is to simulate the forward replay (section 2.4). We use the following physical motion equation to generate random sequences in our experiments:

$$\frac{d^2x}{dt^2} = \frac{F_R}{m} \tag{4.4}$$

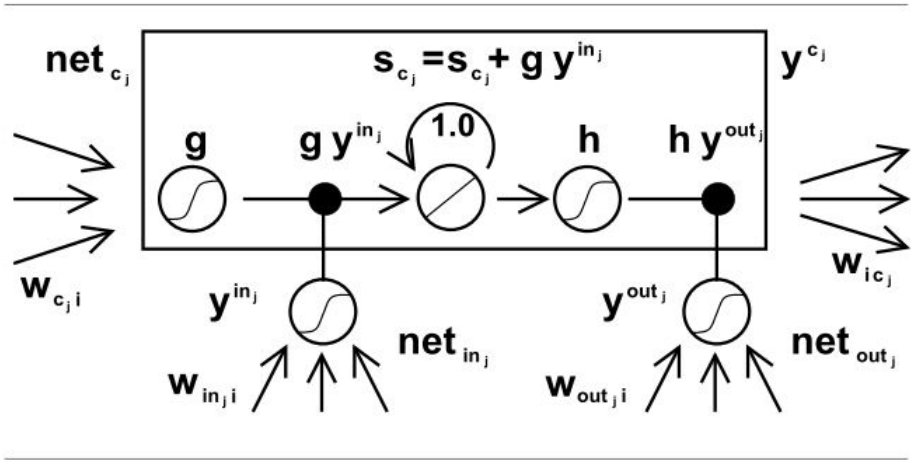
where vector x is the position of a moving particle in a n dimensional space with mass m at time t, and F_R is a random force driving this particle. The random force F_R allows automatically generating different random sequences, while the inertia from a non-zero mass m guarantees that some temporal correlation will be preserved. We can therefore regard traces of these moving particles as memory episodes in a n dimensional memory space.

Since the global feedback loop (section 4.1) of the hippocampus is continually sending the output of the RNN to its input, we only need to train the RNN to predict one step ahead. For supervised learning of temporal sequences, this means, in the training set, the target sequence needs to be one step ahead of the input sequence. In the brain, such a training set can be realized by introducing a delay at the input end. In addition, if the input sequence is incomplete, such that it contains only some of the input dimensions, the learning can still proceed. In this case, the RNN learns to complete the pattern (i.e. pattern completion) at each time step, in addition to predicting future time steps (figure 4.4).

We train RNNs with 150 hidden units, five input units and five output units with a batch containing all the sequences to be learned. Each of these sequences has 20 time steps and five independent dimensions generated from equation 4.4. We define the



(a)



(b)

Figure 4.2: A LSTM model [42] Compared with a standard RNN, a LSTM has more complicated architecture. The hidden units of the standard RNN are replaced by a combination of blocks of memory cells and input gates (a), and more gating units are contained in each memory cell block (b). The unit recurrent connection at the center of (b) maintains stable gradients.

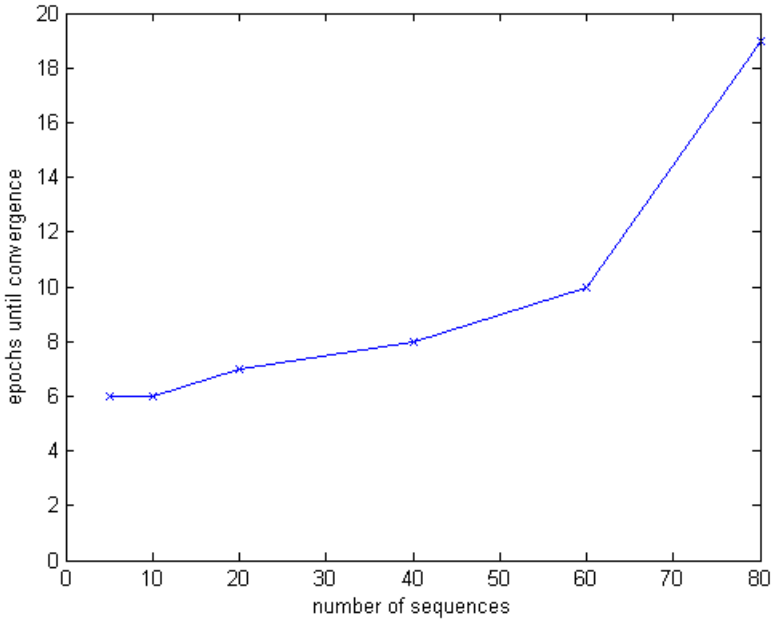


Figure 4.3: Epochs required for for different numbers of sequences with the length of 20 time steps.

training error averaged over all training cases and time steps as

$$e = \frac{1}{NT} \sum_{n,t} (z_t^{n*} - z_t^n)^2 \tag{4.5}$$

where z_t^n is the model output for training case n at time step t, and N, T are the total number of cases and time steps. The training is terminated when this error is below a specified tolerance, which is set to 3×10^{-4} in our experiments. The RNNs are initialized with random weights with -0.5 mean (see section 4.5). Since forward replay only requires the retrieval of trained sequences (i.e., the testing set is the same as the training set), the retrieval errors are the same as training errors. Therefore, we only summarize the epochs[1] required for training different numbers of sequences in figure 4.3. Since the numbers of epochs in HF are not precisely indicative of the training time (appendix E), we did not average these numbers over different runs.

The results (e.g., figure 4.5) show that the model can give continuous predictions of multi-dimensional temporal sequences from dynamic cues that are partial in both space and time. Hence, we obtain a spatio-temporal associative memory that is capable of both pattern completion and sequence learning.

In the experiments by Johnson and Redish [46], only the decoded spatial information from the recorded hippocampal cells is shown, as we noted in chapter 3. However, the replayed sequence may also contain other information that is helpful for the evaluation

[1]Depending on the local quadratic approximation, different epochs may take different time (appendix A, E).

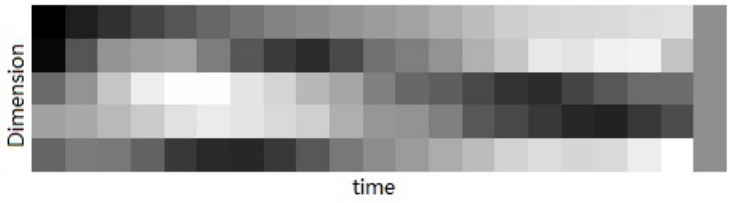
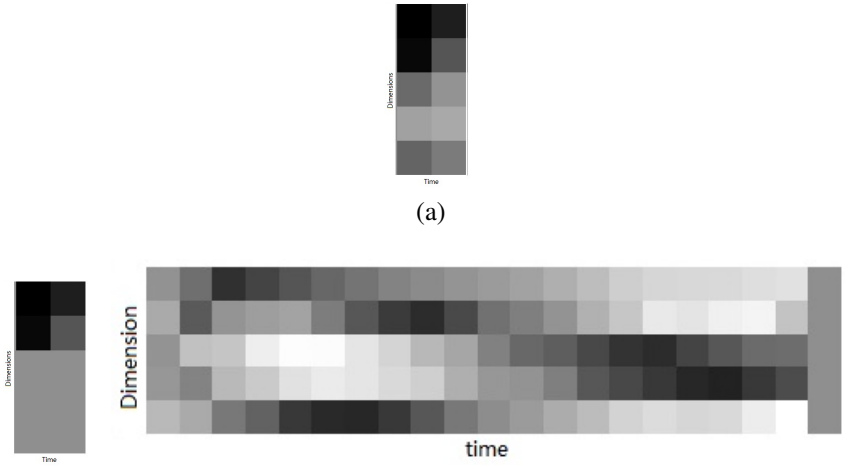


Figure 4.4: An example sequence for forward replay.Each row represents a dimension, and each column represents a time step; the time goes from left to right. This is a five dimensional random sequence generated by equation 4.4 that preserves certain temporal dependencies. In the most basic case, the training targets are the same as the input sequences one step advance, so that the RNN will learn to predict one step ahead.



(a)

(b) (c)

Figure 4.5: Input cues and output from the RNN. Figure (b) shows the first two steps of the sequence in figure 4.4, and (b) has only the first two dimensions (rows) of (a). After training, the RNN can recall the sequence (c) using either a full cue (a) or a partial cue (b). The first few time steps in (c) looks different from those in figure 4.4 and are blurred, since the RNN need to settle down in the cue stage. Subsequent steps match well to the learned sequence.

of future reward [86]. We thus assign the third dimension of the replayed temporal sequences (the third row in figure 4.6 (a)) a signal directly related to reward, so that the magnitude of this signal and the expected reward are directly correlated. Note this assumption is only used to illustrate how a rich representation in the memory can facilitate reward related computation. Figure 4.6 shows the simulation of the T-maze forward replay. At the upper-right corner of 4.6 (h), the light red color indicates that the reward signal is strong enough so that the rat will make the decision to go right.

4.4 Simulations of reverse replay

The RNN can be trained in a similar way for both forward (section 2.4) and reverse replay[2] (section 2.5), but problems arise when considering the functional role of reverse replay in reinforcement learning as well as experimental data. For example, since reverse replay was observed immediately after a rat reaches the goal after very limited training [21, 29], such a short time-scale is demanding for synaptic change-based recording of experiences. On the other hand, since the reversely replayed experience is usually short and recent, it may be well stored through persistent neural activity rather than synaptic weights. Therefore, the RNN for reverse replay is trained (or "pre-wired", to distinguish it from the training for forward replay) differently, so that no more training is required in later reverse replay.

For the reason discussed above, we wish the RNN to be able to replay any sequence it recently experienced[3]. Since reverse replay often happens after a goal or a reward is reached, we assume it is triggered by an additional input signal. An example of an input training sequence and its target is illustrated in figure 4.7. In order to ensure that the RNN is general enough to replay any input sequence (within a certain range), a training batch with a large enough amount of sample sequences is necessary. Accordingly, although the training for reverse replay takes place only once, it is significantly more difficult than the training for forward replay.

In our experiments, we train RNNs with 150 hidden units, two input units and one output units on 300 sequences similar to the one in figure 4.7. With other parameters the same as in the RNNs for forward replay, we train the RNNs until convergence.

As examples, the two plots in figure 4.8 show reverse replay for two random sequences with different length in the same RNN. Recall that this RNN is only trained for sequences with fewer than six time steps, so, interestingly, when sequences with greater than six time steps are presented, only the last few steps of the original sequences are replayed correctly (figure 4.9). Given that only recent experience need to be reversely replayed, such dynamics are plausible.

In the reverse replay experiments in this thesis, we trained the RNN to replay one dimensional sequences with fewer than 6 time steps. Furthermore, our experiment illustrates the extreme case in which the sequences are entirely random without any tempo-

[2]Since remote reverse replay (section 2.5) can be implemented in a way similar to the forward replay, by storing experiences in synapses, it is not discussed in this chapter.

[3]We trained different RNNs for forward and backward replay. Although they could be combined using gating mechanisms [10], we leave that to future work as it is not directly related to our current focus.

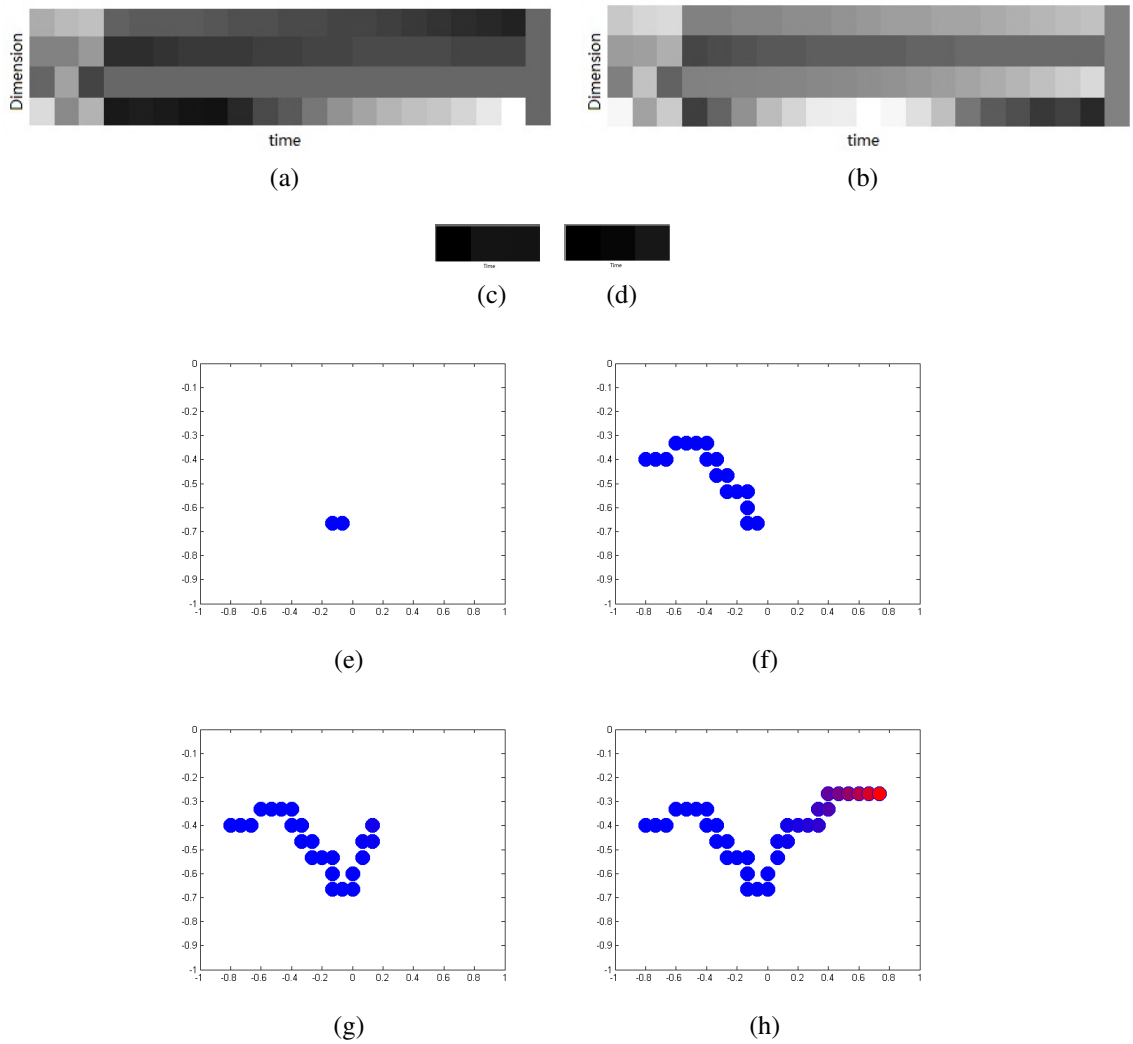
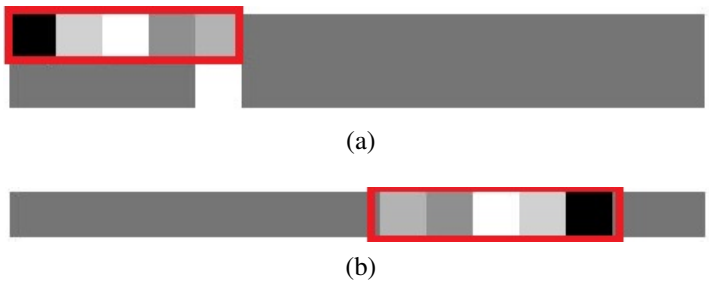


Figure 4.6: Simulation of decision-point forward replay. In (a) and (b), the two 4-dimensional sequences (plotted in the same way as in figure 4.4) represent the replay along the left and right arm respectively. In these two sequences, the first two dimensions represent the locations (coordinates) of the rat, the third dimension represents the reward related signal, where white means high reward. The fourth dimension are randomly generated (equation 4.4) and used as context cues to recall the whole sequences. Figures (c) and (d) show the two cues used to recall the two sequences in (a) and (b). They are the first three steps of the fourth dimension in the input sequences. Figures (e)-(h) plot the 4 time slices in process of replaying the sequences in (a) and (b) consecutively in a 2D panel. The location of each circle is specified by the first two dimensions, and the color of these circles indicates the strength of the reward signal in the third dimension (red means high reward).



(a)

(b)

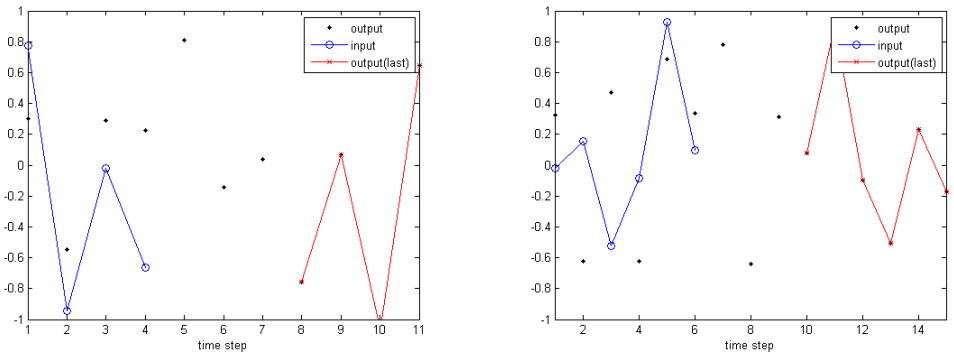
Figure 4.7: An example input sequence and its target for reverse replay. Figure (a) shows the two dimensional input, in which the first dimension represents events to be replayed and the second dimension controls the commencing of the reverse replay through a mark (the while square). Figure (b) shows the target (desired output) used in training, specifying the reversed order of the events (compare the two red squares). Errors occurring in the time steps outside the red square are ignored as irrelevant noise. Note that the sequence is completely random, exemplifying the extreme case of no temporal correlation at all.

ral correlation, although in the real world this is unlikely. In the brain, some synaptic changes may take place before reverse replay [21, 29], which could tailor the neural networks for specific task structures. In summary, the training (or pre-wiring) for reverse replay in the brain could be significantly easier than in our experiment by taking advantage of more specific temporal structure. Nevertheless, our results show that the RNN is powerful enough for a fairly general case.

To demonstrate the overall performance of reverse replay, we trained two RNNs with exactly the same procedures, testing them with random sequences at different lengths, and calculating the averaged mean square errors over 200 samples for each length. When calculating the errors we only consider at most the last 6 steps in the original experiences. Figure 4.10 shows the averaged errors for the 2 RNNs. For both RNNs the replay errors are bounded around 0.1, which confirms the performance of long sequences shown in figure 4.9.

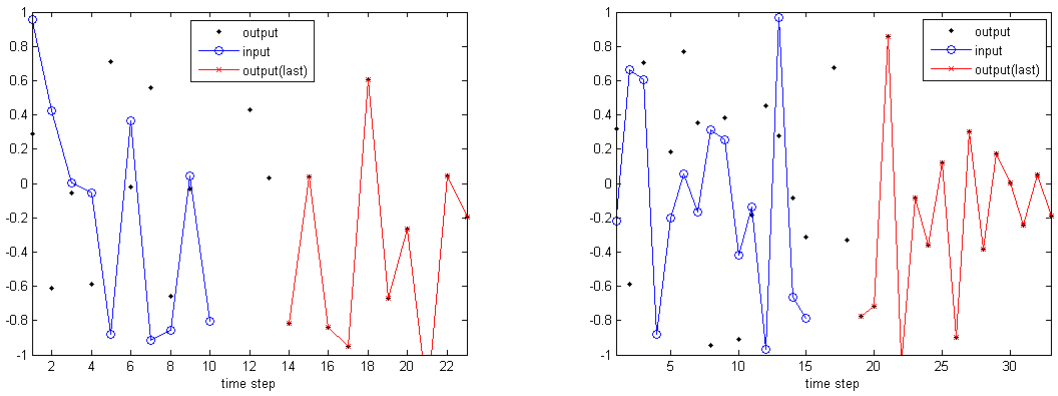
The fundamental difference between RNNs for forward and backward replay is that experiences are stored in synaptic weights for forward replay, but in neural activity alone for reverse replay. On the one hand, forward replay needs longer time to learn because of the required synaptic change, but this brings the benefit that once learned, the sequence can be recalled in a later time as long as the synaptic weights are not altered. On the other hand, reverse replay without synaptic change is fast, at the cost of being more vulnerable to interference – any alternation of the neural activity may affect the reverse replay. This trade-off between time and stability implies their different functional roles, which is worth further investigation.

Based on our simulation results, we predict that reverse replay does not require synaptic change. This prediction is experimentally testable through injection of NMDA blockers, which obstruct synaptic change, before testing trials or before training. In our prediction, the injections before testing trials will not affect reverse replay, but the injection before initial training in a new environment may disrupt it.



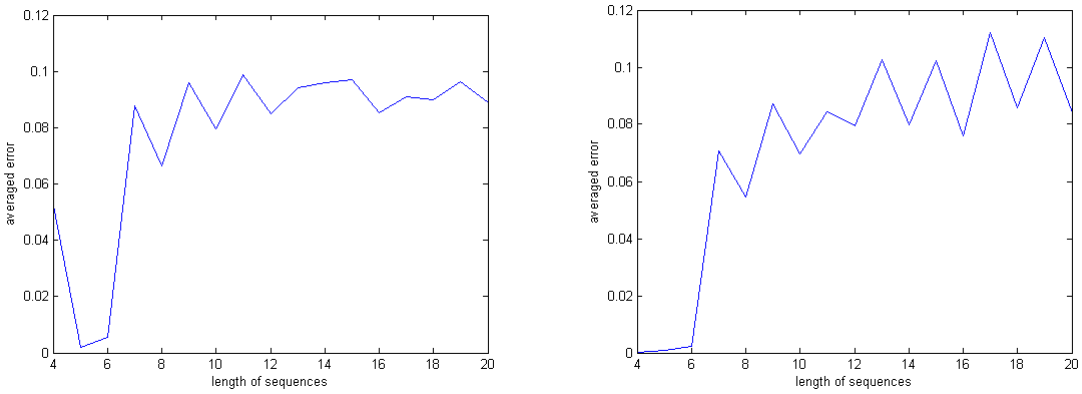
(a) (b)

Figure 4.8: Simulation of reverse replay of 1D sequences, where the vertical axis represents values at each time step. Figure (a) shows the reverse replay of a 4 step sequence, and (b) shows the reverse replay of a 6 step sequence from the same RNN. The symmetry between the blue lines (experience) and red lines (output) shows the replay is successful even in our extreme examples, although with observable distortions. Black dots outside of the red line are irrelevant output noise (output before the start of reverse replay).



(a) (b)

Figure 4.9: Simulation of reverse replay for long sequences. Since the RNN is trained for sequences with length up to 6 time steps, only the last few steps of the original sequence are replayed correctly, when a much longer sequence is used as input.



(a) (b)

Figure 4.10: Averaged reverse replay error for sequences with different lengths. Figure (a) and (b) show the performance of reverse replay on two RNNs trained with the same settings on random sequences with lengths of 4, 5 and 6 time steps. The mean square errors in these figures are average over 200 testing examples. These figures show that replay errors for recent experience are bounded for different lengths of input sequences.

4.5 Biased Initial Parameters and Oscillations

An unexpected finding in our experiments is the discovery of oscillations in the hidden layer. The presence of oscillation has intriguing relationship with the performance of sequence learning. Given recent research about functions of oscillations observed in animal brains (e.g., [37]), this discovery may provide us insights into the computation involved in episodic memory.

An important step before training a neural network is initializing connection weights[4]. Normally, the initial weights are drawn from a random distribution with zero mean, which permits unbiased optimization in the whole weight space. However, we observed a significant boost of the performance when the initial weights are biased with a negative mean. Table 4.1 summaries the relationship between the number of converging epochs and weight biases. For RNNs trained for the same set of sequences, properly biased initialization reduces required training epochs by about 30%, compared with the conventional zero-mean initialization. In addition, the biased initialization nearly triples the capacity of RNNs[5]. However, a too biased initialization harms the performance. We found -0.5 to be roughly the optimal mean in our experiments. Although it is not entirely clear why lowering the mean of initial weights dramatically facilitates learning, the overall inhibition effect created by negative weights may help reduce interference between patterns.

Moreover, we observed interesting oscillatory patterns in the hidden layer activities

[4]Although the parameters of neural networks include both connection weights and biases, biases can always be treated as weights connected with units with unit activity. For easy comparison with biology observations, we use the term "connection weights" or "weights" instead of "parameters".

[5]Here the capacity means the number of statistically independent sequences can be learned.

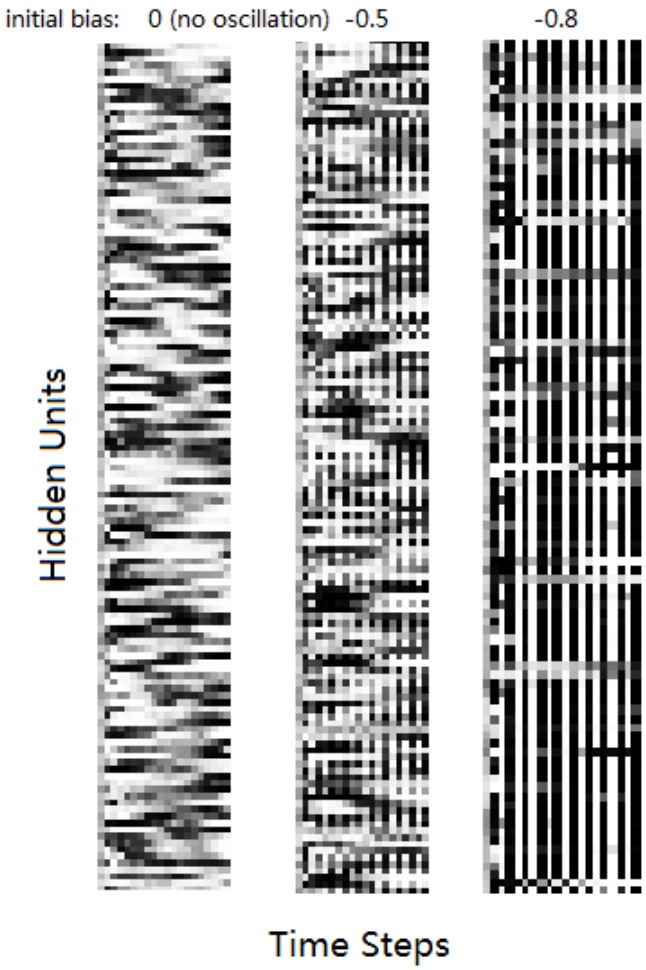


Figure 4.11: Temporal oscillation patterns in the Hidden Layer of RNNs. The three columns represent the activity of hidden units from three RNNs respectively, when recalling the same remembered sequence. The three RNNs have the same structure and were trained with the same sequences, but their weights were initialized with random distributions with different mean. The first has zero mean, the second has a mean of -0.5, and the third has a mean of -0.8. In each of the three columns, each row of pixels represent a hidden unit, and each column of pixels represents a times step. The gray level of a pixel represents its activity.

Mean of Weights	Number of Epochs
0	43
-0.2	30
-0.5	29
-0.8	42*

Table 4.1: Sequence learning performance and initial weights.This table compares the number of epochs needed for RNNs with the same structures to converge on the same training set (18 sequences with 20 time steps). All of these RNNs have 150 hidden units, 5 input units and 5 output units. The RNN in the last row failed to converge with very negative initial weights. Further experiments showed that it takes 42 epochs for the RNN in the last row to converge for a training set with only half the number of sequences. Therefore, the performance of the RNN in the last row is worse than the one with zero-mean initial weights.

that are correlated with the mean of initial weights (figure 4.11). In general, more negative weights produce clearer oscillation patterns. These patterns can be explained by the overall inhibitory effect of the neural network: when the hidden units are highly activated in one time step, strong inhibition tends to mute them in the next time step. Then, the released inhibition allows input to drive these hidden units to a high activated state again. The oscillations may promote synchronized activity, although this remains to be seen.

Interestingly, these oscillations can clearly distinguish whether an input sequence has been learned before. In figure 4.12, the oscillation patterns are largely unseen when the input is a novel sequence. Analysis of the learned weight matrix shows that connection weights between hidden units are highly asymmetric, which may be responsible for such sequence sensitive dynamics. The results suggest that the oscillations might be important for sequence learning, but the exact reason is unclear.

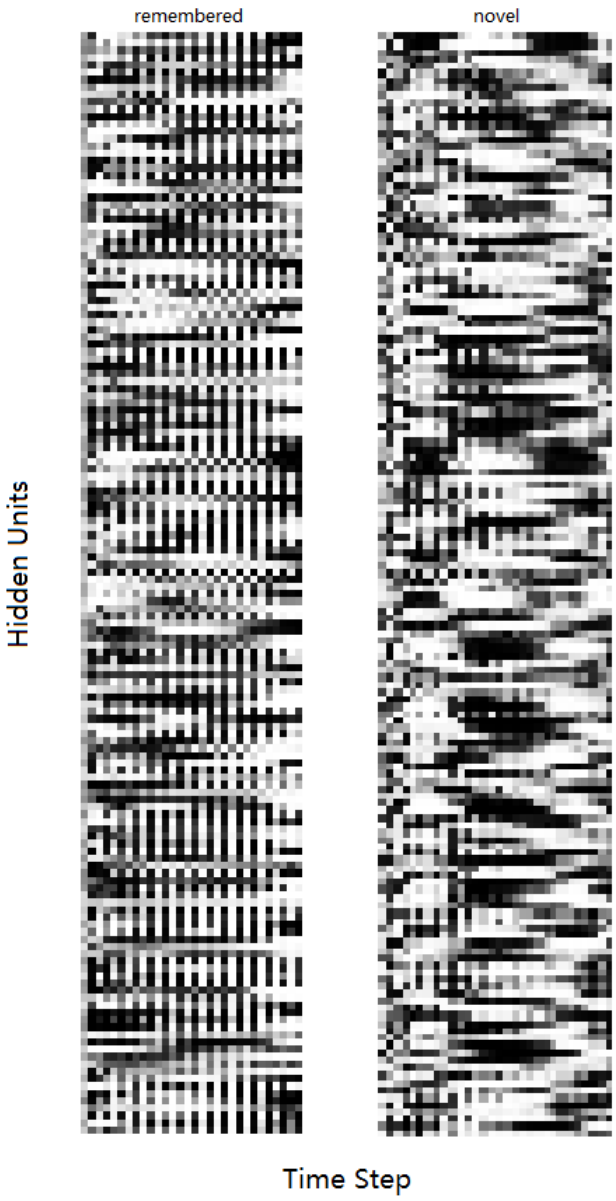


Figure 4.12: Hidden unit states can distinguish novel sequences. When the RNN is initialized with biased weights that give the optimal performance, states of the hidden units clearly distinguish between a remembered sequence (left) and a novel sequence (right).

Chapter 5

Discussion

The interactions between hippocampus and other brain areas, especially the neocortex are necessary in understanding brain functions such as various kinds of learning. However, limited by the scope of this thesis, our model mainly addresses the hippocampus itself. This chapter briefly outlines how our hippocampus model can be extended to shed light on more general issues in neuroscience.

5.1 The hippocampus and the neocortex

The interaction between the hippocampus and the neocortex is crucial in coordinating the brain's memory system. Classical computational models of memory systems [65, 59] suggest that the hippocampus and the neocortex play complementary roles [81].

In the framework proposed in [59], and further developed by [65], the hippocampus rapidly learns experienced input as episodes (episodic memory), which later slowly consolidates into the neocortex. In their proposal, an important purpose of the memory system is to learn the structural relations from patterns in experiences, forming knowledge that is able to benefit an animal in various ways. Although the neocortex is supposed to be mainly responsible for learning such structural relations, from both psychological experiments and simulations using artificial neural networks, this learning process has to be slow in order to both learn deep structural relations associating different patterns and not to disrupt previously learned associations. This slow learning rate makes the neocortex incapable of learning directly from experiences without the help of a buffer system that rapidly stores experiences in the first place. The hippocampus provides such a temporal buffer for the neocortex, by quickly storing experiences without much processing (see also [55, 90]).

The different functional roles of the hippocampus and the neocortex propose distinct structural requirements. In addition to a slow learning rate in the neocortex, a hierarchical structure is also helpful in discovering higher order relations (e.g. [40, 74]). On the other hand, as discussed before, the hippocampus has to be able to learn quickly, and a deep structure is not necessary. However, recurrent connections are necessary to form temporal associations across a long time-span, as demonstrated in 4.

Moreover, the representations required by the hippocampus and the neocortex seem

to be fundamentally different. For the neocortex, distributed representations are beneficial because it is easier to discover associations between different patterns from overlapped representation brought about by distributed representations [39]. However, since the main task of the hippocampus is to store near "raw" experiences as memory episodes, or a sequence of patterns, the main priority is to reduce the interference between different episodes. As a result, instead of being distributed, the neural representations in the hippocampus are sparse, i.e., only a very small portion of hippocampal neurons are active at the same time, so that different patterns are more likely to be represented by different neurons. In our model, we use vectors representing functions with a specific shape (e.g., Gaussian function) to enforce the sparsity (chapter 3); in the hippocampus, the sparsity constraint due to both similar representation schemes and the interactions between dentate gyrus and CA3 that are not yet well known[61].

Both sparse firing patterns and remapping contribute to the broader mechanism of pattern separation. While different patterns will be represented distinctly under the sparse firing of hippocampal neurons, patterns that are similar but occur in different contexts, will also be represented differently because of remapping. As discussed earlier, the dentate gyrus, a part of the hippocampal formation sending input to the CA3 region of the hippocampus, is one of the only two brain regions that are able to generate new neurons (neurogensis) in adult mammals [2]. Adult neurogenesis in the dentate gyrus may play an important role in pattern separation by further orthogonalizing the input from the entorhinal cortex [1, 7, 83]. Together, the sparsity of firing and remapping provide the basis for pattern separation that to a large extent guarantees experiences in the memory will not interfere with each other [50].

Interestingly, pattern completion, a function that is also important for episodic memory and demonstrated by experiments, functionally contradicts pattern separation [32, 65, 73]. Through pattern completion, input that that only partly matches the original experiences may give rise to the same neural activities, i.e., the same episodic memory [63]. In this way, pattern completion allows more robust and flexible representations, as incomplete input patterns that are possibly caused by noise or changing environment may be recovered. Pattern completion can be implemented as attractors in computational models [16, 24, 71]. Because pattern separation and pattern completion are competing demands when an input pattern comes, a decision needs to be made concerning whether the pattern should be stored as a new pattern (pattern separation) or rather be used as a cue to access previously stored patterns (pattern completion). This decision may be reflected in different states of the hippocampus that are governed by a combination of neural oscillations and other neurophysiology processes [61].

5.2 The hippocampus and reinforcement learning

Both forward and reverse replay may have a close connection with reinforcement learning. As discussed in [86], in forward replay (section 2.4), the hippocampus may support a forward model generating informative expectancies of the agent's (e.g. a rat's) future states. The co-activation of cells in ventral striatum signaling reward at the time of replay (or sweep) implies that the expectancies were evaluated by the ventral striatum, which

may implement the critic in the Actor-critic architecture. Given the rich information that can be encoded in the state space (as in the memory space, chapter 3), model-based learning based on this forward model should support flexible behaviors. The ability to incorporate both external and internal information in the state space permits learning of decision-making tasks modulated by latent variables such as internal motivation.

In contrast, since no synaptic change is required, reverse replay introduced in section 4.4 enables fast association between reward and actions in recent past. The relatively short time-scale of reverse replay compared to forward replay may well support the computation needed by temporal difference learning [82]. Due to the difficulties involved in training (or pre-wiring) a RNN for fast reverse replay (section 4.4), both the length and complexity of the replayed sequence is limited. Therefore, we suggest that, unlike forward replay that generates expectancies with rich information about the environment, reverse replay only associates significant actions and values. From this point of view, reverse replay may better support model-free reinforcement learning.

For the more general control problem, the hippocampus may be involved in all three kinds of controls: model-based, caching-based and episodic control [18, 48, 49]. However, different control strategies may be implemented through different mechanisms in the hippocampus (e.g., forward vs. reverse replay), so the transition between different control strategies becomes an additional issue [18]. In addition, especially for the model-based control, a full forward model requires the coordination between different brain areas in addition to the hippocampus [19] (e.g., [44]), so a more comprehensive memory model incorporating semantic memory is likely required to better understand the interaction between memory and reinforcement learning [59, 73, 47, 19].

5.3 Comparison with other models

To our knowledge, apart from our model, only [37] addressed both multi-modal representations and sequence learning. In this very recent model, Hasselmo unifies nearly all experimental findings about the hippocampus, from the molecular to system level. Specifically, the central role of entorhinal cortex in his model gives insights into oscillations in the hippocampus. Based on different model design choices, his biologically realistic model has physiologically grounded accounts of inputs from different modalities, rather than a unified representation scheme at a higher level as in our model. However, restricted by the biologically plausible Hebbian learning rules that is impossible to take advantage of second-order information, as well as the seemingly under-utilized CA3 recurrent connections, his model is computationally less powerful than ours, which means our model can encode more complex and longer memory episodes.

Another relatively comprehensive hippocampus model comes from Levy and colleagues [51]. Levy's model simulated high-level hippocampal functions including replay from biologically plausible neural networks trained by elegant biologically plausible learning rules specifically derived for learning associative memory [52]. In addition to the hippocampus itself, his model also addresses interactions between the hippocampus and the neocortex, proposing two forms of sequence compression mechanisms (on-line and off-line compression). However, he uses ad-hoc style context and inhibitory neurons

that are neither biologically nor mathematically well justified. Similarly restricted by the biologically plausible Hebbian learning rules, his model only demonstrates the replay of rather simple sequences [5].

Chapter 6

Conclusions

Since the proposal of the hippocampus encoding a generic "memory" space [22], researchers have generally accepted that the hippocampus can encode information from different modalities. However, only a few computational model to date address this issue (e.g., [37]). Our model, using the Neural Engineering Framework (NEF), is the only one to our knowledge incorporating multi-modal information in a unified and systematic manner. Our systematic approach makes the behavior of the model easy to characterize by identifying specific mathematical operations. For example, neural integrators can be used to control temporal dynamics of the model representations [24], and transformation matrices can be constructed to account for the shift of place fields.

Based on this systematical representation scheme, we further explored the dynamics supporting storage and retrieval of sequences of vectors as episodic memory. Compared to previous models based on gradient-descent-like Hebbian rules (e.g., [87, 5, 37]), we benefit from the state-of-the-art Hessian-Free Optimization algorithm (HF) [57, 58], resulting much more powerful temporal information processing. Therefore, it is currently the only hippocampus model able to replay long multi-dimensional sequences, supporting the hypothesis proposed in [86] from a computational perspective, linking reactivation of hippocampal activities to reinforcement learning problems. In addition, this is also the first model demonstrating a RNN can replay recent temporal sequences using persistent neural activities only. Based on these results, we make a testable prediction that the forward and reverse replay are supported by different mechanisms, in which only forward replay (and possibly "remote" reverse replay [17, 33]) requires synaptic change.

As the first adaptation of the HF algorithm in computational neuroscience, this thesis provides an additional reference for challenging machine learning problems related to RNNs, and is of merit methodologically in neuroscience research. In order to better introduce the powerful HF method, the author derived most related techniques in the appendix part of this thesis, aiming to present this algorithm in a intuitive and relatively easy to understand way without losing necessary mathematical rigor. This self-contained text on HF is another contribution of this thesis.

Due to time constraints, we have not fully integrated the representation and sequence processing parts of the model. However, they are entirely compatible: they share the same vector representations, and estimation of gradients can be applied to train neural networks in NEF (Appendix B). In the future, we expect to continue exploring interac-

tions between parts of a wider memory system and the ways in which memory systems are used in reinforcement learning.

Appendix

Appendix A

Quadratic functions

Quadratic functions provide good approximations to other higher-order functions, while remaining simple enough to maintain the good properties we shall soon discuss. As a result, they are used as a basic case in our following derivations of the back-propogation and the conjugate gradient method.

This appendix introduces some basic properties of quadratic functions that will be helpful in understanding general optimization methods. A quadratic function, $f(x)$: $\mathbb{R}^n \to \mathbb{R}$, has the following form:

$$f(x) = c + bx + \frac{1}{2}x^\top Ax \tag{A.1}$$

Despite its simple form, when equation A.1 is used as the second-order approximation (e.g., from Taylor's expansion) of a higher-order function, the curvature information is well preserved in matrix A. The quadratic function has gradients

$$g(x) = \nabla f(x) = b + Ax \tag{A.2}$$

Equation A.2 holds true only when A is symmetric (otherwise $\nabla f(x) = b + \frac{1}{2}(A + A^\top)x$). Here it is reasonable to assume the symmetry, because when used for approximation, matrix A (either a Hessian matrix or an approximation of Hessian such as a Gauss-Newton matrix) is always symmetric.

A symmetric matrix has the nice property of having a full set of orthonormal eigenvectors:

$$Au_i = \lambda_i u_i \tag{A.3}$$

$$u_i \cdot u_j = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}, \text{for} i, j = 1, 2, ...N \tag{A.4}$$

where A is a $n \times n$ symmetric matrix, u are eigenvectors and λ are eigenvalues. Because of the orthogonality, the N eigenvectors span the whole N dimensional space.

Assuming that $x^*$is the point where the gradient $\nabla f(x)$ of this quadratic function is zero, we expand equation A.1 at x^*:

$$f(x) = f(x^*) + \frac{1}{2}(x - x^*)^\top A(x - x^*) \tag{A.5}$$

The first-order term disappeared because the gradient at this point is zero.

If A is positive semi-definite,

$$v^\top Av \geqslant 0 \tag{A.6}$$

is true for any vector v. Therefore, $f(x)$ has the minimal value of $f(x^*)$ when A is positive definite. If A is negative semi-definite, i.e., $v^\top Av \leqslant 0$, x^* is the maximal point. If A is indefinite, in which case the sign of $v^\top Av$ can be both larger or smaller than zero, $f(x)$ forms a saddle plane. In a typical optimization problem that aims to minimize an objective function, A being positive semi-definite is more favorable, because the objective function then has a global minimum. Function $f(x)$ is called a convex function when A is positive semi-definite.

To further analyze equation A.5, we can write $(x-x^*)$ as a linear combination of the eigenvectors of A:

$$x - x^* = \sum_i \alpha_i u_i \tag{A.7}$$

Recall that u_i are orthonormal basis (equation A.4), so there exists one and only one set of coefficients α's that satisfies equation A.7. Substituting equation A.7 into equation A.5 and using equation A.3 we obtain:

$$f(x) = f(x^*) + \frac{1}{2}\sum_i (\alpha_i u_i) \sum_j \lambda_j (\alpha_j u_j) \tag{A.8}$$

which can be further simplified using the orthonormal property of u's into:

$$f(x) = f(x^*) + \frac{1}{2}\sum_i \lambda_i \alpha_i^2 \tag{A.9}$$

Comparing the second term of equation A.9 and equation A.6, it is easy to find that all the eigenvalues of positive semi-definite matrix need to be non-negative. Therefore, the contours of $f(x)$ form ellipses with axes aligned with the eigenvectors, and the length of these axes are inversely proportional to the square roots of the eigenvalues.

Having a global minimum (for positive definite A matrix, or minima for positive semi-definite A matrix) makes gradient-based optimization methods plausible on convex quadratic functions, as least in theory, because when the step size is small enough, we can always find the the global minimum by going down the gradient. However, equation A.9 reveals that for a point at the surface of $f(x)$, the gradient along each eigenvectors can be vastly different if the magnitudes of eigenvalues are distinct (the ellipsis is elongated). In which case, convergence of gradient based methods become difficult[1]. A condition number

$$c = \frac{\lambda_{max}}{\lambda_{min}} \tag{A.10}$$

is used to measure the convergence property of a function. A low condition number indicates this function is easy to converge, which is called well-conditioned. On the contrary, large condition numbers mean functions are bad-conditioned. In the later case, preconditioning may be used to lower the condition number.

[1]The intuitive reason is that we usually use a slower learning rate for large gradient to avoid overshooting, and use a faster learning rate for small gradient to accelerate learning. However, if the gradients along different directions are very different, it is hard to find a learning rate that suite both.

Appendix B

Back-propagation in NEF

In this appendix, we present detailed derivation of the back-propagation algorithm in NEF. A two-layer toy neural network is used as an example to illustrate how parameters of NEF models can be learned through back-propagation of errors. Following this example, possible generalization of this derivation, including back-propagation through time (BPTT), is briefly discussed. To make this part self-contained, necessary equations appearing in the main text are repeated. Since the neural networks in NEF can be regarded as an extension of traditional artificial neural networks with additional notations to characterize the heterogeneity and multi-dimensional responses, this derivation also applies to simpler neural networks that are common in machine learning.

B.1 Notations

Figure B.1 shows the toy neural network for which we will derive learning rules for all its parameters. This neural network maps an input vector $X = (x_1, x_2, ..., x_i, ...)^T$, through the neural dynamics of two layers of LIF neurons, $p_1, p_2, ..., p_j, \cdots$ and $q_1, q_2, ..., q_k, ...$, to an output vector $Y = (y_1, y_2, ..., y_l, ...)^T$.

Given a constant input current, the firing rate$a$ of a LIF neuron is determined by its activation function $G(J)$, as described inB.1. Parameters τ^{ref}, τ^{RC}, and $J_{threshold}$ are hold

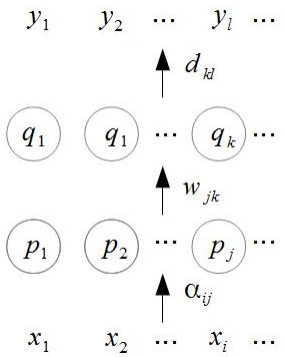


Figure B.1: A toy neural network

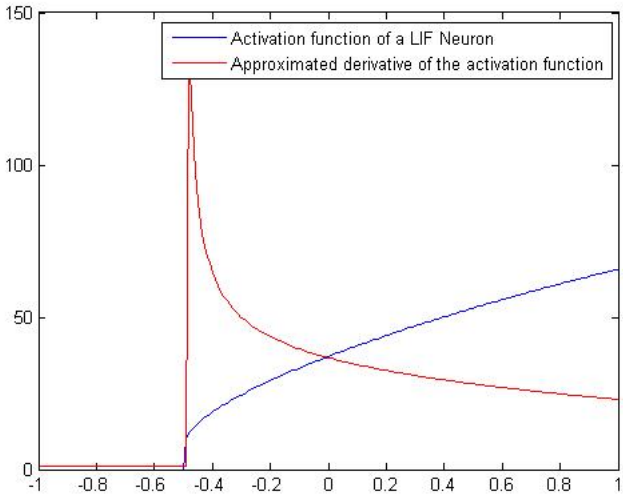


Figure B.2: The activation function $G(J)$ of a LIF neuron and it's approximated first-order derivative $G'(J)^*$.

constant in the derivation [1].

$$a = G(J) = \frac{1}{\tau^{ref} - \tau^{RC} ln(1 - \frac{J_{threshold}}{J})} \tag{B.1}$$

In order to use this activation function in the back-propagation algorithm, we need its first derivative. Although we can write the analytical form of $G(J)$'s first derivative as in equation B.2, note that $G(J)$ is not differentiable at the threshold point, where $\lim_{J \to J_{trheshold}+} G'(J) = \infty$. As a result, we need to approximate $G'(J)$ by $G'(J)^*$that generally preserved the shape of $G'(J)$ (figure B.2), as shown in equation B.3(ξ and ϕ are small positive constants)

$$\begin{aligned} G'(J) &= \frac{\tau^{RC} \cdot J_{threshold}}{(1 - \frac{J_{threshold}}{J}) \cdot J^2} / D^2 \\ D &= \tau^{ref} - \tau^{RC} ln(1 - \frac{J_{threshold}}{J}) \end{aligned} \tag{B.2}$$

$$G'(J)^* = \begin{cases} G'(J) & J \geqslant \xi \\ \phi & J < \xi \end{cases} \tag{B.3}$$

For neurons receiving external input directly, such as neuron p_j, the input current is given by the equation below, where e_j is the encoder with the same length as the input vector X, and b is the input bias of neuron p_j. If we view the bias term as a neuron that always has unit activity, we can see the bias magnitude as the weight of connection to

[1] In most simulations, these parameters remain constants. We can learn them through calculating the gradients of$G(x)$ w.r.t. these parameters, but the process is not given in this appendix.

this unit activity neuron. Therefore, The bias term can be learned the same way as other connection weights. For this reason, we will not derive separated learning rule for bias terms.

We can simplify it by combine the gain factor $g_j$and an element of e_j to form gain factors, α_{ij}, for each element in the input vector, thus gives equation B.5. The encoder and the gain factor can be easily factored out after learning given the magnitude of either of them.

$$J_j^p = \gamma_j \cdot e_j^T \cdot X + b_j^p \tag{B.4}$$

$$J_j^p = \sum_i \alpha_{ij} \cdot x_i + b_j^p \tag{B.5}$$

For neurons receiving input from other neurons, their input currents are given by weighted sum of incoming activities. For example, the input current of neurons q_k is described by equation B.6, where w_{jk} is the connection weights between neuron $p_j$and q_k.

$$J_k^q = \sum_j w_{jk} \cdot a_j^p + b_k^q \tag{B.6}$$

The output, Y, is decoded from linear combination of neural activities in the q layer. This decoding process is governed by decoders, d_{kl}, as described in equation B.7

$$y_l = \sum_k d_{kl} \cdot a_k^q \tag{B.7}$$

To sum up, parameters of this toy neural network are: gain factors α_{ij}, connection weights w_{jk}, and decoders d_{kl}.[2] Given a set of desired outputs Y^* The goal of supervised learning in this toy neural network is to find proper parameters so that the model outputs Y are as close to Y^*as possible.

B.2 The Back-propagation Algorithm

Now we are going to derive the back-propagation algorithm for each parameter introduced in the last section. The algorithm can be divided into forward pass and backward pass. In the forward pass, we run through the neural network by applying the equations described before, obtaining a model output Y. Then we can calculated the output error by comparing it with the desired output Y^*, and back-propagate this error to places that need error signal to facilitate training.

More formally, learning is achieved through optimizing, maximizing for minimizing, an objective function. The objective function, for minimizing, of the supervised learning problem described above can be written as equation B.8. To make model outputs approach desired outputs, the first term is the sum of the squared error for each output element across each training case c. Since large connection weights can make gradient

[2] As explained before, we treat bias terms as connection weights.

descent unstable, we also add the second term, the sum of all connection weights, to penalized large connection weights. In the following derivation, we will drop the summation of cases to simplify the notations.

$$E = \frac{1}{2}\sum_c\sum_l (y_l^c - y_l^{c*})^2 + \frac{1}{2}\lambda\sum_j\sum_k (w_{jk})^2 \tag{B.8}$$

To determine how E is affected by each element of Y, we take the partial derivative with respect to y_l:

$$\frac{\partial E}{\partial y_l} = y_l - y_l^* \tag{B.9}$$

Given equation B.9 and equation B.7, we can calculate the partial derivative w. r. t. the decoder d_{kl} using the chain rule:

$$\frac{\partial E}{\partial d_{kl}} = \frac{\partial E}{\partial y_l}\cdot\frac{\partial y_l}{\partial d_{kl}} = (y_l - y_l^*)\cdot a_k^q \tag{B.10}$$

The effect on E from the activation of a_k^qon y_l is:

$$\frac{\partial E}{\partial y_l}\cdot\frac{\partial y_l}{\partial a_k^q} = (y_l - y_l^*)\cdot d_{kl} \tag{B.11}$$

Since a_k^q acts on all y's, to take into account all the decoders interact with neuron q_k:

$$\frac{\partial E}{\partial a_k^q} = \sum_l \frac{\partial E}{\partial y_l}\cdot\frac{\partial y_l}{\partial a_k^q} = \sum_l (y_l - y_l^*)\cdot d_{kl} \tag{B.12}$$

Then we can then use the approximated gradient, equation B.3, to compute $\frac{\partial E}{\partial J_k^q}$:

$$\begin{aligned}\frac{\partial E}{\partial J_k^q} &= \frac{\partial E}{\partial a_k^q}\cdot\frac{\partial a_k^q}{\partial J_k^q} \\ &= \sum_l (y_l - y_l^*)\cdot d_{kl}\cdot G'(J)^*\end{aligned} \tag{B.13}$$

Similarly, the following equations can derive by applying the chain rule and previous derived intermediate results:

$$\begin{aligned}\frac{\partial E}{\partial w_{jk}} &= \frac{\partial E}{\partial J_k^q}\cdot\frac{\partial J_k^q}{\partial w_{jk}} + \frac{\partial \frac{1}{2}\lambda\sum_j\sum_k (w_{jk})^2}{\partial w_{jk}} \\ &= \sum_l (y_l - y_l^*)\cdot d_{kl}\cdot G'(J)^*\cdot a_j^p + \lambda w_{jk}\end{aligned} \tag{B.14}$$

$$\begin{aligned}\frac{\partial E}{\partial a_j^p} &= \sum_j \frac{\partial E}{\partial J_j^q}\cdot\frac{\partial J_j^q}{\partial a_j^p} \\ &= \sum_j\sum_k (y_l - y_l^*)\cdot d_{kl}\cdot G'(J)^*\cdot w_{jk}\end{aligned} \tag{B.15}$$

$$\begin{aligned}\frac{\partial E}{\partial \alpha_{ij}} &= \frac{\partial E}{\partial a_j^p} \cdot \frac{\partial a_k^c}{\partial \alpha_{ij}} \\ &= \sum_j \sum_k (y_l - y_l^*) \cdot d_{kl} \cdot G'(J)^* \cdot w_{jk} \cdot x_i\end{aligned} \tag{B.16}$$

In equation B.14, λdetermines the penalty for large weights.

B.3 Learning Rules

Using the derivatives given by equation B.10, B.14, and B.16, the learning rules for all parameters in our toy neural network can be written:

$$\Delta d_{kl} = -\varepsilon \frac{\partial E}{\partial d_{kl}} \tag{B.17}$$

$$\Delta w_{jk} = -\varepsilon (\frac{\partial E}{\partial w_{jk}} + \lambda w_{jk}) \tag{B.18}$$

$$\Delta \alpha_{ij} = -\varepsilon \frac{\partial E}{\partial \alpha_{ij}} \tag{B.19}$$

In the above equations, ε is a small positive constant denoting the learning rate. We usually introduced momentum to accelerate the speed of gradient descent, such as:

$$\Delta w_{jk}(t) = -\varepsilon (\frac{\partial E}{\partial w_{jk}} + \lambda w_{jk}) + \beta \cdot \Delta w_{jk}(t-1) \tag{B.20}$$

where the numbers in the brackets indicate the connection weights at a specific time step, and$0 < \beta < 1$ is the magnitude of the momentum.

B.4 Extension

It is most straightforward to extend the toy model in this appendix to models with more hidden layers. By continuing applying the chain rule, errors can be further back-propagated to drive the learning of other parameters.

Another way of extending this model is to view the layers p and q as the states of the same layer at two consecutive time steps, i.e. $p(t) = q(t-1)$. Thus, we can derive the learning rule for Recurrent Neural Networks (RNN) by treating w_{jk} as the recurrent connection determines the transformation between two consecutive time steps. This algorithm is also called Back-propagation through time (BPTT).

Appendix C

Conjugate Gradients

The conjugate gradient method (CG) is the optimizer at the heart of the powerful Hessian-Free optimization algorithm. By utilizing second-order information, it is guaranteed to reach the global minimum of a quadratic error function in n steps, where n is the input dimensionality of a function. For higher-order functions, such as the objective functions of most neural networks, the non-linear version the conjugate gradient method is significantly more efficient than algorithms utilizing only first-order information (basic gradient descent, such as Appendix B). Additionally, it is more economical in both computation and storage compared to quasi-Newton methods.

This self-contained appendix gives derivations of the conjugate gradient algorithm that are essential for understanding the Hessian-Free optimization algorithm. It can also be used as a brief introduction to the conjugate gradient method. For a more comprehensive version, see the excellent introduction by [79].

C.1 Linear conjugate gradient methods

Let us start from the most basic linear CG, where the function to be minimized, $f(x) : \mathbb{R}^N \to \mathbb{R}$, has the following quadratic form of equation A.1, with the gradient of equation A.2. From basic calculus, a convex function $f(x)$ reaches its minimum value $f(x^*)$ (AppendixA) when its gradient

$$g(x^*) = b + Ax^* = 0 \tag{C.1}$$

However, the direct evaluation of $x^* = -\frac{b}{A}$[1]is not plausible for large-scale problems, where the inversion of A is too expensive. As a result, we seek iterative methods to compute $x*$numerically. In the following derivation, we write $g(x_i)$ as g_i when appropriate.

The two central problems are

1. which direction we should search;
2. how large a step we should take along that direction.

[1] This is a special case of Newton's method.

C.1.1 Line search

First, we look at how to find an optimal step size α_i, at step i, given a search direction d_i. In other words, we try to find a value $\alpha_i$that minimizes $f(x_i + \alpha_i d_i)$. By taking the directional derivative,

$$\begin{aligned} \frac{\partial f(x_{i+1})}{\partial \alpha} &= 0 \\ \frac{\partial f(x_i + \alpha_i d_i)}{\partial \alpha_i} &= 0 \\ g_{i+1}^\top d_i &= 0 \end{aligned} \tag{C.2}$$

then substituting equation A.2 to calculate the gradient,

$$\begin{aligned} (b + Ax_{i+1})^\top d_i &= 0 \\ (b + A(x_i + \alpha_i d_i))^\top d_i &= 0 \end{aligned}$$

we obtain:

$$\alpha_i = -\frac{(b + Ax_i)^\top d_i}{d_i^\top A d_i} \tag{C.3}$$

This process is called line-search. If we use the opposite direction of the gradient at each step as d, the same direction as in gradient descent

$$d_i = -g_i \tag{C.4}$$

equation C.3 and equation C.4 give the steepest descent algorithm, which uses the curvature information (matrix A) to regularize step sizes.

Note that g_{i+1} and d_i being orthogonal means that after taking this step, we not only minimize $f(x)$ along the direction of d_i, but also reduce the length (strictly speaking, square of length) of the gradient at its largest extent. Therefore, any change along d_i will lengthen g_{i+1}, because they are already orthogonal. This can be proved by taking the derivative

$$\frac{\partial [g_{i+1}^\top g_{i+1}]}{\partial \alpha_i} = 0 \tag{C.5}$$

Equation C.5 has the same solution of equation C.3.

C.1.2 Conjugate directions

In order to find the direction we should take at each step, we need to analyze the optimization problem in greater depth. Note that the following statement are equal in the context of minimizing a quadratic function $f(x)$:

1. Minimizing $f(x)$.
2. Reaching a point where where the gradient of $f(x)$ is zero.
3. Reaching the minimal point x^*.

The first statement is at the highest level, but it may not give us enough insight into the optimization problem. For example, if we try to reduce the value of $f(x)$ directly at each iteration using gradient descent, the performance can be very bad for functions with low condition numbers. On the contrary, the third statement is at the lowest level. However, it is impractical since we don't know x^* in advance. Therefore, only statement 2 at the middle is left to guide us in selecting the directions to take.

Given a direction d_i and a step size α_i from equation C.3, we can obtain the gradient at x_{i+1}, g_{i+1}. Following the discussion in the last paragraph, our goal is to find the next step that continues reducing the gradient length towards zero, with the constraint that the gradient already reduced by d_i (from g_i) in the last step is not undone. From equation C.2, we infer this can be satisfied by keeping the gradient at the next step, g_{i+2}, still orthogonal to direction d_i, so that step i also minimizes g_{i+2}, in addition to g_{i+1}, along its direction d_i.

$$\begin{aligned} g_{i+2}^\top d_i &= 0 \\ g(x_{i+1}+\alpha_{i+1}d_{i+1})^\top d_i &= 0 \\ [b+A(x_{i+1}+\alpha_{i+1}d_{i+1})]^\top d_i &= 0 \\ (b+Ax_{i+1})^\top d_i + \alpha_{i+1} d_{i+1}^\top A d_i &= 0 \end{aligned} \tag{C.6}$$

From equation C.2, the first term of equation C.6 is zero. To guarantee the second term is also zero, directions d_{i+1} and d_i must satisfy

$$d_{i+1}^\top A d_i = 0 \tag{C.7}$$

Directions satisfying equation C.7 are said to be conjugate, and this is where the name "conjugate gradient" comes from.

To extend the previous argument using induction, since all the following steps also should not undo the "work" by step i(along its direction d_i) we further require that

$$g_k^\top d_i = 0 \tag{C.8}$$

holds true for all $k \geqslant 1$. Applying equation C.6 repeatedly, we see all the search directions must satisfy the conjugacy condition

$$d_i^\top A d_j = 0 \, for \, i \neq j \tag{C.9}$$

C.1.2.1 Search directions are linearly independent

We then prove that if A is positive definite, these search directions are linearly independent. Assuming these directions are not all linearly independent (*reductio ad absurdum*), there exists such d_k that

$$d_k = \sum_{i=0}^{n} \gamma_i d_i \tag{C.10}$$

where γ_i are constant coefficients and $\gamma_k = 0$. From equation C.9

$$d_k^\top A d_j = 0 \tag{C.11}$$

where $j \neq k$. Plugging-in equation C.10 into equation C.11 and simplifying i, we get

$$d_j^\top A d_j = 0 \tag{C.12}$$

which contradicts the positive definite assumption that gives $d_j^\top A d_j > 0$ for any d_j. Therefore, there exists N linearly independent search directions d for this N dimensional quadratic function $f(x)$.

C.1.2.2 Gradients are linearly independent

We can also prove that the gradients at different steps are linearly independent in a similar way. Assuming

$$g_k = \sum_{i=0}^{n} \mu_i g_i \tag{C.13}$$

multiply both sides by d_0:

$$\sum_{i=0}^{N} \mu_i g_i^\top d_0 = 0 \tag{C.14}$$

Using equation C.8, equation C.14 is simplified to

$$g_0 d_0 = 0 \tag{C.15}$$

which is false, since we never choose a search direction orthogonal to the gradient, as it may not change the function value at all. Therefore, the gradients are linearly independent.

Specifically, when the initial direction is chosen as $d_0 = -g_0$, we can use induction to prove that all the N gradients are orthogonal to each other. From equation C.8 we have

$$g_0^\top g_k = 0 \, for \, 1 \leqslant k \leqslant N \tag{C.16}$$

In addition, given that $g's$ and $d's$ both span the same N-1 dimension space, we can write

$$g_k = \sum_{l=1}^{N} \theta_l d_l \tag{C.17}$$

From equation A.2,

$$\begin{aligned} g_{i+1} - g_i &= A(x_{i+1} - x_i) \\ &= \alpha_i A d_i \end{aligned} \tag{C.18}$$

Now assuming

$$g_i^\top g_{i+k} = 0 \tag{C.19}$$

using equations C.18 and C.17:

$$\begin{aligned} g_{i+1}^\top g_{i+k} &= (g_i + \alpha_i A d_i)^\top g_{i+k} \\ &= g_i^\top g_{i+k} + \alpha_i d_i^\top A \Big(\sum_{l=i+1}^{N} \theta_l d_l \Big) \\ &= 0 \end{aligned} \tag{C.20}$$

Together with equation C.16, $g's$ are orthogonal to each other.

$$g_i^\top g_j = 0 \, for all \, i \neq j \tag{C.21}$$

Note that many other derivations regard the orthogonality between gradients as an important property of conjugate gradients, but actually this is only true when the initial direction is the opposite of its gradient ($d_0 = -g_0$). Nevertheless, as proved in section C.1.4, the initial search direction generally does not affect the convergence of the conjugate gradient method.

C.1.3 Constructing conjugate directions

Although in theory we can select the start direction d_0 and the starting point x_0 arbitrarily, the common practice is to start from $x_0 = 0$ and initialize $d_o = -g_0$. $x_0 = 0$ guarantees the non-constant part of the quadratic equation will be non-positive (actually 0) , providing a good starting point for minimization[2]. Taking the initial direction as the gradient ensures that the value of $f(x)$ will reduce quickly and gives the nice property that all gradients are orthogonal, as shown in section C.1.2.2. In addition, since the gradient will be evaluated in all the following steps (equation C.3), such initialization makes the whole process consistent.

After $d_0 = -g_0$ is obtained, the following directions can be constructed one by one using the Gram-Schmidt process

$$d_{i+1} = u_{i+1} - \sum_{k=0}^{i} \beta_{i+1,k} d_k \tag{C.22}$$

where $u's$ are linearly independent vectors. Equation C.22 means we can construct a series of conjugate directions[3] by subtracting from a vector u_{i+1} the components of i directions previously obtained. Given the independence of gradients (section C.1.2.2), we use the gradients obtained at step $i+1 (g_{i+1})$ as u_{i+1} in equation C.22. Multiplying both sides of equation C.22 by $d_i^\top A$, we can simplify it using conjugacy and orthogonality (equation C.8,C.9; here we drop the subscription k of β for simplicity)

$$0 = d_i^\top A g_{i+1} - \beta_{i+1} d_i^\top A d_i$$

$$\beta_{i+1} = \frac{g_{i+1}^\top A d_i}{d_i^\top A d_i} \tag{C.23}$$

Note that because of conjugacy between directions, β_1, β_2, ...β_i are not relevant for constructing consecutive conjugate directions. Therefore, equation C.22 can be written as

$$d_{i+1} = u_{i+1} - \beta_{i+1,k} d_k \tag{C.24}$$

[2]However, for Hessian-Free optimization, there are even better initial points (section E.3).

[3]Recall that the Gram-Schmidt process was first introduced to construct a set of orthogonal vectors. Conjugacy sometimes are also called A-orthogonality, where A is the matrix at the middle of equation C.9. Actually, the basic idea underlying the Gram-Schmidt process allow it to construct vectors with any such orthogonal relationships.

The matrix-vector product in equation C.23 can be avoided using equation C.18

$$\beta_{i+1} = \frac{g_{i+1}^\top (g_{i+1} - g_i)}{d_i^\top (g_{i+1} - g_i)} \tag{C.25}$$

which is known as the Hestenes-Stiefel expression. It can be simplified using equation C.8, C.24 to obtain the Polak-Ribiere form:

$$\beta_{i+1} = \frac{g_{i+1}^\top (g_{i+1} - g_i)}{g_j^\top g_i} \tag{C.26}$$

When we start with the opposite of the gradient as the initial direction, thus different gradients are orthogonal (equation C.21), the Polak-Ribiere form can be further simplified to the Fletcher-Reeves form

$$\beta_{i+1} = \frac{g_{i+1}^\top g_{i+1}}{g_i^\top g_i} \tag{C.27}$$

All three expressions computing the coefficients β are the same when conjugacy between direction is strictly held. This condition is generally true in the linear case, where only minor numerical errors may affect conjugacy. Therefore, we usually use the simplest one, equation C.27, in linear conjugate gradients.

C.1.4 Convergence of conjugate gradients

The conjugate gradient method is efficient because it is guaranteed to converge in n steps for n dimensional quadratic functions. Now we prove why this is true (equation C.3, C.23).

Suppose we have n conjugate directions $d_1, d_2, ...d_n$ constructed using equation C.23. Since they are linearly independent (section C.1.2.1), they span the n dimensional space where function $f(x)$ is defined. Therefore, we can find a set of steps $\alpha_i d_i$ leading to the solution x^* from an initial point x_0

$$x^* - x_0 = \sum_{i=1}^{n} \alpha_i d_i \tag{C.28}$$

Multiply both sides by $d_0^\top A$ and use equation C.1 to get rid of x^*

$$-d_0^\top (b + Ax_0) = \sum_{i=1}^{n} \alpha_i d_0^\top A d_i \tag{C.29}$$

Again, using equation C.9, we simplify the above equation and solve for

$$\alpha_0 = -\frac{(b + Ax_0)^\top d_0}{d_0^\top A d_0} \tag{C.30}$$

Similarly, we can solve all the coefficients

$$\alpha_i = -\frac{(b + Ax_i)^\top d_i}{d_i^\top A d_i} \tag{C.31}$$

which is the same as equation C.3. Therefore, we proved the conjugate gradient method converges to the minimal point x^* in N steps for any N dimensional quadratic function.

C.1.5 The conjugate gradient algorithm

Since the derivation of conjugate gradients is quite complicated, here we summarize it in the algorithm form. It may look slightly different from the derivation above because of some pre-multiplication in order to improve the performance.

1. Initialize the starting position $x \leftarrow x_0$
2. $g \leftarrow b + Ax$
3. $d \leftarrow -g$
4. $\delta_i \leftarrow g^\top g$
5. $p \leftarrow Ad$
6. $\alpha \leftarrow \frac{\delta_i}{d^\top p}$
7. $x \leftarrow x + \alpha d$
8. $g \leftarrow g + \alpha p$
9. $\delta_{i-1} \leftarrow \delta_i$
10. $\delta_i \leftarrow g^\top g$
11. $\beta \leftarrow \frac{\delta_i}{\delta_{i-1}}$
12. $d \leftarrow r - \beta d$
13. goto step 5 until meet stop condition

The stop condition in the last step can be reaching a certain precision or the maximum iterations. Another good choice of the termination condition is the on described in [57], which terminates the CG when the decrease of the objective function is small. It terminates CG at iteration i when the following inequities satisfy:

$$\begin{cases} i > k \\ \phi(x_i) < 0 \\ \frac{\phi(x_i) - \phi(x_{i-k})}{\phi(x_i)} < k\varepsilon \end{cases}$$

Where $k = max(10, 0.1 \times i)$ and $\varepsilon = 1e-4$ is chosen in the experiments in this thesis. The ε is smaller than that in [57], because the regression here requires higher precision. Although increasing ε may accelerate learning, a too large ε could produce low quality steps that slow down the learning and even make the Hessian-Free optimization fail to converge.

C.2 Non-linear Conjugate gradient methods

Most practical problems, such as optimization in machine learning, are non-linear. We can apply the linear conjugate gradient method described before with only slight modification to the line-search and constructing conjugate direction steps.

For line-search, since the error surface is not quadratic, the first step one may tempt to do is to substitute matrix A with the second-order derivative f"(x), or the Hessian matrix, the same as in the second-order expansion of $f(x+\delta)$ at x

$$q(\delta) = f(x) + \nabla f(x)^\top \delta + \delta^\top H \delta \tag{C.32}$$

where δ is the increment at a step. In other words, if x represents parameters at the last step, δ represents the increment of these parameters from the last step. However, equation C.3 is no longer able to minimize the error in one step. A simple solution is to apply equation C.3 repeatedly until the the change of error function after one step is smaller than a threshold value. This is actually the Newton-Raphson method. When the second-order derivative of $f(x)$ is not available or too expensive to compute, one may approximate it using secant method or other line-search algorithms that do not depend on second order information (see [79] for more information).

As discussed in section C.1.3, all three expressions of the coefficients β (equation C.25, C.26 and C.27) are the same when conjugacy between directions is strictly held, which is generally true in linear cases. However, the conjugacy deteriorates fast in the non-linear case, in which matrix A, thus the definition of conjugacy, is constantly changing. In non-linear case, the Polak-Ribiere (equation C.26) form is found to perform slightly better than the Fletcher-Reeves form. One reason might be that the assumption of orthogonality between gradients, in order to obtain equation C.27, is usually not true, as discussed in section C.1.2.2.

When applying the modified steps above, since conjugacy between search directions is no longer preserve, CG generally cannot converge in N steps. For this reason, a common practice is to restart the conjugate gradient method after N steps. Despite the break of conjugacy in non-linear problems, CG still has relatively good performance.

However, the non-linear conjugate gradient method, as well as many other second-order algorithms, has the problem that it may be trapped in local minima or even local maxima. These unfavored behaviors can be explained using the conclusion from appendix A: if the instantaneous Hessian at a step is negative-definite, the line-search will move towards local maxima, increasing the error; if the Hessian is indefinite, the line-search may either increase or reduce the error. As a result, once the Hessian is not positive-definite, which usually happens for highly non-linear functions, the behavior of conjugate gradients (and many other second-order optimization algorithms) may not be as good as one may expect. Furthermore, even if the Hessian is always positive-definite, when the line-search is exact, the error function will go to local minima. On the other hand, if the line-search is inexact, the conjugacy will deteriorate even faster.

Appendix D

An efficient way of computing matrix-vector products

In the conjugate gradient method introduced in appendix C, many steps require computing the product between a matrix and a vector (e.g., step 2 and 5 in the algorithm in C.1.5). Since the matrix A in has the size of $n \times n$ for any n dimensional function $f(x)$, simply accessing matrix A has the complexity $O(n^2)$. In large-scale problems, such as training neural networks with a reasonable size, the size of A will be too large for both storing A and computing the matrix-vector product directly. This appendix introduces the $\Re\{\cdot\}$ operator[67] that efficiently computes matrix-vector products with linear complexity ($O(n)$).

We start our derivation with the first-order expansion of $f(x+\Delta x)$ at x

$$f(x+\Delta x) = f(x) + f'(x)^\top \Delta x + O(||\Delta x||^2) \tag{D.1}$$

Because of the error term $O(||\Delta x||^2)$, the expansion above is exact. The expansion can be rewritten by substituting Δx with the product of a scalar r and a vector v

$$f(x+rv) = f(x) + f'(x)^\top rv + O(||rv||^2) \tag{D.2}$$

Rearranging equation D.2 and dividing both sides by the scalar r, we obtain the product between $f'(x)$, a matrix, and the vector v[1]:

$$\begin{aligned} f'(x)^\top v &= \frac{f(x+rv) - f(x)}{r} + O(|r|) \\ &= \lim_{r \to 0} \frac{f(x+rv) - f(x)}{r} \end{aligned} \tag{D.3}$$

where we further drop the error term and obtain the exact product by taking the limit $r \to 0$. Equation D.3 is actually the definition of the directional derivative of $f(x)$ along the direction v. Therefore, we can define the $\Re\{\cdot\}$ operator[2]

$$\Re_v\{f(x)\} \equiv \frac{\partial}{\partial r} f(x+rv) \mid_{r=0} \tag{D.4}$$

[1] In actual application in optimization, the matrix in the place of $f'(x)$ is usually a symmetric matrix, such as a Hessian or Gauss-Newton matrix.

[2] The subscript v in equation D.4 is usually omitted for simplicity when no confusion will occur.

Since $\Re\{\cdot\}$ is a differential operator, rules of differential operators, such as chain rule, product rule, quotient rule also apply to it:

$$\Re\{c \cdot f(x)\} = c\Re\{f(x)\}$$

$$\Re\{f(x) + g(x)\} = \Re\{f(x)\} + \Re\{g(x)\}$$

$$\Re\{f(x)g(x)\} = \Re\{f(x)\}g(x) + f(x)\Re(g(x))$$

$$R\{f(g(x))\} = f'((g(x))\Re\{g(x)\}$$

$$\Re\{\frac{df(x)}{dt}\} = \frac{d\Re\{f(x)\}}{dt}$$

Specifically,

$$\Re\{x\} = v$$

can be proved by the definition in equation D.3, D.4

$$\begin{aligned} \Re\{x\} &= \lim_{r \to 0} \frac{(x + rv) - x}{r} \\ &= v \end{aligned}$$

It is exact the product

$$(x)'v = Iv = v$$

A straightforward application of the$\Re\{\cdot\}$ is to compute the product between Hessian and another vector as a part of the nonlinear conjugate gradient method.

Appendix E

Hessian-Free Optimization

E.1 Overview

With the background knowledge introduced in Appendices A, C and D, we can derive the Hessian-Free optimization algorithm (HF). As discussed in section C.2, directly applying the non-linear conjugate gradient method or other second-order algorithms may result in unfavored behaviors caused by the Hessian matrices that are not positive semi-definite. To solve this problem, the general idea behind the Hessian-Free Optimization is to regulate the local curvature (second-order) information, providing an ideal environment for the conjugate gradient method[1].

Following the discussion above, we functionally divide the Hessian-Free optimization into two steps: local quadratic approximation and optimization. In the first step, a positive-definite quadratic function is used to approximate the original non-linear objective function locally. In the second step, we optimize the quadratic function obtained in the first step using the linear conjugate gradient method. Since only the quadratic function is used for optimization, the convergence properties of the linear CG are well preserved.

E.2 Positive semi-definite approximation

Recall that the problem of many second-order methods lies with the Hessian – it may not be positive semi-definite. To solve this problem, we need to find a matrix guaranteed to be positive semi-definite as an approximation of the Hessian, and use this matrix in place of the Hessian to approximate the original objective function. We start to find this positive semi-definite matrix by analyzing the Hessian[2].

[1]The conjugate gradient method, instead of other second order optimizer, is used here, because it is efficient in both storage and computation (both are linear) for large scale problems. A detailed comparison of second-order methods is beyond the scope of this appendix. Please refer to text such as Chapter 7 of [10] for more detailed account of different optimization algorithms.

[2]In this appendix, we write the objective function as $f(\theta)$ instead of $f(x)$ to more clear indicate our purpose of optimizing for parameters of a neural network.

Following the notation from [77], we re-write the objective function as the composition of function L, M and N

$$f(\theta) = L(M(N(\theta))) = L \circ M \circ N \tag{E.1}$$

For a neural network with n input and m output units, N is the function mapping the parameter set θ to the m outputs(input is given in the training set). $M : \mathbb{R}^m \to \mathbb{R}^m$ is the function of the output unit. For linear output units which are usually used in regression, M is the identity function (i.e. multiplication by an identity matrix), whereas for classification problems, M can be the logistic function (two class, binomial distribution) or the softmax function (multiple classes, multinomial distribution). In addition, $L : \mathbb{R}^m \to \mathbb{R}^1$ is the loss function measuring the discrepancy between model outputs and target outputs. The loss function can be the squared error function or cross entropy error.

The choice of loss function depends on the output function M. If M is a linear output function, the squared error function will be its matching loss function. On the other hand, for logistic or softmax outputs, the cross entropy error will be the matching loss function. Matching loss functions are also called canonical response functions. Statistically, the matching loss function is consistent with the maximal likelihood estimation. One way to derive these matching loss functions is to first assume the distribution of model output (Gaussian distribution for linear output units, binomial distribution for logistic output units and multinomial distribution for softmax output units), and then simplify the maximal likelihood function (see [10] for more detail).

With the notation of equation E.1, we can further rewrite the Hessian:

$$H = \frac{\partial}{\partial \theta}(J_{L \circ M} J_N) = J_N^\top H_{L \circ M} J_N + \sum_{i=1}^{m} (J_{L \circ M})_i H_{N_i} \tag{E.2}$$

where J's are Jacobians, the first derivative of a function, and the second term of the above equation denotes the sum across all m output units of the neural network. Another good property of matching loss function is that if L and M are matching, the Jacobian $J_{L \circ M}$ can be simplified to the difference between the model output z and the target outputs z^*,

$$J_{L \circ M} = z - z^* \tag{E.3}$$

Therefore, the second term of equation E.2 diminishes when θ approaches the true solution θ^*[3]. For this reason, we can discard the second term in equation E.2 without affecting the purpose of optimizing, since this term approaches zero near the solution. This gives the Gauss-Newton matrix

$$G \equiv J_N^\top H_{L \circ M} J_N \tag{E.4}$$

which we will use in place of the Hessian in equation C.32.

Still, all approximation-based algorithms have the problem of model-trust region. When the conjugate gradient method optimize the approximated quadratic equation, if

[3]We call it "solution" in the context of solving the linear equation A.2 that gives the minimal value of the quadratic function to be optimized. Therefore, θ^* is also the optimal parameter set.

the step taken is too large (large δin equation C.32), the approximation may be bad. As a result, decrease in the (approximated) quadratic function may not be accompanied by a similar decrease in the objective function. We thus need to regularize the step size, usually by adding a Tikhonov term $\lambda\delta^\top\delta$ that penalizes large δ[4]. After using the Gauss-Newton matrix, and adding the Tikhonov term, we obtain the objective function

$$q(\delta) = f(\theta) + \nabla f(\theta)^\top \delta + \delta^\top (G + \lambda I)\delta \tag{E.5}$$

where I is the identity matrix.

Worth noting that equation E.5 is the same as the objective function in the Levenberg-Marquardt algorithm when the neural network has linear output units, where $G = J_N^\top J_N$. Therefore, we can understand the objective function in the Hessian-Free algorithm as an extension of the Levenberg-Marquardt algorithm to non-linear outputs[5]. The parameter λ in equation E.5 is also adjusted the same way as in the Levenberg-Marquardt algorithm:

$$\begin{cases} \lambda = c\lambda & \phi < \frac{1}{4} \\ \lambda = \frac{1}{c}\lambda & \phi > \frac{3}{4} \end{cases} \tag{E.6}$$

where c is a positive value (c is usually set to 10 in the classical Levenberg-Marquardt algorithm, and set to 4 in Hessian-Free), and the positive number

$$\phi = \frac{\text{the decrease of the quadratic function}}{\text{the decrease of the original function}}$$

The intuition behind this heuristic of adjusting theλvalue is that if the decrease in the objective function is small compared to the decrease in the quadratic function, the approximation is bad, so we need to increase λ to impose a more conservative step. On the contrary, if ϕ is already large enough, thus the approximation is accurate, larger steps can accelerate the optimization.

In [58], another method called the structural damping is used in conjunction with the Tikhonov term to further regularize the objective function for recurrent neural networks (RNN) by specifically penalizing the distortion compared with the non-linearity in the hidden layer of a RNN. However, We do not find it useful in the tasks in this thesis. Given that the structural damping is not a general approach, it is not presented here.

E.3 Optimization on quadratic approximation

Now we can put everything together, optimizing an objective function $f(\theta_n)$ through optimizing its quadratic approximation $q_n(\delta)$ at each step using the conjugate gradient methods. We stop the conjugate gradients once the value of $q_n(\delta)$ decreased to below a tolerance value, thus obtaining a step δ_n. Next, we update both the approximated quadratic function and parameters and optimize the new quadratic function q_{n+1} from δ_n, where q_{n+1} is the approximation of the objective at $\theta_{n+1} = \theta_n + \delta_n$. Note that the

[4]It is also called L2 regularization, which is always written as $\frac{1}{2}\lambda||\delta||^2$

[5]However, the Levenberg-Marquardt algorithm does not use conjugate gradients as its optimizer.

optimization on q_{n+1} starts from the direction of the last step δ_n, instead of 0 (section C.1.3), since this initial direction usually significantly accelerates the conjugate gradient method in the Hessian-Free optimization [57]. The reason might be that q_{n+1} and q_n are similar, so they are likely to have resembling minimal points.

In some situations, especially when the step-size becomes very small, the Tikhonov regularization is not enough to deal with the very high local non-linearity. In these cases, we shorten δ_n[6] through an inexact backtracking line-search to obtain a δ_n^* that better accommodates the local surface. The Wolfe conditions can be used to govern this inexact line search by finding a step length α_k, so that $\delta_n^* = \alpha_k \delta_n$ gives a sufficiently smaller value of the objective function than δ_n. A step length α_k is said to satisfy the Wolfe conditions if

1.
$$f(\theta_n + \alpha_k \delta_n) \leqslant f(\theta_n) + c_1 \alpha_k \delta_n^\top \nabla f(\theta_n) \tag{E.7}$$

2.
$$\delta_n^\top \nabla f(\theta_n + \alpha_k \delta_n) \geqslant c_2 \delta_n^\top \nabla f(\theta_n) \tag{E.8}$$

After the initialization $\alpha_1 = 1$, if any of the Wolfe conditions is violated, we update the step length by backtracking $\alpha_{n+1} = \tau \alpha_n$, where τ is a value smaller than 1 (0.7 in the experiments in this thesis). Since the Wolfe condition is hard to satisfy, we usually limit the times of backtracking performed (a limit of $k_{max} = 5$ is used here so that δ_n^* can not be smaller than $\tau^5 \delta_n$).

E.4 Modified Passes

Backpropagation is a way of applying chain rule to calculate derivatives of a function. In neural networks, the original backpropagation computes the gradient of an objective function[74] (see also appendix B). Using the R-operator introduced in appendix D, we can modify the original backward and forward passes to produce useful measures such as Hessian, Fisher Information matrix or Gauss-Newton matrix[77].

As an example, this section gives detailed derivations of using forward and back passes to calculate the product of a Gauss-Newton matrix and a vector v in a recurrent neural network with linear output units and squared error objective function. We assume the hidden units using the tanh function. Equations for other structures (not necessarily neural networks) can be derived in similar ways.

In the following equations, "$\cdot$" denotes matrix multiplication, and "$*$" denotes element wise multiplication.

[6] An additional "CG back-tracking" process is introduced in [57], in which the parameter is updated by δ_n' that is obtained from go back a few steps of the conjugate gradient methods until if the back-tracking process yields a smaller value of the original objective function. However, I found this process slows down the overall convergence, and some time even make a few problems failed to converge, so the CG back-tracking is not presented. The drawback of CG back-tracking might be a result of going back to local minima that ignored by the smoothy quadratic approximation.

E.4.1 Standard forward pass

This is the standard forward pass a typical recurrent neural network.

Initialize $h_0 = 0$, run from time step 1 to the last step T:

$$y_t = W_{hi} \cdot x_t + W_{hh} \cdot h_{t-1} + b_h \tag{E.9}$$

$$h_t = \tanh(y_t) \tag{E.10}$$

$$z_t = W_{oh} \cdot h_t + b_o \tag{E.11}$$

$$f = \sum_t (z_t - z_t^*)^\top \cdot (z_t - z_t^*) \tag{E.12}$$

E.4.2 Standard backward pass

The backward pass is obtained from take the derivative of the objective function using the chain rule. Note it uses intermediate results from the standard forward pass.

Initialize $\frac{\partial f}{\partial y_{t+1}} = 0$, run from time step T backward to step 1:

$$\frac{\partial f}{\partial z_t} = z_t - z_t^* \tag{E.13}$$

$$\frac{\partial f}{\partial h_t} = W_{hh} \cdot \frac{\partial f}{\partial y_{t+1}} + W_{oh} \cdot \frac{\partial f}{\partial z_t} \tag{E.14}$$

$$\frac{\partial f}{\partial y_t} = (1 + h_i) * (1 - h_i) * \frac{\partial f}{\partial h_t} \tag{E.15}$$

Only this line is responsible for structural damping:

$$\frac{\partial f}{\partial y_t} = \frac{\partial f}{\partial y_t} + \lambda \mu (1 + h_t) * (1 - h_t) * y_t \tag{E.16}$$

$$\frac{\partial f}{\partial W_{oh}} = \frac{\partial f}{\partial z_t} \cdot h_i^\top \tag{E.17}$$

$$\frac{\partial f}{\partial b_o} = \frac{\partial f}{\partial z_t} \tag{E.18}$$

$$\frac{\partial f}{\partial W_{hh}} = \frac{\partial f}{\partial y_{t+1}} \cdot h_i^\top \tag{E.19}$$

$$\frac{\partial f}{\partial b_h} = \frac{\partial f}{\partial y_t} \tag{E.20}$$

$$\frac{\partial f}{\partial W_{hi}} = \frac{\partial f}{\partial y_i} \cdot x_i^\top \tag{E.21}$$

E.4.3 R-forward pass

The R-forward is derived from applying the R operator around equations for the standard forward pass. It uses intermediate results from both standard forward and backward passes.

Initialize$R(h_0) = 0$

$$R(y_t) = v^{W_{hi}} \cdot x_t + v^{W_{hh}} \cdot h_{t-1} + W_{hh} \cdot R(h_{t-1}) + v^{b_h} \tag{E.22}$$

$$R(h_t) = (1 + h_t) * (1 - h_t) * R(y_t) \tag{E.23}$$

$$R(z_t) = v^{W_{oh}} \cdot h_t + W_{oh} \cdot R(h_y) + v^{b_o} \tag{E.24}$$

E.4.4 R-backward pass

To construct the product between the Gauss-Newton matrix and vector v, we only R-backpropagating through the loss function $\mathscr{L}$, and then backpropagate in the standard way. The only step different from standard backward pass is that we initialize $R(\frac{\partial f}{\partial z_t}) = R(z_t)$, rather than E.13. In calculating the Gauss-Newton matrix, this step can be understood as back-propagating the product $J_N v$, as defined by $\Re\{Z_t\}$ (equation D.4), instead of the error as in the standard backward pass. It uses the intermediate results from all above passes.

E.4.5 Mini-batches

As described in section C.1.5, both the gradient and the matrix-vector product are calculated for conjugate gradients. However, the matrix-vector products need to be evaluated much more often than the gradients, since for each HF step (optimization on a given quadratic equation) the gradient only need to be evaluated once while as the matrix-vector products need to be evaluated at each CG line-search step. Moreover, it's clear from the above passes that the evaluation of matrix-vector product is more expensive than evaluating the gradients. Therefore, it is reasonable too consider using smaller batches, as subsets of the full training batch, in evaluating the matrix-vector products, when the training batch is large.

E.5 The algorithm

To summarize this appendix, the steps in the Hessian-Free optimization are outlined here:

1. Randomly initialize parameters of a modelθ; initialize the first step $\delta = 0$.
2. Run CG on the quadratic function q approximated at $f(\theta)$, noting (see section C.1.5 for processes of CG):

(a) Start CG from $q(\delta)$

(b) Evaluate the gradient g using standard backpropagation.

(c) Evaluate the matrix-vector product (such as $G \cdot v$) used in CG by modified passes.

(d) Terminate CG when stop condition of CG meets, obtaining a step δ.

3. Use inexact backtracking line-search guided by the Wolfe conditions to obtain δ^*.

4. Update the parameters by $\theta = \theta + \delta^*$.

5. Go to step 2 until termination conditions of HF meets.

The termination conditions of HF, which terminate the whole learning process, is somewhat task dependent. It usually includes reaching maximum epochs or exceeding a tolerance of error.

References

[1] James B. Aimone, Janet Wiles, and Fred H. Gage. Computational influence of adult neurogenesis on memory encoding. *Neuron*, 61(2):187–202, January 2009. 3, 7, 43

[2] Joseph Altman. Are new neurons formed in the brains of adult mammals? *Science*, 135(3509):1127–1128, March 1962. 43

[3] Pablo Alvarez and Larry R. Squire. Memory consolidation and the medial temporal lobe: a simple network model. *Proceedings of the National Academy of Sciences of the United States of America*, 91(15):7041–5, July 1994. 3

[4] Per Anderson, Richard Morris, David Amaral, Tim Bliss, and John O'Keefe. *The hippocampus book*. OXford University Press, 2007. 3, 4

[5] David A. August and William B. Levy. Temporal sequence compression by an integrate-and-fire model of hippocampal area CA3. *Journal of computational neuroscience*, 6(1):71–90, January 1999. 45, 46

[6] Arnold Bakker, C. Brock Kirwan, Michael Miller, and Craig E. L. Stark. Pattern separation in the human hippocampal CA3 and dentate gyrus. *Science (New York, N.Y.)*, 319(5870):1640–2, March 2008. 3

[7] Suzanna Becker. A computational principle for hippocampal learning and neurogenesis. *Hippocampus*, 15(6):722–38, January 2005. 7, 43

[8] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 5(2):157–66, January 1994. 29

[9] Chris M. Bird and Neil Burgess. The hippocampus and memory: insights from spatial processing. *Nature reviews. Neuroscience*, 9(3):182–94, March 2008. 3, 7

[10] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. OXford University Press, 1996. 30, 34, 66, 67

[11] Neil Burgess, Caswell Barry, and John O'Keefe. An oscillatory interference model of grid cell firing. *Hippocampus*, 17(9):801–812, 2007. 18

[12] Neil Burgess, Suzanna Becker, John A. King, and John O'Keefe. Memory for events and their spatial context: models and experiments. *Philosophical transactions of the Royal Society of London*, 356(1413):1493–503, September 2001. 3, 6

[13] Margaret F. Carr, Shantanu P. Jadhav, and Loren M. Frank. Hippocampal replay in the awake state: a potential substrate for memory consolidation and retrieval. *Nature neuroscience*, 14(2):147–53, February 2011. 2

[14] Neal J. Cohen and Howard Eichenbaum. *Memory, Amnesia, and the Hippocampal System*. The MIT Press, 1993. 1, 3, 6

[15] Laura Lee Colgin, Edvard I. Moser, and May-Britt Moser. Understanding memory through hippocampal remapping. *Trends in neurosciences*, 31(9):469–77, September 2008. 7, 8

[16] John Conklin and Chris Eliasmith. A controlled attractor network model of path integration in the rat. *Journal of computational neuroscience*, 18(2):183–203, 2005. 17, 18, 43

[17] Thomas J. Davidson, Fabian Kloosterman, and Matthew A. Wilson. Hippocampal replay of extended experience. *Neuron*, 63(4):497–507, 2009. 2, 9, 46

[18] Nathaniel D. Daw, Yael Niv, and Peter Dayan. Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature neuroscience*, 8(12):1704–11, December 2005. 44

[19] Peter Dayan. Goal-directed control and its antipodes. *Neural networks : the official journal of the International Neural Network Society*, 22(3):213–9, April 2009. 44

[20] Sachin S. Deshmukh and Upinder S. Bhalla. Representation of odor habituation and timing in the hippocampus. *The Journal of neuroscience : the official journal of the Society for Neuroscience*, 23(5):1903–15, March 2003. 1

[21] Kamran Diba and György Buzsáki. Forward and reverse hippocampal place-cell sequences during ripples. *Nature neuroscience*, 10(10):1241–2, October 2007. 34, 36

[22] Howard Eichenbaum, Paul Dudchenko, Emma Wood, Matthew Shapiro, and Heikki Tanila. The hippocampus, memory, and place cells: is it spatial memory or a memory space? *Neuron*, 23(2):209–26, June 1999. 1, 6, 12, 46

[23] Arne D. Ekstrom, Micheal J. Kahana, Jeremy B. Caplan, Tony A. Fields, Eve A. Isham, Ehren L. Newman, and Itzhak Fried. Cellular networks underlying human spatial navigation. *Nature*, 425(September), 2003. 1

[24] Chris Eliasmith. A unified approach to building and controlling spiking attractor networks. *Neural computation*, 17(6):1276–314, June 2005. 23, 28, 43, 46

[25] Chris Eliasmith and Charles H. Anderson. *Neural Engineering*. The MIT Press, 2004. 12, 15

[26] Jeffrey Elman. Finding Structure in Time. *Cognitive Science*, 211:179–211, 1990. 20, 29

[27] André A. Fenton, Hsin-Yi Kao, Samuel a Neymotin, Andrey Olypher, Yevgeniy Vayntrub, William W Lytton, and Nandor Ludvig. Unmasking the CA1 ensemble place code by exposures to small and large environments: more place cells and multiple, irregularly arranged, and expanded place fields in the larger space. *The Journal of neuroscience : the official journal of the Society for Neuroscience*, 28(44):11250–62, October 2008. 4

[28] Ila R. Fiete, Yoram Burak, and Ted Brookings. What grid cells convey about rat location. *The Journal of neuroscience : the official journal of the Society for Neuroscience*, 28(27):6858–71, July 2008. 20

[29] David J. Foster and Matthew A. Wilson. Reverse replay of behavioural sequences in hippocampal place cells during the awake state. *Nature*, 440(7084):680–3, March 2006. 2, 9, 11, 34, 36

[30] Marianne Fyhn, Sturla Molden, Menno P. Witter, Edvard I. Moser, and May-Britt Moser. Spatial representation in the entorhinal cortex. *Science (New York, N.Y.)*, 305(5688):1258–64, August 2004. 20

[31] Lisa M. Giocomo, Eric A. Zilli, Erik Fransén, and Michael E. Hasselmo. Temporal frequency of subthreshold oscillations scales with entorhinal grid cell field spacing. *Science (New York, N.Y.)*, 315(5819):1719–22, March 2007. 18, 20

[32] April E. Gold and Raymond P. Kesner. The role of the CA3 subregion of the dorsal hippocampus in spatial pattern completion in the rat. *Hippocampus*, 15(6):808–14, January 2005. 43

[33] Anoopum S. Gupta, Matthijs A. A. van der Meer, David S. Touretzky, and A. David Redish. Hippocampal replay is not a simple function of experience. *Neuron*, 65(5):695–705, March 2010. 6, 9, 46

[34] Torkel Hafting, Marianne Fyhn, Sturla Molden, May-Britt Moser, and Edvard I. Moser. Microstructure of a spatial map in the entorhinal cortex. *Nature*, 436(7052):801–6, August 2005. 20

[35] Demis Hassabis, Carlton Chu, Geraint Rees, Nikolaus Weiskopf, Peter D. Molyneux, and Eleanor A. Maguire. Decoding neuronal ensembles in the human hippocampus. *Current biology : CB*, 19(7):546–54, April 2009. 6

[36] Demis Hassabis, Dharshan Kumaran, Seralynne D. Vann, and Eleanor A. Maguire. Patients with hippocampal amnesia cannot imagine new experiences. *Proceedings of the National Academy of Sciences of the United States of America*, 104(5):1726–31, January 2007. 6

[37] Michael E. Hasselmo. *How we remember*. MIT Press, 2011. 3, 38, 44, 46

[38] Micheal E. Hasselmo, Lisa M. Giocomo, and Eric A. Zilli. Grid cell firing may arise from interference of theta frequency membrane potential oscillations in single neurons. *Hippocampus*, 17(12):1252–1271, 2007. 20

[39] Geoffrey E. Hinton. Learning distributed representations of concepts. In R. G. M. Morris, editor, *Parallel distributed processing: Implications for psychology and neurobiology*, pages 46–61. Clarendon Press, Oxford, England, 1989. 43

[40] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006. 29, 42

[41] Sepp Hochreiter and Paolo Frasconi. Gradient Flow in Recurrent Nets : the Di culty of Learning Long-Term Dependencies 1 Introduction 2 Exponential error decay Gradients of the error function. In John F. Kolen and Stefan C. Kremer, editors, *A Field Guide to Dynamical Recurrent Networks*. Wiley-IEEE Press, 2001. 29

[42] Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–80, November 1997. 29, 31

[43] Etsuro Hori, Yoichi Nishio, Kenichi Kazui, Katsumi Umeno, Eiichi Tabuchi, Kazuo Sasaki, Shunro Endo, Taketoshi Ono, and Hisao Nishijo. Place-related neural responses in the monkey hippocampal formation in a virtual space. *Hippocampus*, 15(8):991–6, January 2005. 2

[44] Daoyun Ji and Matthew A. Wilson. Coordinated memory replay in the visual cortex and hippocampus during sleep. *Nature Neuroscience*, 10(1):100–107, 2006. 44

[45] Adam Johnson and a David Redish. Hippocampal replay contributes to within session learning in a temporal difference reinforcement learning model. *Neural networks : the official journal of the International Neural Network Society*, 18(9):1163–71, November 2005. 2

[46] Adam Johnson and A. David Redish. Neural ensembles in CA3 transiently encode paths forward of the animal at a decision point. *The Journal of neuroscience*, 27(45):12176–89, November 2007. 8, 32

[47] Szabolcs Káli and Peter Dayan. Off-line replay maintains declarative memories in a model of hippocampal-neocortical interactions. *Nature neuroscience*, 7(3):286–94, March 2004. 44

[48] Mate Lengyel and Peter Dayan. Hippocampal contributions to control: The third way. *Advances in neural information processing systems*, 20:889–896, 2007. 44

[49] Mate Lengyel and Peter Dayan. Performance analysis of alternative controllers Tree-structured MDPs. In *Advances in neural information processing systems*, pages 1–22, 2009. 44

[50] Jill K Leutgeb, Stefan Leutgeb, May-Britt Moser, and Edvard I. Moser. Pattern separation in the dentate gyrus and CA3 of the hippocampus. *Science*, 315(5814):961–6, February 2007. 43

[51] William B. Levy, Ashlie B. Hocking, and Xiangbao Wu. Interpreting hippocampal function as recoding and forecasting. *Neural networks : the official journal of the International Neural Network Society*, 18(9):1242–64, November 2005. 44

[52] William B. Levy and Oswald Steward. Synapses as associative memory elements in the hippocampal formation. *Brain research*, 175:233–245, 1979. 44

[53] Christopher J. MacDonald, Kyle Q. Lepage, Uri T. Eden, and Howard Eichenbaum. Hippocampal "Time Cells" Bridge the Gap in Memory for Discontiguous Events. *Neuron*, 71(4):737–749, August 2011. 1, 4, 6, 7

[54] Ludise Malkova and Mortimer Mishkin. One-trial memory for object-place associations after separate lesions of hippocampus and posterior parahippocampal region in the monkey. *The Journal of neuroscience : the official journal of the Society for Neuroscience*, 23(5):1956–65, March 2003. 1, 6

[55] David Marr. From the Retina to the Neocortex. *Citeseer*, 1991. 42

[56] James Martens. *A New Algorithm for Rapid Parameter Learning in Linear Dynamical Systems*. PhD thesis, 2009. 30

[57] James Martens. Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, volume 951, page 2010, 2010. 46, 62, 69

[58] James Martens and Ilya Sutskever. Learning Recurrent Neural Networks with Hessian-Free Optimization. In *Proceedings of the 28th International Conference on Machine Learning*, 2011. 46, 68

[59] James L. McClelland, Bruce L. McNaughton, and Randall C. O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419–57, July 1995. 42, 44

[60] Matthew D. McEchron, Wilbur Tseng, and John F. Disterhoft. Neurotoxic lesions of the dorsal hippocampus disrupt auditory-cued trace heart rate (fear) conditioning in rabbits. *Hippocampus*, 10(6):739–51, January 2000. 1

[61] Edvard I. Moser, Emilio Kropff, and May-Britt Moser. Place cells, grid cells, and the brain's spatial representation system. *Annual review of neuroscience*, 31:69–89, January 2008. 43

[62] John O'Keefe and Jonathan Dostrovsky. The hippocampus as a spatial map: Preliminary evidence from unit activity in the freely-moving rat. *Brain Research*, 34:171–175, 1971. 1, 4

[63] John O'Keefe and Lynn Nadel. *The Hippocampus as a Cognitive Map*. OXford University Press, 1978. 4, 43

[64] John O'Keefe and Lynn Nadel. Précis of O'Keefe & Nadel's The hippocampus as a cognitive map. *Behavioral and Brain Sciences*, 2:487–494, May 1979. 1

[65] Randall C. O'Reilly and Jerry W. Rudy. Computational principles of learning in the neocortex and hippocampus. *Hippocampus*, 10(4):389–97, January 2000. 42, 43

[66] Eva Pastalkova, Vladimir Itskov, Asohan Amarasingham, and György Buzsáki. Internally generated cell assembly sequences in the rat hippocampus. *Science (New York, N.Y.)*, 321(5894):1322–7, September 2008. 1, 4, 20

[67] Barak A. Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 6(1):147–160, 1994. 64

[68] Fernando J. Pineda. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, 59(19):2229–2232, 1987. 29

[69] L. R. Rabiner and B. H. Juang. An introduction to hidden Markov models. *ASSP Magazine, IEEE*, 3(1):4–6, June 1986. 29

[70] A. David Redish. *Beyond the Cognitive Map*. A Bradford Book, 1999. 1

[71] A. David Redish and David S. Touretzky. Cognitive maps beyond the hippocampus. *Hippocampus*, 7(1):15–35, January 1997. 28, 43

[72] Sébastien Royer, Anton Sirota, Jagdish Patel, and György Buzsáki. Distinct representations and theta dynamics in dorsal and ventral hippocampus. *The Journal of neuroscience*, 30(5):1777–87, February 2010. 23

[73] Jerry W. Rudy and Randall C. O'Reilly. Contextual fear conditioning, conjunctive representations, pattern completion, and the hippocampus. *Behavioral Neuroscience*, 113(5):867, 1999. 43, 44

[74] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. 29, 42, 69

[75] Alexei Samsonovich and Bruce L. McNaughton. Path integration and cognitive mapping in a continuous attractor neural network model. *The Journal of neuroscience : the official journal of the Society for Neuroscience*, 17(15):5900–20, August 1997. 5

[76] Daniel L Schacter and Donna Rose Addis. The cognitive neuroscience of constructive memory: remembering the past and imagining the future. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 362(1481):773–86, May 2007. 6

[77] Nicol N Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7):1723–38, July 2002. 67, 69

[78] William Beecher Scoville and Brenda Milner. Lss of recent memory after bilateral hippocampal lesions. *J. Neurol. Neurosurg. Psychiat.*, 20(11), 1957. 1

[79] Jonathan Richard Shewchuk. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Technical report, 1994. 56, 63

[80] Trygve Solstad, Edvard I. Moser, and Gaute T. Einevoll. From Grid Cells to Place Cells : A Mathematical Model. *Hippocampus*, 1031:1026–1031, 2006. 18, 20

[81] Larry R. Squire. Memory systems of the brain: a brief history and current perspective. *Neurobiology of learning and memory*, 82(3):171–7, November 2004. 1, 42

[82] Richard S. Sutton and Andrew Barto. *Reinforcement Learning*. MIT Press, 1998. 44

[83] Alessandro Treves, Ayumu Tashiro, Menno P. Witter, and Edvard I. Moser. What is the mammalian dentate gyrus good for? *Neuroscience*, 154(4):1155–72, July 2008. 7, 27, 43

[84] Endel Tulving. *Elements of episodic memory*. OXford University Press, New York, 1983. 1, 3

[85] Matthijs A. A. van der Meer. Covert expectation-of-reward in rat ventral striatum at decision points. *Frontiers in integrative*, 3(February):1–15, 2009. ix, 2, 8, 9

[86] Matthijs A. A. van der Meer and A. David Redish. Expectancies in decision making , reinforcement learning , and ventral striatum. *Frontiers in Neuroscience*, 4(1):29–37, 2010. 2, 8, 9, 10, 34, 43, 46

[87] Gene V. Wallenstein, Howard Eichenbaum, and Micheal E. Hasselmo. The hippocampus as an associator of discontiguous events. *Trends in neurosciences*, 21(8):317–23, August 1998. 46

[88] Paul J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. 29

[89] Ronald J. Williams and David Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. In *Back-propagation: Theory, architectures and Applications*, pages 1–45. 1995. 29

[90] David J. Willshaw and Joseph T. Buckingham. An assessment of Marr's theory of the hippocampus as a temporary memory store. *Philosophical transactions of the Royal Society of London*, 329(1253):205–215, 1990. 42

[91] Matthew A. Wilson and Bruce L. McNaughton. Dynamics of the hippocampal ensemble code for space. *Science*, 261(5124):1055–1058, 1993. 5

[92] Matthew A. Wilson and Bruce L. McNaughton. Ractivation of Hippocampal Ensemble Memories During Sleep. *Science*, 265(July):676–679, 1994. 2