Computation with Spiking Neurons

Sebastian A. Wills
Clare College
Cambridge

A dissertation submitted in candidature for the degree of Doctor of Philosophy,
University of Cambridge

Inference Group
Cavendish Laboratory
University of Cambridge

September 2004

DECLARATION

I hereby declare that my dissertation entitled "Computation with Spiking Neurons" is not substantially the same as any that I have submitted for a degree or diploma or other qualification at any other University.

I further state that no part of my dissertation has already been or is being concurrently submitted for any such degree or diploma or other qualification.

Except where explicit reference is made to the work of others, this dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration. This dissertation does not exceed sixty thousand words in length.

Date: .... Signed: ....

Sebastian A. Wills
Clare College
Cambridge
September, 2004

Abstract

We develop two artificial neural network models which use 'spiking' neurons to perform recognition and memory tasks.

The first task studied is that of recognising spoken words. We present a network built from approximately 1000 biologically plausible integrate-and-fire neurons which was developed as the winning solution to an academic competition posed by John Hopfield and Carlos Brody. The network recognises a vocabulary of ten words, regardless of the speed at which they are spoken ('time-warp invariance'). The network employs two key mechanisms: transient equality in the firing rates of subsets of neurons, and the inducement of synchrony among groups of weakly-connected neurons.

In the remainder of the thesis we study an autoassociative memory. The network consists of a single layer of spiking neurons with recurrent time-delay connections. Each neuron is built from a number of *coincidence detector* subunits, and can be thought of as an approximation to a binary 'sigma-pi' unit. The network stores and recalls spatiotemporal spike sequences (patterns of neural activity over time). When stimulated with a noisy or incomplete version of a stored memory, the network will 'clean up' the memory and, if periodic spike sequences are used, repeatedly replay the corrected, completed memory. Unlike most memory models, the system is able to simultaneously recall more than one stored memory, using a single population of neurons. We explore the capacity properties of this 'Concurrent Recall Network'.

Using a mapping from real numbers to spatiotemporal spike sequences, we extend the Concurrent Recall Network into a *line attractor* memory. In this mode of operation, it is able not only to store individual memories (point attractors) but also a continuum of memories (a line attractor). After training, the network can stably recall the spike sequence corresponding to any point along the attractor. Furthermore, by applying a global 'velocity control' signal to all neurons in the network, we can cause the recalled state to drift along the attractor in a controlled fashion.

Finally, we demonstrate the ability of the Concurrent Recall Network to store multiple line attractors, recall different positions along each attractor concurrently and independently move along each attractor.

Acknowledgements

Above all, I am grateful to David MacKay, who has provided a constant stream of ideas, encouragement and feedback throughout my studies in the Inference Group. I would also like to thank Keith Briggs at BT Exact, and the other members of the Inference Group, especially Sanjoy Mahajan, John Winn, Edward Ratzer, Phil Cowans, Hanna Wallach and David Stern, for interesting discussions and moral support.

I am indebted to Samira Barakat, Tom Counsell, Phil Cowans and David Stern for their comments on drafts of this thesis.

I am grateful to BT Exact and EPSRC for funding my research.

Finally, I would like to thank my wife, Abi, for her love and support, and for sharing her life with me.

Contents

Chapter 1

Introduction

1.1 Modelling networks of spiking neurons

The human brain contains around 100 billion highly interconnected neurons. Information is transmitted between neurons in the form of *action potentials* (short electrical pulses which propagate along the interconnecting axons), also known as *spikes.* How this 'wetware' is able to perform the feats of perception and memory to which we are accustomed is not yet understood.

The aim of this thesis is to develop theoretical models of computation with spiking neurons. The motivation for doing so is two-fold. Firstly, we hope that such models may contribute to our understanding of the mechanisms by which real brains operate. Secondly, models of computation that diverge from those conventionally deployed in artificial systems are interesting in their own right. Even if it turns out that they bear no similarity to the operation of biological neural networks, we may gain insights into the nature of computation itself. Bearing both of these aims in mind, the approach we take in this thesis is not to limit the models under study to those which precisely match the known physiology of real brains, but nonetheless to use models which are *biologically plausible.* We take this approach in the belief that even relatively simple models of biological neurons may be sufficient to capture key aspects of the computational processes used in the brain, and that to focus on building models consistent with every known fact about the brain would be a hindrance to finding general principles of computation with spiking neurons.

The 'spiking' nature of most types of biological neuron has been known since the first experiments of Adrian in the 1920s [5, 6, 7], who observed that, rather than having a continuous, analogue output, neurons respond to stimuli by 'firing' one or more action potentials (spikes).[1] Adrian also found that the sensory neurons that he studied fire spikes at a rate which increases monotonically with the intensity of stimulus. This observation led to the widespread adoption of the 'rate coding hypothesis' — the idea that all neurons in the cortex communicate purely through their rate of firing.

[1]In contrast, a typical Artificial Neural Network (ANN) has neurons which have real-valued inputs and outputs.

While it is indisputable that rate coding is used in the functioning of the brain, there is mounting evidence that the precise timing of individual spikes also plays an important role [67]. It is within this 'temporal coding' framework that the models explored in this thesis lie.

1.2 Thesis outline

Speech, being a temporal form of sensory input, is a natural candidate for investigating temporal spike coding in neural networks. It is only through comprehension of the temporal relationship between different sounds which make up a spoken word or sentence that speech becomes intelligible. At some level, the processing by the brain of the incoming sensory input from the ear must deconstruct these temporal relationships in order to decode what is being said. In chapter 2, we describe an artificial, biologically plausible neural network capable of performing recognition of spoken words. The network blurs the distinction between rate coding and temporal coding. Firing rates are used to represent the elapsed time since a particular auditory feature was detected. But the word as a whole is recognised by the transient formation of temporal synchrony — phase locking — between sets of weakly-connected neurons.

Synchrony among coupled oscillators is observed in many situations both in the laboratory and in nature [81]. The pendula of identical pendulum clocks placed near to each other are known to synchronise due to extremely weak coupling from air movements and vibrations [38, 39]. Synchrony also occurs between organisms. In one spectacular example from South East Asia, thousands of flashing fireflies in a tree will synchronise their flashing, because the fireflies' internal chemical cycle is advanced in phase by flashes of light from other fireflies [59].

In the brain, synchrony of spikes plays a critical role in the echo-location circuits in bats [50], the electro-sensory system of electric fish [27] and sound localisation in birds and mammals [44, 18]. Synchrony has also been observed more generally among subsets of neurons in mammalian brains, and it has been suggested that synchrony may be used to 'bind' different items of sensory information, i.e. to signify that they are all properties of the same object [77, 67, 47]. The model studied in chapter 2 increases our understanding of the ways in which synchronised spiking can be used to recognise, store or compute information in a network of neurons.

Chapter 3 introduces a different spiking neural network model, the Concurrent Recall Network, built from a single layer of spiking neurons with recurrent time-delay connections and coincidence detector subunits. Whereas the model in chapter 2 used collective synchrony of large groups of neurons to perform recognition of a stimulus, the model in chapter 3 uses the coincidence (synchrony) in the arrival time of small clusters of spikes to implement an autoassociative memory.[2] In a typical neural network implementation of an autoassociative memory, a memory is stored and recalled in the form of the level of activity of all the neurons

[2]Autoassociative memories are introduced at the beginning of chapter 3.

in the network. In this system, a memory takes the form of a spatiotemporal pattern of spikes — in other words, the precise spiking activity of a population of neurons over a given period of time.

The network of chapter 3 is able to take advantage of this representation of information by simultaneously recalling more than one stored memory, using a single population of neurons. Such a feat is not possible in most models of memory. In a rate-based model, a neuron cannot simultaneously have several different firing rates,[3] but it can fire at several different moments during a 100ms period. Chapter 4 explores the capacity of the Concurrent Recall Network through simulation and analytical results.

In chapter 5 the Concurrent Recall Network is extended from a memory that can store individual memories (point attractors in the high-dimensional space of all possible spatiotemporal spike sequences) into a memory that can store one or more *line attractors*. This gives the network the ability to store a *continuum* of related memories. For example, an organism could use such a facility to robustly store a continuous variable such as the angle of one of its limb joints. Equally, the line attractor could represent a physical route through the organism's environment, with stimuli and memories associated with different points along the route. The network is also capable of controlled motion along the line attractor, turning it into an *integrator* of a 'velocity control' signal. Such a network could therefore be used to mentally move along a memorised route in a controlled fashion, or to integrate a motion signal emanating from another brain area to compute the new limb position.

[3]As Izhikevich has pointed out [41, 37], some forms of interaction between weakly-coupled neurons depend on their relative intrinsic firing frequencies; the synchronisation between neurons in the model in chapter 2 is one example. This raises the possibility of superimposing frequency-encoded information in much the same way as is done with FM radio broadcasts. However, a simple measurement of firing rate cannot disentangle information combined in this fashion.

Chapter 2

Speech Recognition with Spiking Neurons

2.1 Introduction

Humans can recognise words spoken by unfamiliar speakers in noisy environments, whether spoken slowly or rapidly. Automatic speech recognition aims to bring these abilities to machines. The algorithms employed for computer speech processing are typically based on Hidden Markov Models [45]. While algorithms such as these may offer good performance for artificial recognition, they bear little resemblance to how the brain decodes spoken words.

This chapter explores an alternative recognition mechanism, which uses an artificial neural network to recognise temporal patterns. The network is biologically plausible, so offers insight into how the brain may recognise words or other temporal patterns. It is trained on a single example of each word, and is then able to recognise speech from different speakers, spoken at different speeds.

The origin of this research is a competition set in October 2000 by Professor John Hopfield and Dr Carlos Brody. They constructed an artificial neural network from approximately 1000 biologically plausible spiking neurons. The network could recognise ten spoken words, regardless of whether they were spoken slowly or quickly ('time-warp invariance'). They described this artificial organism, which they called *mus silicium*, in a paper [34] revealing only data that could have been obtained experimentally had the neural network been part of a real organism's brain (that of a mouse, say). The paper included plots of the (simulated) outputs of various individual neurons while the organism was being presented with different stimuli, as well as observations about the broad pattern of connectivity they "observed". Neither the precise connectivity of the network nor the reasoning behind the network's construction were revealed.

Hopfield and Brody proposed two competitions. Competition A was to deduce and describe the mechanism by which their organism performed word recognition. Competition B involved creating an artificial network with a similar number of spiking neurons to Hopfield

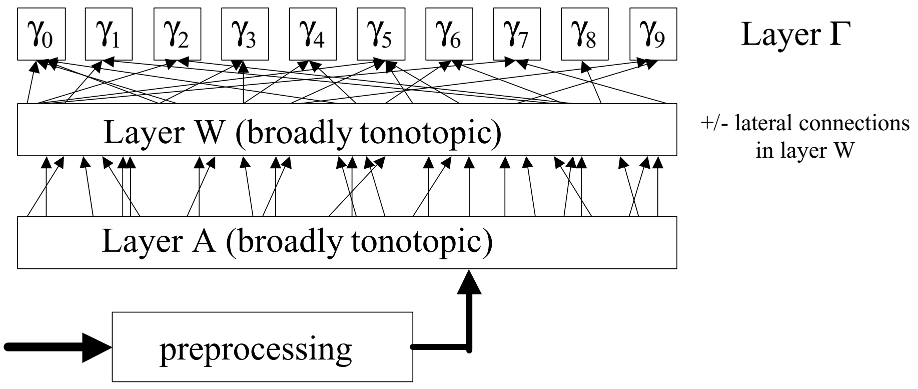

γ0
γ1
γ2
γ3
γ4
γ5
γ6
γ7
γ8
γ9
Layer Γ
Layer W (broadly tonotopic)
+/- lateral connections in layer W
Layer A (broadly tonotopic)
preprocessing

Figure 2.1: Architecture of the *mus silicium* network. Layers A and W are broadly tonotopic, i.e. a neuron's position in the layer determines approximately what frequencies of sound it responds to. In addition to the feed-forward connections illustrated, layer W contains lateral (within-layer) connections of both excitatory and inhibitory types.

and Brody's that could perform the same task. The entry achieving the lowest error score in the recognition task was the winner. Competition A was solved collectively by our research group [53] and the problem provided a convenient focus for research into computation with spiking neurons. This research, including the winning entry to competition B which resulted from it, is described in the remainder of this chapter.

We begin with a brief summary of the information about *mus silicium* that appeared in Hopfield and Brody's original paper and supplementary online material [34, 61]. Sections 2.3 and 2.4 describe how *mus silicium* is able to recognise words. Finally, we present the results from our own implementation of *mus silicium* .

2.2 Inside *mus silicium*

The network consists of three layers of neurons: an input layer (A) fed by a preprocessing unit, a middle layer (W) and an output layer (Γ) (figure 2.1). Each layer has feed-forward connections to the next layer. Layer W additionally has lateral (intra-layer) connections. The following sections describe each layer in more detail.

2.2.1 Preprocessing

The preprocessing unit is fed as its input a short sound sample (typically one spoken word) and detects the occurrence of 40 different features in the following way. The input sound first undergoes spectrographic analysis by Fourier transform, akin to the way in which the cochlea in the ear determines the frequency spectra of sounds. The frequency spectrum is divided into 20 frequency bands. For each band, the times of three events are determined: *onset* (the start of the first period during which there is significant energy in that frequency band), *offset* (the end of that period) and *peak* (the first maximum of energy in the band) (figure 2.2). Of the resulting 60 different events, a fixed choice of 40 (specified by Hopfield and Brody) form

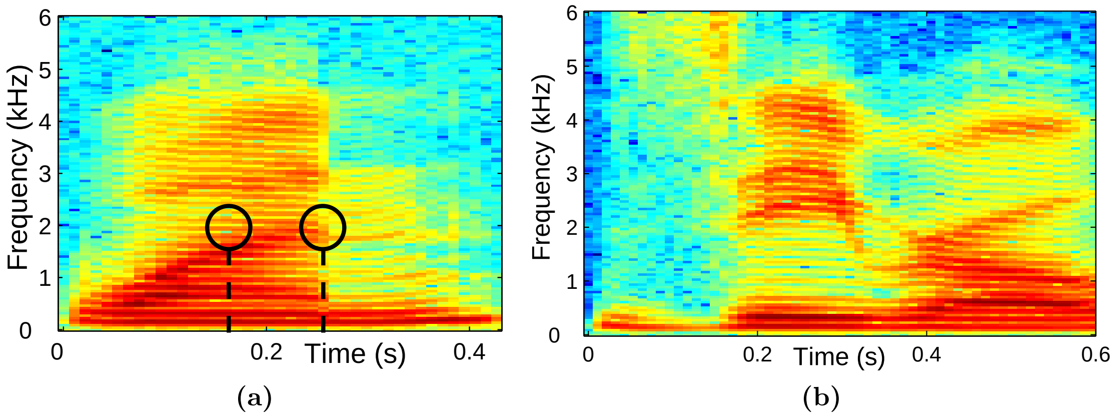


Figure 2.2: Spectrograms show the intensity of sound across the frequency spectrum with time. Red pixels denote highest intensity. **(a)** A speaker saying "one". The onset and offset times for a frequency band centred around 2kHz are shown by black circles as an example. See text for details. **(b)** A speaker saying "zero", for comparison.

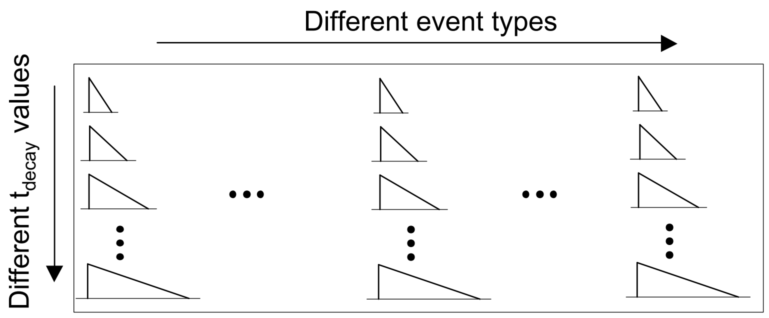


Figure 2.3: Map of layer A. For each of 40 event types (columns in this figure) there are 20 different outputs from layer A, with different values of t_{decay}. Each miniature set of axes shows how the current emitted on one output varies as a function of the time since its associated event occurred.

the final output of the preprocessing unit. The times of occurrence of these events is used to trigger activity in layer A.

2.2.2 Layer A

The purpose of layer A is to generate 800 analogue output currents, which can be used to drive neurons in layer W. We first describe the form of currents required, then describe a mechanism which could be used to generate them using spiking neurons.

The 800 output currents are divided into 40 equally sized groups, corresponding to the 40 types of event detected in the preprocessing stage. When an event occurs, all output currents in the group associated with that event rise to a value of 1 (in arbitrary units). Each current then decays linearly to zero, reaching zero a time t_{decay} later. The 20 output currents in each group all have different values of t_{decay}, ranging from 0.3 to 1.1 seconds, but the same 20 values of t_{decay} are used in each group of 20. Figure 2.3 summarises the setup. The end result is that layer A provides a set of 800 different analogue output currents: each current rises to

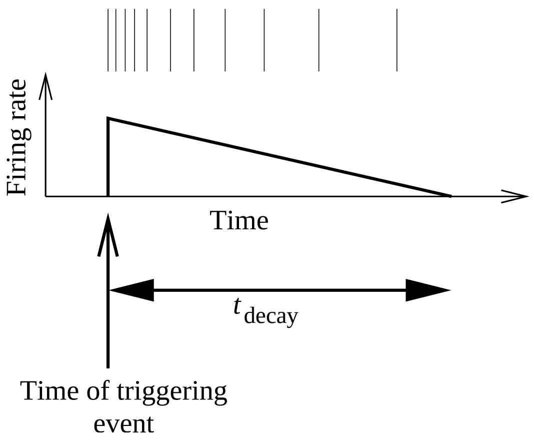


Figure 2.4: Illustration of the behaviour of a neuron which could be used to generate the required output currents from layer A. Spike raster (top) and firing rate (main axes).

1 when one particular type of event occurs in the stimulus and decays linearly back to zero at a particular rate.

The generation of output currents from layer A could be considered part of the preprocessing of the stimulus before it reaches the main network of spiking neurons in layer W. However, it is possible to generate the required currents using biologically plausible spiking neurons. For each of the 800 output currents that layer A is required to produce, we envisage having a large number, M, of identical spiking neurons. Thus, each set of M neurons is associated with the same stimulus event and has the same value of t_{decay}. The neurons have a firing rate which is adaptive in the following way. When a neuron's associated event occurs, it begins to fire periodically at a particular rate, to which we assign the value 1 in arbitrary units. The firing rate then decays linearly to reach zero t_{decay} later. Figure 2.4 illustrates the output of one such neuron.

Owing to noise or tiny variations in the time at which each neuron starts to fire, the precise firing times (phases) of the M neurons which have the same triggering event and the same value of t_{decay} will not be equal; only their firing rates will be the same. Since each neuron outputs a short pulse of current when it fires, the total, 'collective', current output of all M identical neurons will, if M is large enough, be a smooth, analogue current which varies in the manner required for one of the 800 outputs from layer A.

A decaying firing rate of the kind described above is not biologically implausible: spike-frequency adaptation is a commonly observed property of cortical neurons. For example, one study of visual cortex neurons in an anaesthetised cat found that the firing rate in response to the onset of constant input is well described by an exponential decay to steady-state [9]. Neurons such as these could provide the decaying firing rate in layer A which *mus silicium* requires. As will be noted in section 2.3.2, *mus silicium* does not require the firing rate decay of layer-A neurons to be linear.

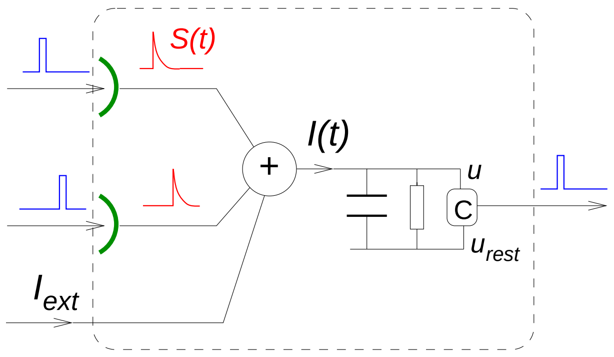


Figure 2.5: Schematic illustration of an integrate-and-fire neuron. The dashed box encloses one neuron. The neuron receives spikes (shown as blue delta functions) at its many synapses (two are shown). Each synapse converts incoming spikes into a *post-synaptic current* $S(t)$, which can be viewed as the impulse response of the synapse. The post-synaptic currents from all synapses, plus any external current I_{ext}, are summed, and then fed into a capacitor which can slowly discharge through a leakage resistance. A comparator (marked C) monitors the voltage drop across the capacitor; if it rises above a set threshold, the charge on the capacitor is reset to a particular reset value and a spike is emitted on the neuron's output. Typically, each neuron's output fans out to make synapses with multiple other neurons.

2.2.3 Integrate-and-fire neurons

The neurons in the W and Γ layers of the network are non-adaptive *leaky integrate-and-fire neurons* [22], a standard neuron model which we summarise here and in figure 2.5.

Each neuron has a membrane potential, $u_i(t)$, which obeys the following equation while $u_i(t)$ is less than a threshold $u_{\text{threshold}}$:

$$\frac{d}{dt}u_i(t) = \frac{1}{C}\left(-\frac{u_i(t) - u_{\text{rest}}}{R} + I_i(t)\right), \tag{2.1}$$

where u_{rest} is the resting potential of the neuron, R denotes the leakage resistance, C the membrane capacitance, and $I_i(t)$ the total current flowing into the neuron. The neuron acts as a leaky integrator of the current flowing into it, with time constant RC.

If the potential reaches u_{trigger}, the neuron 'fires'. The consequences of firing are:

1. The potential, $u_i(t)$, is reset to a value given by u_{reset};

2. The neuron enters an *absolute refractory period*, during which time the input current $I_i(t)$ is ignored.

3. The neuron emits a spike on its output.

Neurons are connected at *synapses* which connect the output of one neuron (referred to as the *pre-synaptic* to the input of another (the *post-synaptic* neuron). The synapse between the output of pre-synaptic neuron i and the input of post-synaptic neuron j has an efficacy w_{ij}, also referred to as the *weight*, or strength, of the connection between the two neurons. $w_{ij} = 0$ denotes that there is no connection from neuron i to neuron j.

When neuron i emits a spike, an extended current pulse of the form $S_i(t)$ is generated at synapses for which i is the pre-synaptic neuron (actual forms of $S_i(t)$ used in *mus silicium* are given in the next section and plotted in figure 2.12). The amplitude of the current pulse is scaled by the connection weight, resulting in the *post-synaptic current* which then contributes to the input of the post-synaptic neuron. A neuron's total input current, $I_i(t)$, is therefore given by

$$I_i(t) = \sum_j \sum_{f^{(j)}} w_{ji} S_j(t - t_{f^{(j)}}) + I_{\mathrm{ext},i}(t) \tag{2.2}$$

where j is an index over all neurons in layers W and Γ, $f^{(j)}$ is an index over all firings of neuron j, and $t_{f^{(j)}}$ is the time of the fth firing of neuron j. $I_{\mathrm{ext},i}(t)$ represents current flowing into the neuron from sources other than other integrate-and-fire neurons. In *mus silicium*, neurons in layer W receive such an external input from a single layer-A output. The magnitude of this external current reaches much higher values than does the current contributed by the post-synaptic currents caused by impinging spikes from other neurons.

The effect of an individual post-synaptic current on the post-synaptic neuron's potential, i.e. the cumulative 'leaky integral' of the post-synaptic current, is known as the *post-synaptic potential* (PSP). The actual forms of the post-synaptic currents used in *mus silicium* are given in the next section and plotted in figure 2.12.

Note that if the input current $I_i(t)$ is a constant and has a value larger than $(u_{\mathrm{trigger}} - u_{\mathrm{rest}})/R$, the neuron will fire periodically at a rate dependent on $I_i(t)$ as the membrane potential repeatedly charges up to u_{trigger} and is reset (figure 2.6). We can therefore think of a strongly-driven integrate-and-fire neuron as an oscillator. The relationship between input current and firing rate is shown in figure 2.7.

In simulations, Gaussian noise with standard deviation 0.2mV is added to the membrane potential of every neuron independently at each simulation time step (every 0.1ms). Values of u_{rest}, $u_{\mathrm{threshold}}$ and u_{reset} are -65mV, -55mV and -75mV respectively, the time constant of W-neurons (the product RC) is 20ms and the absolute refractory period is 2ms.

2.2.4 Layer W

The middle layer, W, consists of 800 integrate-and-fire neurons of the type described in the previous section. 50% of the neurons are *excitatory* (α-neurons), meaning that they increase the internal potential of other neurons to which they are connected when they fire. A pre-synaptic α-neuron induces a post-synaptic current which takes the form of a decaying exponential with a time constant of 2ms. The other 50% of neurons in layer W are *inhibitory* (β-neurons). The post-synaptic current induced by a β-neuron takes the form of a negative alpha function ($-te^{-t/6\mathrm{ms}}$), and is of the opposite sign to the post-synaptic current induced by an α-neuron. Both post-synaptic currents are plotted in figure 2.12; their precise roles will be discussed in section 2.4.2.

Each α-neuron receives an input from a single layer-A output. This forms I_{ext} in equa-

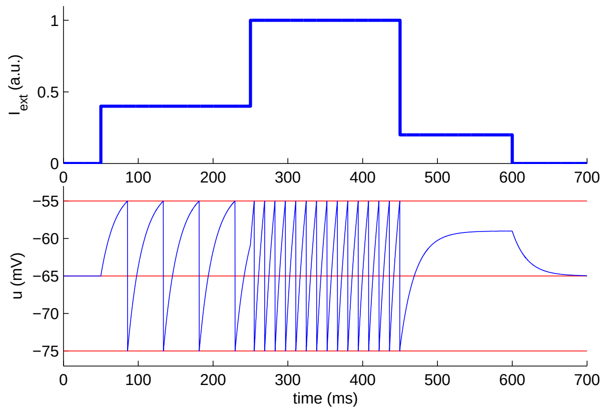


Figure 2.6: Demonstration of the effect of different values of a constant driving current on an integrate-and-fire neuron. The neuron's potential, $u(t)$, is shown on the bottom axes. $u_{\text{threshold}}$ is -55mV, u_{rest} is -65mV and u_{reset} is -75mV. The top axes show the input current to the neuron. The neuron does not receive input from other neurons, so the total input current (equation 2.2) consists only of an external driving current I_{ext}. A value of 1 corresponds to the maximum output of a layer-A output current. For driving currents larger than $(u_{\text{trigger}} - u_{\text{rest}})/R$, which corresponds to 1/3 in these units, the potential u is repeatedly charged up to the firing threshold and subsequently reset, leading to periodic firing of the neuron. For input currents below this value, the driving current is not sufficient to drive the potential above threshold, and a steady-state is reached.

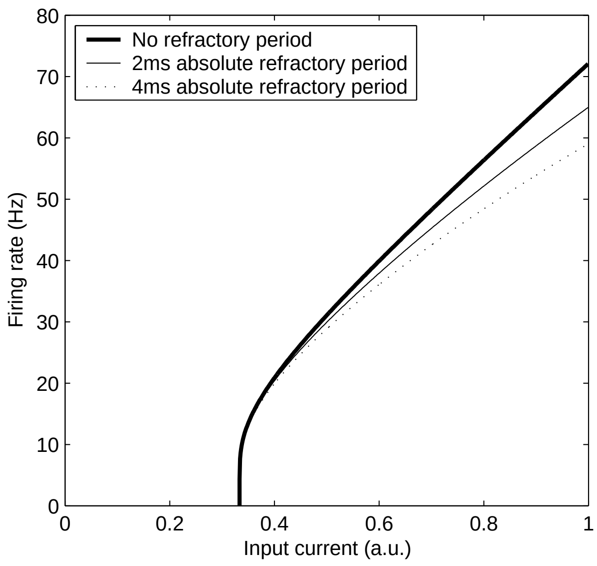


Figure 2.7: Firing rate as a function of constant input current magnitude for the integrate-and-fire neuron model used in *mus silicium*. The current is shown in arbitrary units, with 1 being the maximum current of a layer-A output.

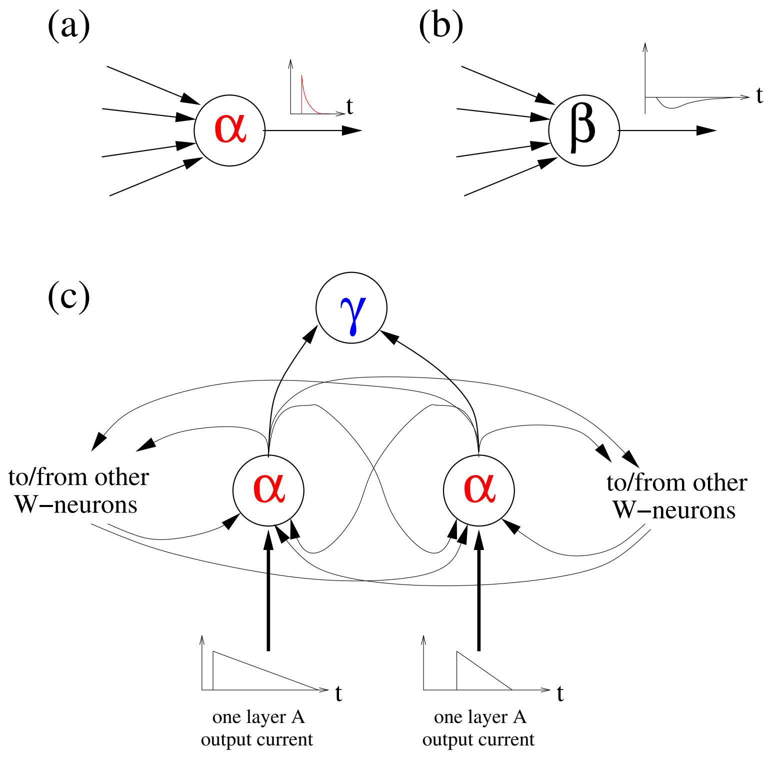


Figure 2.8: Spiking neurons in layer W. **(a)** Schematic representation of an excitatory (α) neuron. The neuron receives input from many other neurons. When the neuron spikes, it contributes a positive post-synaptic current (inset axes) to the neurons its output is connected to. **(b)** An inhibitory (β) neuron behaves like an α-neuron except that the post-synaptic current it induces in neurons which receive its spikes is of the opposite sign and has a different form, as shown. **(c)** Typical connections among W- and γ-neurons in *mus silicium*. Each W-neuron receives a strongly-weighted input from layer A, plus many weaker connections from other W-neurons. Each W-neuron's output feeds a single γ-neuron (as well as other W-neurons); as there are far fewer γ-neurons than W-neurons, each γ neuron receives input from a large number of W-neurons. Only α-neurons have been shown in the W layer for clarity; in the real network a β-neuron accompanies each α-neuron.

tion 2.2. Additionally, each α-neuron has connections to and from numerous other α-neurons in layer-W. Figure 2.8(c) summarises the types of connections to and from α-neurons in layer W; a precise specification of the required connections will be given when we discuss how *mus silicium* is able to perform speech recognition in section 2.3.

In *mus silicium*, each β-neuron operates as a partner to an α-neuron. The nature and purpose of this partnership will be discussed in section 2.4.2; for the moment it suffices to focus on α-neurons alone.

2.2.5 Layer Γ

The output layer, Γ, consists of ten neurons — one for each word the network is to recognise. These 'γ' neurons have similar dynamics to W-neurons, but have a lower leakage resistance (shorter time constant). Each γ-neuron receives input from multiple neurons in layer W.

The goal of *mus silicium* is for each γ-neuron to detect when the network is presented with a sound similar to one of ten training words. In particular, it should respond regardless of the speed at which the word is spoken (within certain limits). A γ-neuron should respond to its associated word by firing a burst of spikes close to the end of the stimulus word, as shown in figure 2.14. In the next section, we explain how the neural hardware we have described in this section can be used to perform this task.

2.3 Recognition through transient synchrony

In the previous section, we described the neural hardware of *mus silicium*, as outlined in Hopfield and Brody's first paper [34]. The first part of their competition was to deduce how this system could perform recognition of spoken words in a manner which is invariant to the speed at which they are said.

The basic principles of operation of the network were deduced collectively by members of our research group [53]. In this section, we develop these principles into a detailed description of how *mus silicium* operates. Hopfield and Brody provided their own explanation of how their system functions in a second paper [35] published after the close of the competition.

2.3.1 Overview

Training *mus silicium* to recognise ten words involves setting up the connections from layer A to layer W, within layer W, and from layer W to layer Γ. The task of the training algorithm is to determine, given the ten words that the network is required to recognise, the precise connectivity of neurons in the network. Since an understanding of the training rule requires an understanding of the overall mechanism by which *mus silicium* operates, we will introduce the training rule gradually: as we see how *mus silicium* operates, we will also see which connections are required. Later, in section 2.4.3, we discuss the training rule in more detail.

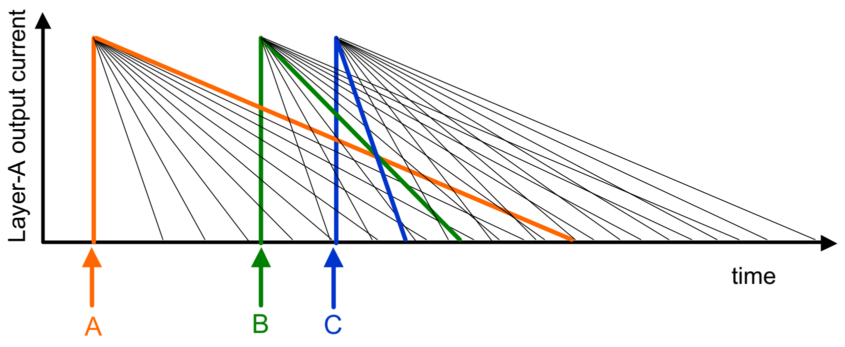


Figure 2.9: Selection of layer-A outputs for an example word with three features. The plot shows the outputs of three groups of layer-A outputs, which are triggered by events A, B, and C at the times indicated. From the range of slopes (values of t_{decay}) available, the training algorithm selects one output from each group (coloured lines) such that the three selected outputs have intersecting current trajectories.

2.3.2 Coincidence and time-warp

A spoken word, after preprocessing as described in section 2.2.1, consists of a sequence of events over time. Recognising a learnt word means detecting the presence of the correct features (events) in the correct temporal relationship to each other.

We now show how the selection of a collection of output currents from layer A forms the first stage of detecting a temporal sequence of features.

Imagine a word with three features: a, b and c. Each of these three features triggers a set of 20 layer-A output currents, with 20 different values of t_{decay}. Assuming the period of time over which the events occur is less than the range of t_{decay} values in layer A, there will be at least one triplet of output currents — one output from each group of 20 — that have decay rates such that the currents are approximately equal at some later time (figure 2.9). This equality of currents provides evidence that the word was present. In general there will be more than one possible intersection. In the training rule used in this chapter, we define the 'target' intersection point for a given training examplar to be at the intersection of the slowest-decaying output current from the first feature to be activated with the most rapidly-decaying current from the last feature to be activated. The output current selected for each remaining feature is the one which passes through the target intersection current closest in time to the target intersection time.

Leaving aside for the moment the question of how the equal output of a group of layer-A output currents leads to the firing of a γ-neuron, we are now in a position to see how the time-warp invariant nature of *mus silicium* works.

If a word is spoken more slowly, the same sequence of features will be present, but their times of occurrence will be expanded in time.[1] We take the case in which there is a linear

[1]If the same recorded sound sample were played at an altered speed, it would not be true that the same set of features would be activated. The pitches of the sounds making up the word would also be altered, causing a different set of features to be activated. When a person speaks more quickly or slowly, however, the pitches of the sounds produced remain the same.

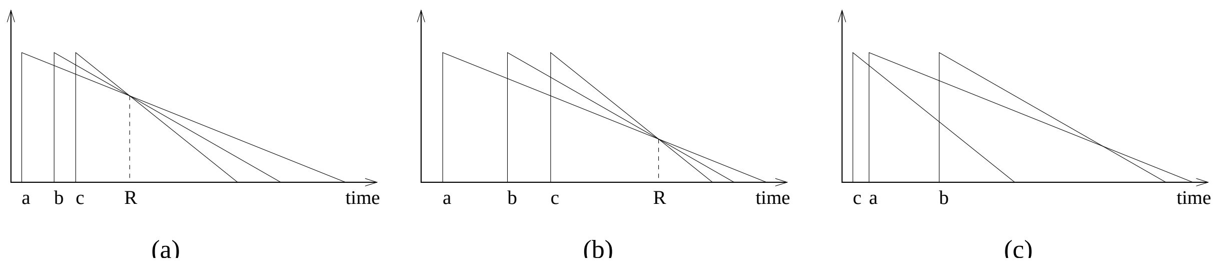


Figure 2.10: Time-warp invariance. **(a)** Three features occur at times a, b, c. Three layer-A output currents are selected with decay rates such that the currents coincide at a later time (R). The vertical axis represents either the layer-A output currents, or the firing rates of corresponding layer-W neurons to which they are connected. **(b)** The same word spoken more slowly: the events a, b, c are linearly stretched along the time axis but the gradients of the lines remain unchanged. Coincidence still occurs, now at a different position on the y-axis. **(c)** When features a, b, c occur at random times (such as when a different word is presented), the three lines are unlikely to intersect.

scaling of the times of the events making up a word, which is the form of time-warp that *mus silicium* can naturally cope with. For our example word made from three events, the relative timings of the three events a, b and c are stretched linearly, however the rates of decay of the three layer-A output currents which we identified remain unchanged since rate of decay is a property of which layer-A output was selected and not of the input sound. As can be seen in figure 2.10, in this linearly time-warped situation the outputs of the three currents still coincide. The output current at which they coincide is different, and the time of coincidence is shifted in relation to the event times by the same time-warp scale factor.

The property that the currents will still coincide after their triggering events have been subjected to linear time-warp does not depend on the decay of the output currents over time being linear. All that is required is that the ith output current is given by $f((t-t_e^{(i)})/t_{\text{decay}}^{(i)})$, where $f(\cdot)$ is any single-valued function and $t_e^{(i)}$ is the time of the triggering event. Then, if $f((t_c-t_e^{(i)})/t_{\text{decay}}^{(i)}) = f((t_c-t_e^{(j)})/t_{\text{decay}}^{(j)})$ (i.e. two outputs are equal at the coincidence time t_c), it follows that $f((\alpha t_c-\alpha t_e^{(i)})/t_{\text{decay}}^{(i)}) = f((\alpha t_c-\alpha t_e^{(j)})/t_{\text{decay}}^{(j)})$ (i.e. the same two outputs, if their event times are rescaled by α, are equal at the new coincidence time αt_c).

The time-warp invariance property of the decaying layer-A output currents means that if, when training the network to recognise our imaginary three-feature word, we select the three particular layer-A outputs whose outputs will decay to reach a particular value at some particular time after the third event, then those currents will still have coinciding outputs if the word is spoken with a linear time-warp. In fact we have 40 features per word, not three. For each one, the training algorithm must select 40 layer-A outputs which, when triggered by the events in the training word, have outputs that all reach a particular value at a particular moment soon after the end of the last event.

2.3.3 From equal currents to equal firing rates

Each of the 40 selected layer-A outputs is connected to the input of a single pair of W-neurons: an α-neuron and a β-neuron that work in tandem as will be discussed in section 2.4.1. As noted in section 2.2.3, an integrate-and-fire neuron driven by a continuous input current above a certain minimum value will fire periodically at a rate dependent on the current (figure 2.7). Consequently, the decaying analogue output current of each of the 40 layer-A outputs is translated into the decaying *firing rate* of a pair of W-neurons. At the 'coincidence time', when the layer-A output currents are equal, there will therefore be 80 W-neurons (40 pairs) which have identical firing rates.

2.3.4 Synchrony

So far, we have seen how it is possible to have a set of 40 pairs of W-neurons whose firing rates become momentarily equal if one particular word is presented to the network. As a word is played to the network, different W-neurons in the set will suddenly start firing as the spectrographic feature associated with the layer-A output which is driving them is detected by the preprocessing unit. Their firing rate will then decay as their driving current from layer A decays. Only if the word being played was the training word associated with that set of W-neurons will the temporal pattern of feature detections match up with the decay rates of the 40 neurons to create a brief moment when the firing rates are all equal. If any other word is present, the firing rates will never collectively intersect as they decay.

How is this transient equality of firing rates exploited by the network to result in the firing of a γ-neuron? The size of the post-synaptic potential induced in a γ-neuron when it receives a spike from a W-neuron is small compared to the increase in potential needed to cause the γ-neuron to fire. Furthermore, the leaky integrator inside a γ-neuron has a short time-constant, so it is not able to accumulate post-synaptic currents induced by incoming spikes from W-neurons over a long period.

If, however, a γ-neuron receives *simultaneous* spikes from large number of W-neurons, the combined post-synaptic current will be large enough to immediately force the γ-neuron over threshold and cause it to fire. Effectively, we use the γ-neuron as a detector of synchrony among the set of W-neurons from which it receives connections.

This immediately suggests the further step that is required to complete the chain of events in *mus silicium*. As things stand, there is moment when, if the trained word is presented, our set of 40 pairs of W-neurons have equal firing rates. They will not, in general, be firing exactly in synchrony, since each neuron will be at a different phase in its oscillation. If we can induce the set of W-neurons to fully synchronise when their firing rates are equal, then by giving them all a connection to one particular γ-neuron, that γ-neuron will fire a spike each time the set of W-neurons fires.

The set of W-neurons can be caused to synchronise by making weak all-to-all connections between them all, i.e. by setting w_{ij} to a small positive value for all neurons i and j in the set.

As will be discussed further in section 2.4.1 this form of weak coupling between the neurons is sufficient to induce synchrony for the short period during which the neurons have similar firing rates, i.e. either side of the 'coincidence time' at which the firing rates are equal.

2.3.5 Summary of the recognition process

We can summarise the whole process by which a word is recognised in a trained *mus silicium* network as follows. When a word is input to the network, groups of output currents in layer A (each group containing a variety of decay times), are activated by each sound feature within the word as it occurs. In layer W are ten sets (one for each trained word) of 40 W-neuron pairs. Each W-neuron is driven primarily by one particular layer-A output current which causes it to fire at a rate determined by that current, but additionally receives weak post-synaptic currents when other W-neuron pairs in its set fire, due to the weak mutual connections within each set.

If the stimulus word is identical (or similar) to one of the ten which the network was trained to recognise, then the neurons in the corresponding set of W-neuron pairs will all (or mostly) arrive at the same firing rate near the end of the word. The equality of firing rate of the neurons in the set will allow the weak mutual connections to induce synchrony. The same set of W-neurons also has connections to a single γ-neuron, which will respond to the highly synchronous firing of those 40 pairs of W-neurons by firing each time the W-neurons fire, thereby signalling recognition of the word. Once the W-neuron firing rates diverge, synchrony is lost, the inputs to the γ-neuron are once again uncorrelated in time rather than arriving in a series of short, intense pulses, and it will cease to fire.

In reality, variations in the input stimuli, noise in the neuron dynamics and the stochastic nature of the synchronising mechanism explained in the following section will mean that not all the W-neuron pairs in the relevant set will fall into synchrony. Equally, some neurons will synchronise even if their associated word is not being presented. The system is robust against these variations. All we require is that a γ-neuron is able to detect the difference between 'many' and 'few' of its W-neuron pairs firing in synchrony.

2.4 Discussion

Having described in detail how *mus silicium* is able to recognise words, we return to discuss three areas in more detail.

2.4.1 Mechanism for synchrony

The mechanism for the formation of synchrony among a set of mutually-connected integrate-and-fire neurons can be understood in the following way. Consider two α-neurons in layer W which have a weak mutual connection. Initially, we will consider both to be driven by a constant and equal input current, meaning that they have the same intrinsic firing rate. We use the term ‘intrinsic firing rate’ to refer to the rate at which the neuron would fire when driven only by a constant input current, i.e. ignoring the effect of the post-synaptic currents induced by spikes they receive from each other. The intrinsic firing rate as a function of input current is shown in figure 2.7. Unless the neurons happen to start at the same phase in their cycles, they will be firing at the same frequency, but not phase-locked together.

Because the neurons have a weak mutual connection, each time one of them fires, the other one’s internal potential, u, is boosted by the post-synaptic potential (PSP) induced by the firing neuron’s spike, scaled according to the connection weight. Since the coupling is weak, the effect will be small. If, however, the two neurons are *almost* firing in synchrony, for example if neuron 2 fires soon after neuron 1, then the PSP will have a significant effect. When neuron 1 fires, neuron 2 will be almost ready to fire (its membrane potential will be nearly at threshold), and the PSP it receives from neuron 1 will be enough to push it over threshold, causing it to fire immediately after neuron 1. Neuron 2’s phase will have been nudged forward to join in synchrony with neuron 1, and they will remain phase-locked thenceforth.

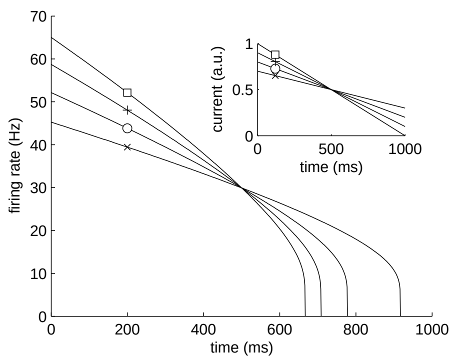
In *mus silicium*, α-neurons are driven by linearly decaying outputs from layer A, so different neurons’ relative intrinsic frequencies are constantly shifting. If one neuron has a firing rate which “overtakes” another one’s, then there will be a brief period when the two neurons have sufficiently similar firing rates for the synchronising mechanism described above to be effective. Synchrony will be maintained while the firing rates are sufficiently similar.[2]

A short study of the criteria under which synchrony can be maintained between a pair of α-neurons[3] was carried out and is summarised in figure 2.11. One α-neuron was driven by a constant current (0.5 in arbitrary units, where 1.0 is the maximum output of a layer-A output current in *mus silicium*; see figure 2.7 for corresponding intrinsic firing rates). The second α-neuron was driven by a linearly decaying current, just like a layer-A output in *mus silicium*. Various decay rates were tried, with the corresponding starting currents chosen so that the decaying driving current would pass through a current of 0.5 500ms after the start of the 1s simulation. The intrinsic firing rates corresponding to the input currents used are shown as a function of time in figure 2.11(a).

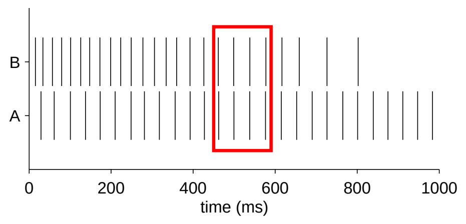
The α-neurons were deemed to have become synchronised if they both fired within 1ms of each other, and were deemed to have fallen out of synchrony if either one fired without the other one firing within 1ms. The times of onset and offset of the longest period of synchrony between the α-neurons during the 1s simulation were computed, and translated into the intrinsic firing rates of the two α-neurons at those times. These values are plotted in figure 2.11(d) as a function of the strength of the coupling between the two neurons. The graph shows that, in the range of parameters explored in this study, the coupling strength between two pairs of W-neurons required to induce them to synchronise is approximately linearly

[2]This dependence of the interaction between neurons on their relative firing frequencies is sometimes referred to as a *frequency modulated interaction* [41].

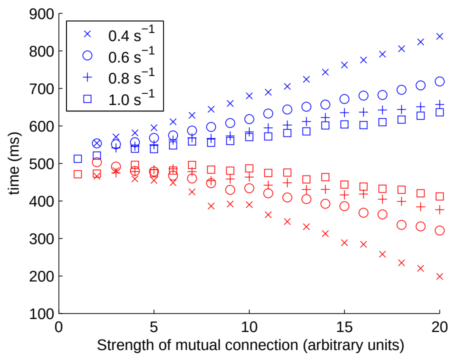
[3]In fact this study used not just two α-neurons, but two pairs of W-neurons, to effect a combined post-synaptic potential which sums to zero, as explained in section 2.4.2. The β-neuron accompanying each α-neuron was connected in such a way that it always fired with its partner.



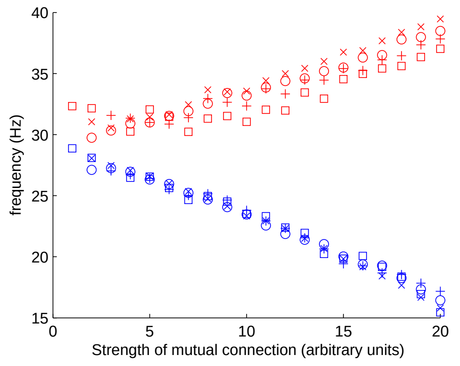
(a)



(b)



(c)



(d)

Figure 2.11: Simulations of two mutually-coupled α-neurons. Neuron A is driven by a constant current of 0.5 units (corresponding to a firing rate of 30Hz). Neuron B is driven by a linearly decaying current passing through a current of 0.5 units at $t = 500$ms. **(a)** Driving currents for neuron B (inset) and corresponding intrinsic firing rates (main axes) as a function of time. **(b)** Example simulated spike times for the two neurons. Neuron B's input current decays at 0.6s^{-1} and the coupling strength is 5 (arbitrary units). The red box encloses the period of synchrony between the neurons. **(c)** Times of onset (red) and offset (blue) of synchrony as a function of the strength of the mutual connection (averaged over 20 trials). The legend shows the rate of change of the difference in driving currents for the two neurons. **(d)** The same data as displayed in (c), with onset and offset times translated into the firing rate of neuron B at those times.

proportional to the difference in their intrinsic frequencies.

The strength of the weak coupling connections within layer W, then, determines how similar the layer-A output currents must be in order for the group of W-neurons to synchronise. This, given the continuously changing nature of the layer-A outputs, determines the duration of the synchronised period, which will be centred around the the moment when the decaying firing rates intersect. In *mus silicium*, we use a value for the connection weight between mutually-coupled W-neurons which results in the pulse height of each individual incoming excitatory post-synaptic potential being approximately 1mV. The qualitative behaviour of the system does not depend critically on the precise value used.

In *mus silicium*, we have groups of not two but around 40 α-neurons which are driven by currents decaying at different rates. Each group of W-neurons is required to detect when all (or most) of their input currents are approximately equal, using the same mechanism described above for two neurons. Initially, a small number of neurons which happen to be 'almost' in synchrony will synchronise because of the effect of each others' PSPs. As the set of synchronised neurons grows larger, their combined PSP has an ever stronger tendency to pull the remaining unsynchronised neurons into synchrony, thereby increasing the set of synchronised neurons. As in the two-neuron case, synchrony will be maintained for as long as the intrinsic firing rates of the neurons remain sufficiently similar, i.e. until the layer-A output currents to which the neurons are connected begin to diverge again.

Synchrony between weakly-connected integrate-and-fire neurons can be understood by considering them as weakly-coupled oscillators [83]. Synchrony is then equivalent to phase-locking of the oscillators. Analytical studies of this phenomenon typically involve each oscillator having a fixed intrinsic frequency (driving current), being coupled to other oscillators by delta function pulses or sinusoidal functions of relative phase, and with spatially homogeneous connections (all-to-all or nearest neighbour connections) [90, 59, 36, 23]. We are not aware of any theoretical analysis which has been performed for a system such as the *mus silicium* network in which each oscillator has a different and continuously changing intrinsic frequency.

2.4.2 Purpose of inhibitory (β) neurons

The previous section described the mechanism for generation of synchrony between groups of α-neurons in layer W in *mus silicium*. What then is the purpose of the β-neurons also found in layer W?

The post-synaptic potential induced in a layer-W neuron by a connection from an α-neuron is positive. The combined PSPs from many other α-neurons (which, in the unsynchronised state, will be randomly distributed throughout the receiving neuron's firing cycle) will continuously boost the receiving neuron's potential. This will lead to an increase in the firing rate of the receiving neuron that varies in a manner dependent on the average rate of firing of the other W-neurons from which it receives input. This altered firing rate could interfere with the gradual decay in firing rate which is required for the operation of *mus silicium*.

In order that other W-neurons do not influence the overall firing rate, we would like a

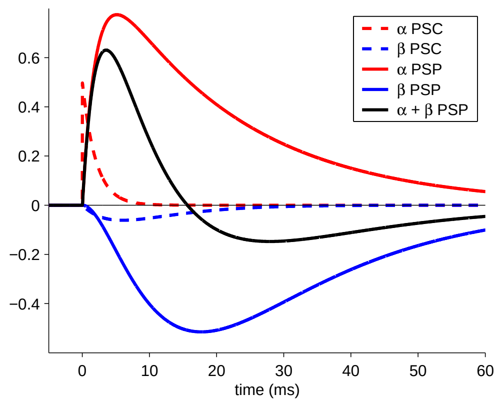
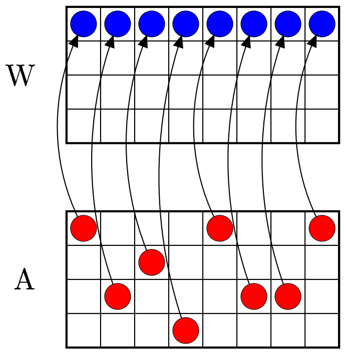


Figure 2.12: Dashed lines show the form of the post-synaptic currents (PSCs) induced by an α-neuron (red) and a β-neuron (blue), in arbitrary units. Solid lines show the corresponding post-synaptic potentials (PSPs), i.e. the post-synaptic currents integrated over time by a leaky integrator with a 20ms time constant. The black line shows the combined post-synaptic potential induced by an α- and β-neuron firing together. The form of the curves is discussed in sections 2.2.4 and 2.4.2.

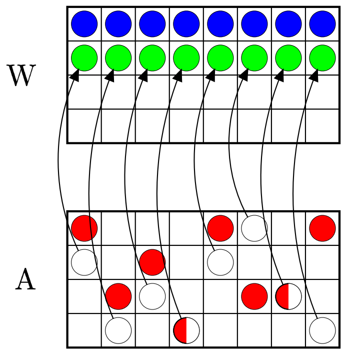
scheme in which the incoming post-synaptic potentials do not cause a net contribution to the membrane potential, but still induce synchrony. This is achieved by allocating each α-neuron a partner β-neuron with the same output connections, set up to fire at the same time (or at least at the same rate) as its partner α-neuron. We can either create a strong connection from the α- to its partner β-neuron (causing the β-neuron to fire immediately after the α-neuron), or simply give the β-neuron the same inputs as the α-neuron (which will ensure that it fires with the same frequency). Because the inhibitory β-PSC is scaled to contain the same net charge as the α-PSC, and because the β-PSC peaks at a later time than the α-PSC (figure 2.12), the result is that the combined α- and β-PSP consists of an excitatory pulse followed by an inhibitory pulse. If the post-synaptic neuron's potential is close to threshold, the excitatory pulse will induce synchrony. If not, the effect of the excitatory pulse on the post-synaptic neuron's potential will be cancelled out by the inhibitory follow-up pulse. The combined pulse, therefore, does not make a net contribution to the potentials of the neurons it is impinging upon, and so does not, in general, affect their firing rate.

2.4.3 Training rule variants

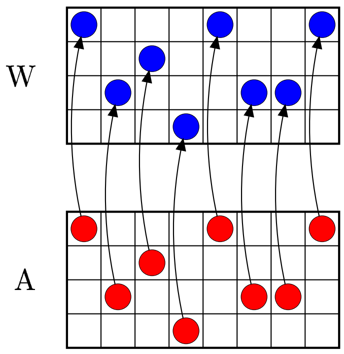
We developed two variations of the training rule. As outlined in section 2.3, if there are 40 features output by the preprocessing stage, then a group of 40 pairs of W-neurons are involved in recognising a particular word. If we wish to learn ten words, then we can accomplish this using ten distinct groups of 40 pairs of W-neurons in a total of 800 W-neurons, which happens to be the number permitted in Hopfield and Brody's competition. In this 'split brain' approach, illustrated in figure 2.13(a,b), the W and Γ layers are divided into ten entirely separate networks of equal size, each recognising one of the training words. The key role of the training algorithm becomes the selection of which layer-A neurons each group of



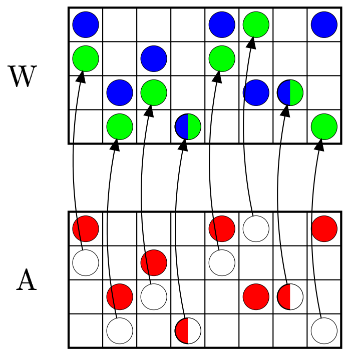
(a) ‘Split brain’ algorithm, 1st word



(b) ‘Split brain’ algorithm, 2nd word added



(c) Distributed algorithm, 1st word



(d) Distributed algorithm, 2nd word added

Figure 2.13: Training rule variants. Layer-A outputs are shown in the ‘A’ grid in each figure. The layout of each ‘A’ grid is as in figure 2.3: each column contains layer-A outputs which are triggered by a particular event; each row has a different decay rate. For clarity, 8 event types and 4 decay rates are shown; the real network has 40 event types and 20 decay rates. Each cell in the ‘W’ grid represents a pair of W-neurons, of which there are 400, not the 32 illustrated, in the real network. **(a,b)** ‘Split brain’ training. **(a)** One pattern (i.e. a selection of one layer-A output from each column) has been learnt. The selected set of layer-A outputs (marked by red circles) are connected to a particular block of W-neuron pairs (blue circles). The blue neurons are all given mutual connections within the W layer (not shown) to induce synchronisation. **(b)** When a second word is learnt, a different, but possibly overlapping, set of layer-A outputs is selected (white circles; layer-A outputs used in both the first and second words are shown half white, half red). Connections are made from these to a *separate* block of W-neurons (green circles), which are once again mutually-connected. Therefore there is no coupling between the W-neurons associated with the first word (blue) and second word (green). **(c,d)** Distributed training algorithm. Each layer-A output, i.e. each cell in the ‘A’ grid, is connected to the W-neuron pair in the corresponding grid location in the ‘W’ grid, regardless of the training data. **(c)** As a result, the set of mutually connected W-neuron pairs associated with the first word are distributed throughout layer W (blue circles). As before, mutual all-to-all connections (not shown) are added between the blue neurons. **(d)** When a second word is learnt using the distributed training algorithm, some of the same W-neuron pairs are used (shown half blue, half green). The mutually-connected set of blue W-neurons and the similarly mutually-connected green set now contain some W-neuron pairs in common. If the overlap between sets becomes too large, the firing of neurons in one group will affect the firing of neurons in the other group (crosstalk between stored memories).

W-neurons receives its connections from. This 'split brain' architecture has the advantage of having zero crosstalk between the ten stored words. However, it is somewhat biologically implausible, imposing as it does a hard limit on the number of words which can be stored given the size of the W layer.

The second variant (figure 2.13(c,d)) connects layer-A outputs to pairs of W-neurons in a one-to-one mapping,[4] and allows a single W-neuron pair to participate in more than one group if more than one training word requires the same feature with the same decay rate. This introduces crosstalk between the groups, which slightly reduces the performance of the network in the ten word case. However, it has the advantage that it allows more than ten words to be stored. Whereas in the 'split brain' training rule storing a new word requires additional W-neurons, in this 'distributed' training rule storing a new word simply adds connections between the existing W-neurons. As more words are added, the ability of the network to perform recognition will degrade as crosstalk between groups of mutually-connected W-neurons becomes stronger. The 'distributed' training rule we have just described is the one which Hopfield and Brody revealed they had been using after the close of their competition [35].

2.5 Simulation results

A simulation of *mus silicium* was built to test and develop the ideas outlined in this chapter. The network was also the winning entry to part B of Hopfield and Brody's competition. This required the construction of a simulated neural network, using limited numbers of neurons of the type they had described in their first paper, which would perform best out of all entries at recognising ten spoken words after being trained on a single example of each.

The simulated network was implemented in the Matlab programming language, using a time-step approach[5] with a temporal resolution of 0.1ms. The results presented in this section use the 'split brain' training algorithm described in section 2.4.3.

Figure 2.14 shows the output of the network in response to five input sounds from the competition test data. In these experiments, two or more spikes from a γ-neuron denote recognition. The γ-neuron trained to respond to the word "one" (γ_1) fires repeatedly in cases (a) and (b) (two instances of "one" said by different speakers). It also responds in case (c), in which the spoken word is obscured by a continuous tone at 800Hz. Given the nature of the pre-processing done by the network, this background tone should not significantly damage recognition: it will disrupt at most two features out of 40 (onset and offset for the frequency band containing 800Hz), thereby removing just two pairs of W-neurons from the synchronising group. The network is robust against the corruption of certain frequency bands in the input.

[4]Since there are 800 layer-A outputs, a one-to-one mapping requires 800 pairs of W-neurons, which is twice as many as permitted in Hopfield and Brody's competition. However, if only ten words are learnt, at least 400 of these will end up not being used (i.e. will have no outgoing connections). The unused W-neuron pairs can be pruned, resulting in a total of no more than 400 pairs.

[5]A comparison of time-step and event-driven simulation techniques is made in section 3.3.1.

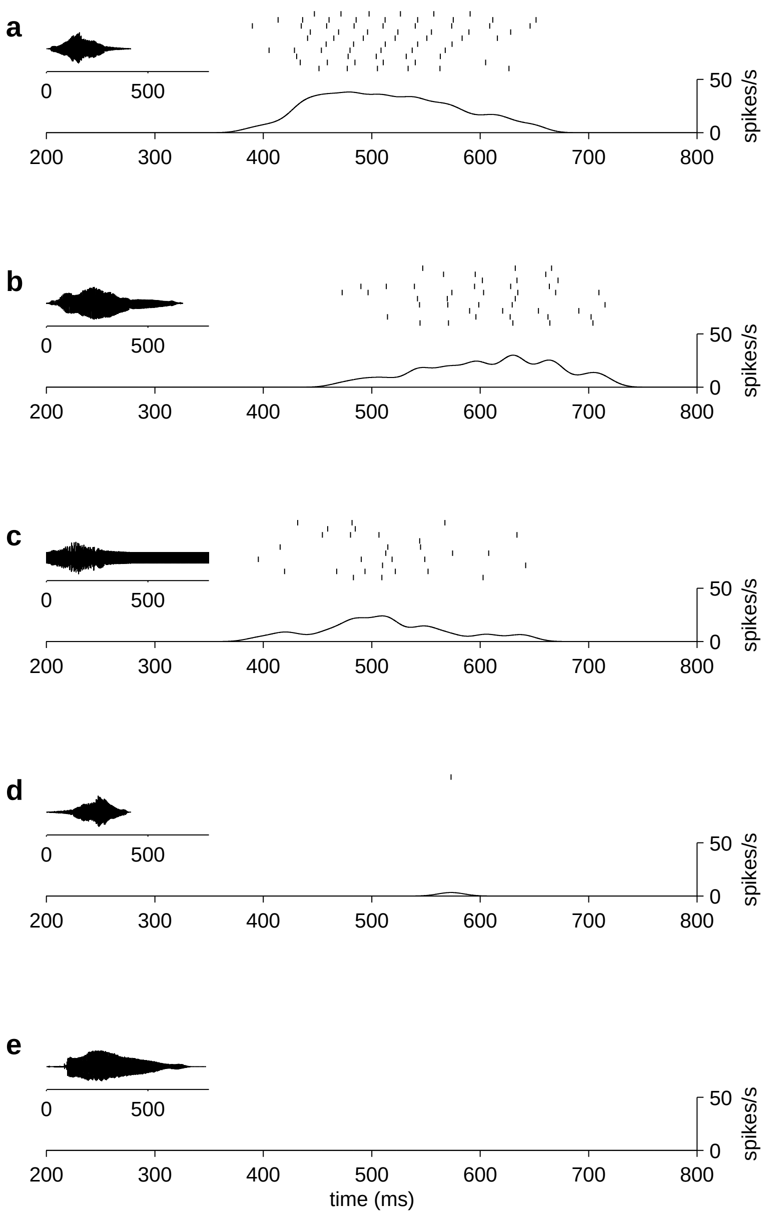


Figure 2.14: Example simulation results. Each panel shows the waveform of the input sound (inset top left), the firing times of the γ-neuron corresponding to recognition of the word "one" for ten separate trials (each trial on a separate line), and the smoothed spike histogram (main axis). In each case, the network was trained on a single example of "one" and nine other words whose gamma neurons are not shown. The input to the network in each case was **(a)** "one" said by speaker A (not the training speaker); **(b)** "one" said by speaker B (not the training speaker); **(c)** as for (a) but with a continuous tone superimposed; **(d)** as for (a) but reversed in time; **(e)** "three". The figure is formated similarly to figure 1 in [34] for ease of comparison.

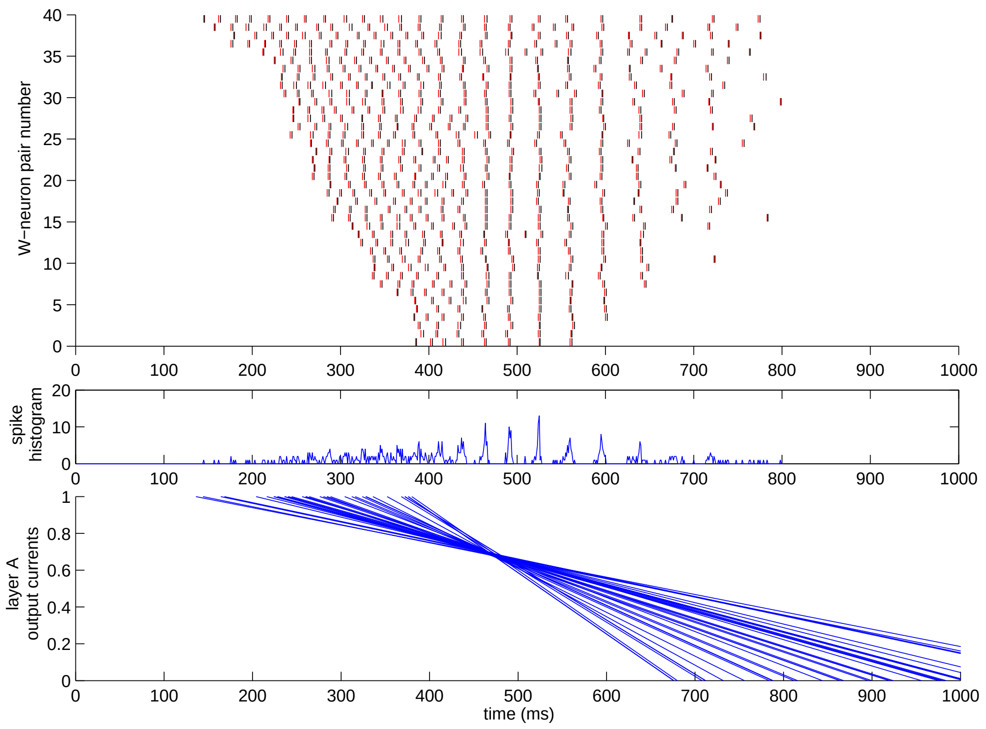


Figure 2.15: The network's behaviour when presented with the single word used to train the network ("one"). **Bottom axis:** output over time of the 40 layer-A outputs selected by the training algorithm to have outputs that intersect after "one" is heard. **Top axis:** firing times of the set of 80 W-neurons responsible for recognising "one". Each row in the raster plot shows a pair of W-neurons (an α- and β-neuron, shown in red and black respectively). Because each β-neuron fires just after its partner α-neuron, we see double spikes on the raster plot. **Middle axis:** unsmoothed spike histogram, indicating the degree of synchrony among the W-neurons. Notice that, at around 500ms, the A-neuron outputs intersect and the W-neurons fire in near synchrony as a result.

The γ_1 neuron does not respond to a time-reversed "one" (d), demonstrating that the network is not merely detecting the presence of the relevant features (frequencies), but is sensitive to their temporal relationship. As a final example, the γ_1 neuron does not respond to the word "three".

Figures 2.15, 2.16 and 2.17 show the internal functioning of the network when presented with the "one" it was trained on, a different unseen instance of "one" and "nine", respectively. Each figure shows the spike rasters from the set of W-neurons involved in recognising "one" (see figure captions for details).

To quantify the performance of the network we used a test set of 500 clean recordings from the NIST database, as provided by Hopfield and Brody to compare entries to their competition. The test set contained the 10 digits "zero" to "nine", each spoken 10 times by five different speakers. After being trained to recognise "one" (from a single example) plus nine other input patterns, we used the number of times that the γ-neuron corresponding to "one" fired to classify each input sound as "one" or non-"one".

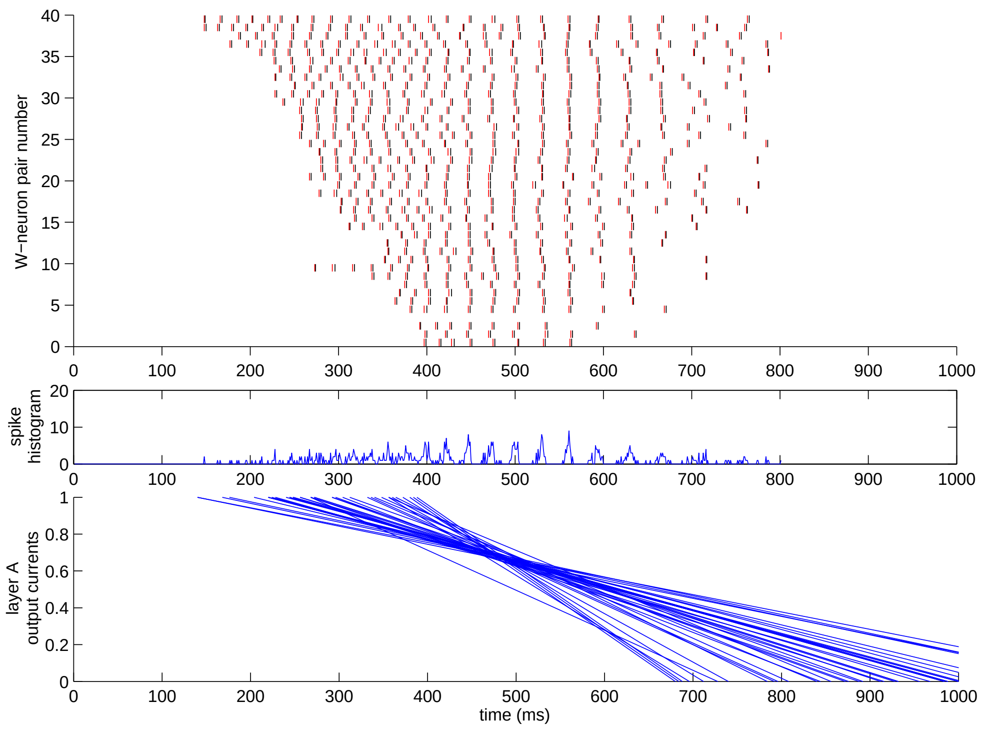


Figure 2.16: As figure 2.15, but with the input to the network being a different example of the word "one" to the training example. Note that in the bottom axes the decay lines no longer cross exactly, but most still pass through a small region. As a result there is still significant synchronisation among most of the W-neurons in the set.

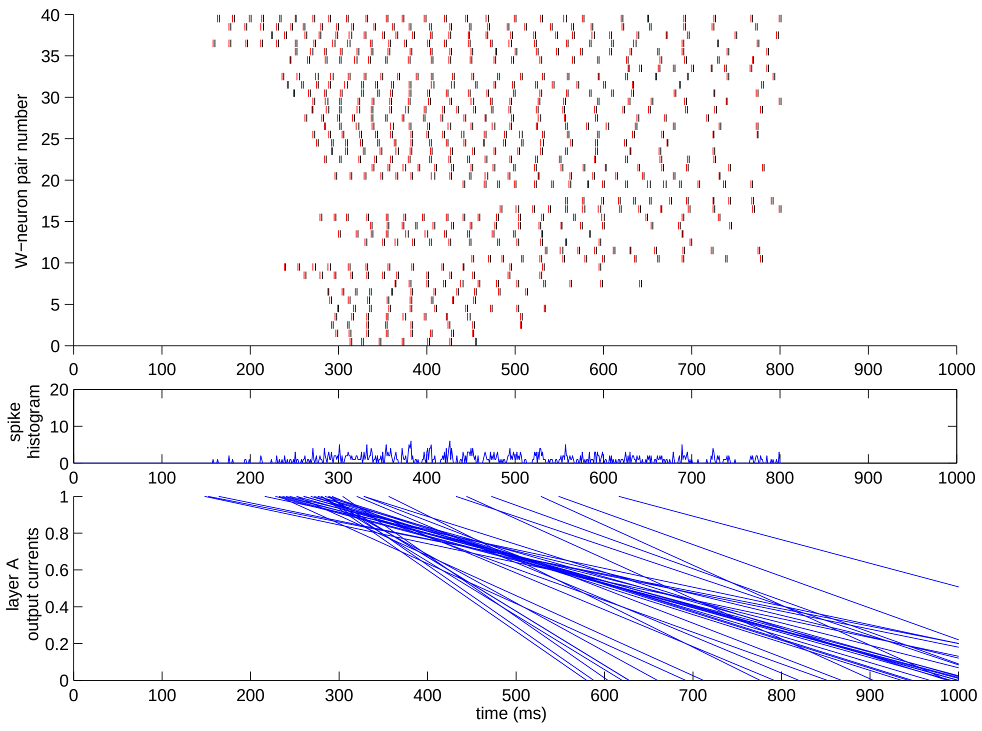


Figure 2.17: As figure 2.15, but with the word "nine" as the input to the network (trained on "one"). There is no significant synchronisation.

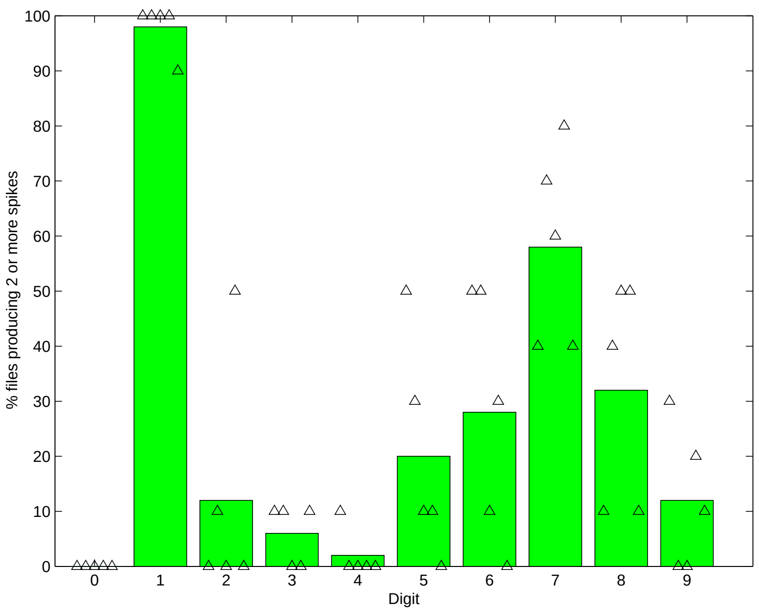


Figure 2.18: Performance of the network on the test data. Each bar shows the percentage of recordings of each spoken digit which caused the γ-neuron responsible for detecting instances of "one" to fire two or more times (deemed to be recognition). Ideal results would correspond to 100% for the '1' bar and 0% for all others. The triangles denote averages for each of five different speakers.

Figure 2.18 shows a breakdown of how the network (using the 'split brain' training rule) classifies the test data as "one" and non-"one". 98% of the instances of "one" were correctly identified. The false-positive rate for the other spoken digits varied from 58% for "seven" to zero for "zero".

2.6 Conclusion

We have demonstrated that an artificial neural network built from less than 1000 spiking neurons can perform speech recognition with a small vocabulary. The recognition is invariant to linear time-warp and is robust across speakers. The system is not intended to be competitive against alternative computer speech recognition algorithms, but it demonstrates that robust time-warp-invariant speech recognition can be done with neural hardware similar to that found in the brain.

2.7 Relationship to other work

As has been explained in the text, part of the research described in this chapter was conducted while working on a solution to an academic competition organised by John Hopfield and Carlos Brody. It is therefore similar to work done by Hopfield and Brody themselves, but was done independently.

Chapter 3

Temporal Memory with Spiking Neurons

3.1 Introduction

The aim of this chapter is to develop an artificial neural network model capable of storing and reproducing memories in the form of spatiotemporal spike patterns (spikes distributed over both time and a population of neurons). The first section introduces the mechanism used by the model and gives an overview of its capabilities. The remainder of the chapter defines the model more precisely in preparation for a detailed investigation of its behaviour in chapter 4.

3.1.1 Autoassociative memory

A conventional memory, such as that typically found in a computer, can be thought of as an array of boxes, each labelled with an *address*, and each capable of storing one item of data. Storage involves putting an item of data into a specified box, and recall involves asking for the contents of the box with a particular address.

An alternative paradigm for memory is known as *content-addressable* memory. A content-addressable memory does not require an address in order to retrieve data, but can retrieve an item of data in response to another item of data. For example, an *autoassociative* memory can, in response to a partial or noisy version of one of its stored memories, retrieve the complete, corrected memory. More generally, an *associative* memory is one that stores associations between pairs of data elements. Presenting an associative memory with one member of a stored pair should cause the recall of its partner data element. A simple example would be a memory in which the associations between people's names and their favourite colours are stored.

In general, an autoassociative memory can be configured as an associative memory simply by considering each stored data element in the autoassociative memory to consist of two parts. If we present the memory with a version of a stored memory which has one part missing, then from the point of view of the autoassociative memory we are presenting a noisy version of the memory and it will attempt to fill in the missing half. But if we think of the two parts as a pair of associated data elements, then filling in the missing part is in fact retrieving the data element associated with the first part. In other words, autoassociation between two halves of a data element can equally be thought of as association between two distinct data elements. The distinction, then, between autoassociative and associative memories is quite blurred, and the term associative memory is frequently used to cover both types.

The most well-known example of an (auto)associative memory is the Hopfield network [32]. A binary Hopfield network consists of neurons which have an activity (output), x_i, which takes a value of ± 1 at any moment in time; i is an index over the N neurons in the network. A memory is a vector containing the activities of all the neurons in the network, $\mathbf{x}$. Neurons are linked by symmetric, bidirectional connections with real-valued weights, $w_{ij} = w_{ji}$ denoting the weight for the connection between neurons i and j. The weights are set during the learning of the stored memories, $\{\mathbf{x}^{(n)}\}$, which takes place according to the Hebb rule [26],

$$w_{ij} = \eta \sum_n x_i^{(n)} x_j^{(n)}, \tag{3.1}$$

where n is an index over memories to be stored. The strength of the connection between two neurons is therefore a measure of the degree of correlation between those two neurons' states in the set of training memories.

Each neuron sums its inputs:

$$a_i = \sum_j w_{ij} x_j \tag{3.2}$$

and uses a threshold activation function to compute its output:

$$x(a) = \Theta(a) \equiv \begin{cases} 1 & a \geq 0 \\ -1 & a < 0. \end{cases} \tag{3.3}$$

Recall is begun by setting the activity of the neurons to a corrupted version of one of the memories and then repeatedly updating the neuron activities. The precise procedure may be varied by choosing the order of updates and whether they are performed in sequence or in parallel. Ideally, the state of the network will stabilise at the "nearest" stored memory to the corrupted stimulus state. This can be viewed as descending an energy landscape which has been shaped by all the stored memories, towards a local minimum, or *attractor* state. When the network is not overloaded with too many memories, these attractor states correspond exactly to the stored memories.

3.1.2 Spatiotemporal spike sequences

Neurons in the brain communicate using spikes. A binary Hopfield network, on the other hand, transmits the values $\{-1, 1\}$ over connections with real weights. Furthermore, although the Hopfield network, as originally stated, uses a sequence of updates to reach a stable state, the final state is static, unlike a network of spiking neurons which can only output information in the form of a dynamic pattern of spikes.

Many attempts have been made to implement Hopfield networks and similar associative memory models using networks of spiking neurons. Some use a rate code in which a high frequency of firing denotes the 1 state and a low frequency of firing (or not firing at all) denotes the -1 state [24]. Other models utilise temporal coding, typically involving a fixed period oscillation with each neuron firing at a particular time within the cycle to denote the 1 state and at a different time within the cycle (or not at all) to denote the 0 state [52, 60, 24]. It has also been suggested that analogue memories could be encoded in the form of spike timing relative to a sub-threshold oscillation in the membrane potential of each neuron [33, 60].

In this work, we do not seek a direct mapping of the Hopfield network onto spiking neuron hardware. Instead, we extend the domain in which the memory operates. Whereas in the Hopfield network each memory is a binary vector $\{-1, 1\}^N$, one memory in our model is a pattern of spikes distributed over both a population of neurons ("space") and time, otherwise known as a spatiotemporal spike sequence. Thus, the aim of our memory is to store a number of different spatiotemporal spike sequences as an autoassociative memory, so that any stored memory can be recalled by the network when presented with a partial or corrupted version of that memory.

In this work, we take the spatiotemporal spike sequence as the fundamental 'data type' that we are trying to store and, in common with most analyses of associative memories, use randomly generated patterns for the purposes of measuring the performance of the network. In a biological setting, it is more plausible that the spike sequences would encode sensory stimuli, desired motor system commands, or the results of intermediate computations in the brain. Indeed, evidence is beginning to emerge that precise, repeating spatiotemporal spike sequences are present in brain activity (see for example [62, 86, 66, 4, 3], but see also [12] which disputes the evidence).

It is not difficult to map a variety of other data types onto spatiotemporal spike sequences if the memory is required to operate on other types of data. For example, a vector of analogue values can be mapped onto a periodic spatiotemporal spike sequence by using one neuron per element of the vector and converting the value of that element into the firing time of the neuron relative to a global background oscillation ([33] and [42] describe neural mechanisms by which analogue or rate-coded inputs can be converted into spike timing). A related technique is called upon in chapter 5 when we require a mapping of a single real number onto a spatiotemporal spike sequence.

In addition to their biologically plausible nature, spatiotemporal spike sequences have another advantage over conventional representations: they allow simultaneous representation of multiple patterns. A neural network such as the Hopfield network represents its present state by the output of its neurons, which take discrete (or, in some networks, real) values. A spatiotemporal spike sequence representation, on the other hand, encodes a state by a sequence of spikes over a finite time period. Because neurons are able to fire more than

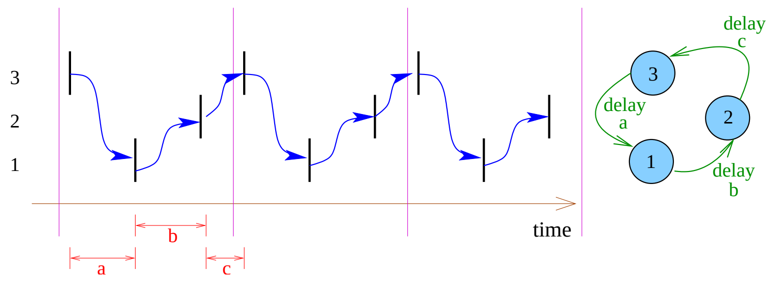


Figure 3.1: Storing a simple spike sequence in a network of spiking neurons and delay lines. The figure shows, on the left, a spike raster plot showing the firing times of three neurons, making up a periodic pattern consisting of three spikes, repeated three times (vertical red lines mark possible period boundaries for this sequence). By making connections between the neurons with delays a, b and c as shown on the right hand side, the network is configured to generate the periodic pattern indefinitely once triggered by the stimulation of any participating neuron.

once in such a period, a superposition of two states (two spike patterns) produces another spike pattern. In other words, we can superpose memories without leaving the domain of spatiotemporal spike sequences. This ability of the network to effectively be in a superposition of states is put to good use in the model described in this chapter since, as will be explained in section 3.1.5, the network is able to recall and clean up multiple stored memories concurrently. One possible application of this ability is short term memory, which could be implemented by a single population of neurons whose activity is the superposition of the thoughts being held 'in mind'. More generally, the ability to combine mental states into superpositions provides a general-purpose mechanism for building up compound concepts. A simplistic example of this would be the idea that the neural representation of a tree could be built up from a superposition of the representations for 'leaves', 'green', 'trunk', and so on. This also relates to the *binding problem* [68, 87]: the question of how the brain combines multiple aspects of sensory input in such a way that they are identified as jointly explaining a single object in the external world.

3.1.3 Spike sequence storage

We begin with a simple network constructed from neurons that fire when they receive an incoming spike. The neurons are connected by delay lines which transmit spikes with a fixed delay. With a suitable choice of connections and corresponding delays, such a network can, through a simple chain reaction, generate a simple spike sequence in which each neuron fires at most once (figure 3.1). Optionally, the chain of delays can be closed to create a periodic cycle generator which will repeatedly 'play out' the desired spike pattern, as is the case in figure 3.1.

This simple idea cannot be trivially extended to spike patterns in which a given neuron fires more than once, nor to a network capable of storing more than one such memory if there is any overlap between the neurons participating in the different patterns (figure 3.2). In this

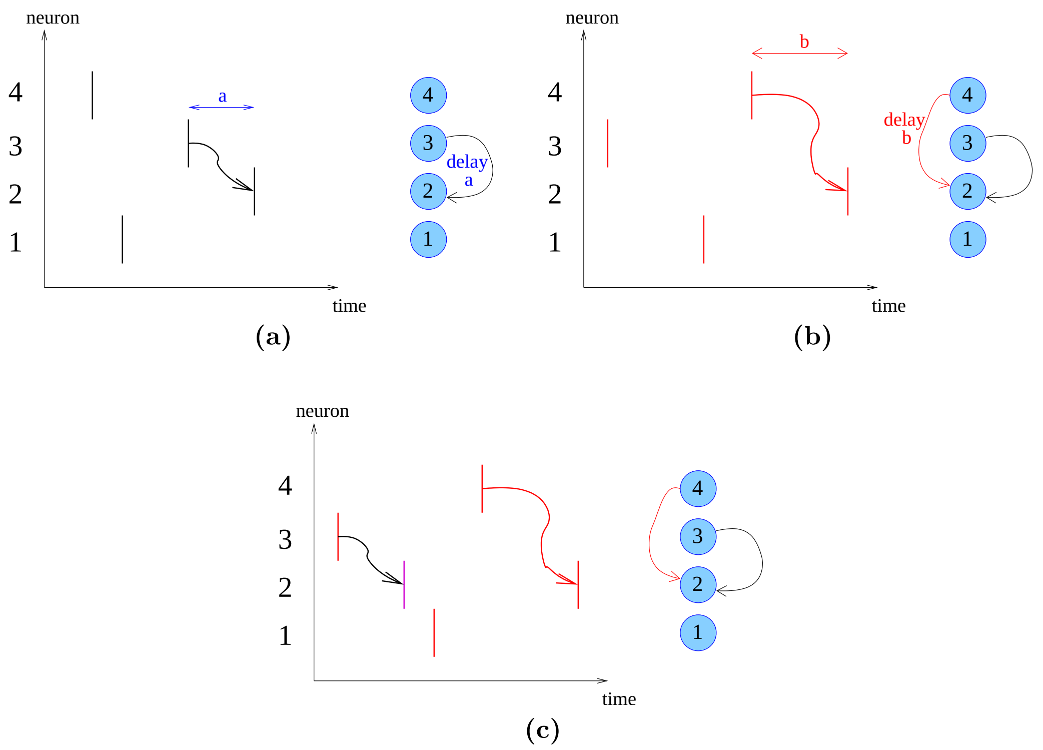


Figure 3.2: Naïve attempt to store multiple patterns using simple spiking neurons and delay lines. For clarity, short non-periodic patterns are used and only connections to neuron 2 are shown. **(a)** The first pattern to be learnt involves a connection with delay *a* from neuron 3 to neuron 2. **(b)** The second pattern to be learnt, involving a connection from neuron 4 to neuron 2, is added. **(c)** Recall of the second pattern now induces spurious spikes since neuron 2 is triggered in the incorrect context.

very simple scheme, the 'context' each neuron uses to determine when to fire is a single spike from another neuron. Such a simple context is certain to occur in many other spike patterns (or elsewhere in the same pattern), resulting in the target neuron firing at incorrect times.

3.1.4 Coincidence detectors

One way to get around this problem of simple context is illustrated in figure 3.3. Here we have made two changes. Firstly, we trigger each spike that neuron 2 is required to produce not from a single preceding spike in the relevant pattern, but instead from a *conjunction* of several preceding spikes, each delayed by an appropriate amount such that they arrive at neuron 2 simultaneously at the moment that it is required to fire. Secondly, we combine these incoming signals using a 'conjunction detector'[1] which only responds to simultaneous spikes on all its inputs. The neuron has as many conjunction detectors as contexts it is required to recognise. Each conjunction detector, together with the appropriate delay lines which connect to it, detects a particular configuration of preceding spikes that indicates that the neuron is required to fire. By enlarging the context each neuron uses to identify when to fire from a

[1]We use the terms 'conjunction detector' and 'coincidence detector' interchangeably in this thesis.

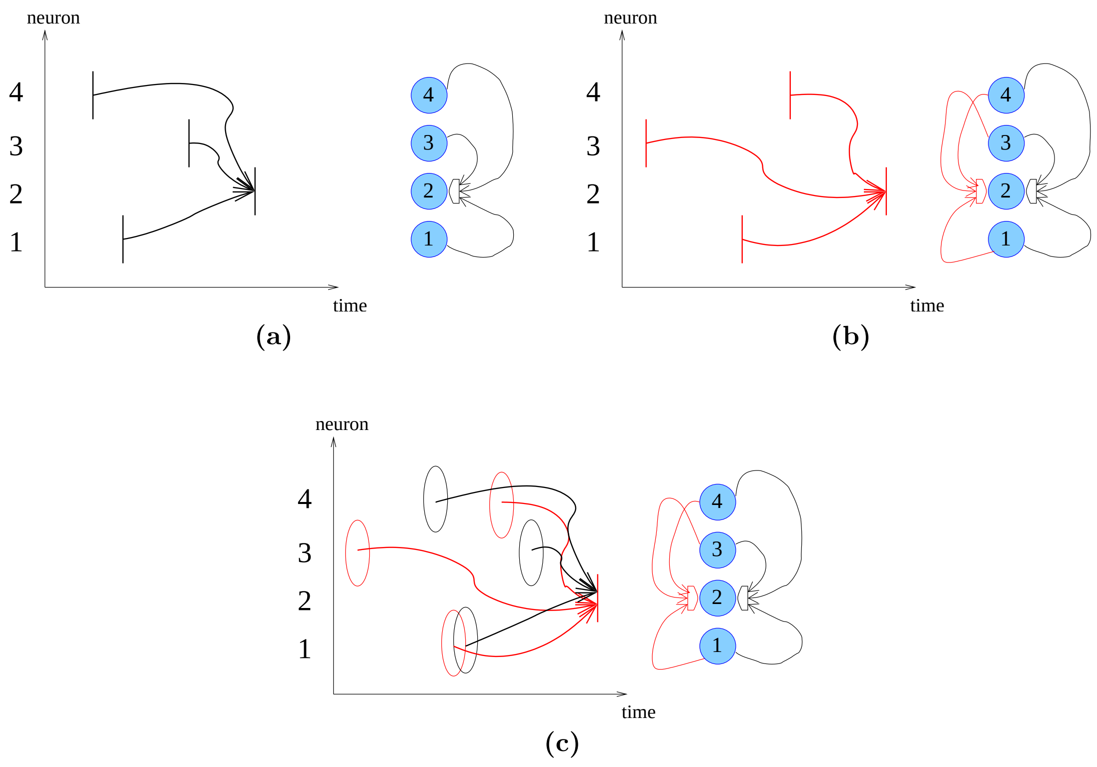


Figure 3.3: Addition of coincidence detectors allows storage of multiple memories, solving the problem highlighted in figure 3.2. **(a)** The required spike from neuron 2 is now triggered by a conjunction of preceding spikes from neurons 1, 2 and 4. **(b)** A second stored memory uses a different conjunction detector to store neuron 2's firing context in that memory. **(c)** Neuron 2 is now 'watching out for' two different temporal patterns of preceding spikes. Because each triggering pattern involves several spikes in a particular temporal relationship, it is unlikely that a conjunction detector will be triggered by the wrong memory.

single spike to several spikes we dramatically reduce the probability of the triggering context occurring 'by accident' in other stored memories.

Coincidence detectors have a long history of use in spiking neural network models. An integrate-and-fire neuron (section 2.2.3) with a time constant that is long compared to the typical inter-spike interval will behave predominantly as an integrator of incoming spikes. With a short time constant and suitable firing threshold, an integrate-and-fire neuron behaves predominantly as a coincidence detector, only firing when several spikes arrive within a short time interval. Indeed, we have already used this mode of operation for the γ-neurons in chapter 2, which were integrate-and-fire neurons configured to fire only when they receive a large number of synchronous incoming spikes. This dual role for integrate-and-fire neurons has led to a debate over whether the primary role of neurons in the real brain is as an integrator or a coincidence detector [1, 48].

In this model, however, we are not simply treating the neuron as a single coincidence detector. Instead, we are endowing each neuron with *several* coincidence detectors, each on the look-out for one of the several contexts the neuron is required to fire in response to. The

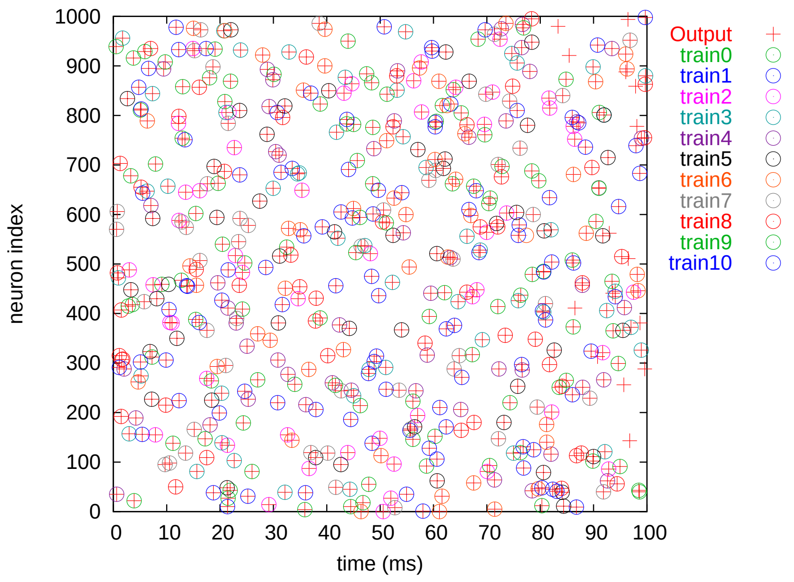


Figure 3.4: Demonstration of concurrent recall. The network defined in section 3.2, with 1000 neurons ($S = 4$, $s = 3$, $d = 1$) was trained on 100 memories each consisting of 50 spikes spread over 100ms. The model was stimulated with the first 5 spikes of 11 of the stored memories, and successfully generated the remainder of all 11 patterns concurrently. Crosses denote the actual spiking output of the network; circles denote the position of spikes in the 11 selected training patterns. Each colour denotes a different training pattern.

task performed by each coincidence detector is that performed by a 'grandmother neuron'[2] in some models involving time delays and coincidence detectors [84, 33]: detecting a particular temporal sequence of spikes or input events. Since each of our neurons contains more than one coincidence detector, it cannot map directly onto a straightforward integrate-and-fire neuron model. Some further discussion of possible neural implementations is contained in sections 3.2.2 and 3.2.9.

3.1.5 Concurrent recall

In the model we have just introduced, each conjunction detector allows a neuron to detect some context (a group of a few spikes) from one learnt memory and causes the neuron to contribute its own spike at the correct time for that memory. Given that arrangement, there is no reason why only one memory can be played out by the network at a time. Suppose that a particular neuron has one spike during each of two memories $\mathcal{M}_1$ and $\mathcal{M}_2$. Since the network has been trained on these memories (and perhaps others), the neuron will have one conjunction detector configured to detect a particular temporal pattern of spikes which indicates the moment within $\mathcal{M}_1$ at which the neuron should fire, and another conjunction

[2] A 'grandmother neuron' has the sole purpose of recognising one particular complex stimulus. The name comes from a caricature of the idea that there is a neuron responsible for recognising each possible object that a person interacts with. According to this naïve approach, somewhere in a person's brain would be a neuron which responds only when they see their grandmother.

detector doing the same thing for the neuron's contribution to $\mathcal{M}_2$.

Now suppose that the actual spiking activity of the network is the superposition of $\mathcal{M}_1$ and $\mathcal{M}_2$. One conjunction detector will detect a particular temporal spike pattern made up from spikes in $\mathcal{M}_1$, and the other will detect a temporal spike pattern made from spikes in $\mathcal{M}_2$. Unless the neuron happens to be required to fire at exactly the same time in $\mathcal{M}_1$ and $\mathcal{M}_2$, the neuron will be triggered to fire twice within the recall period: once to participate in $\mathcal{M}_1$ and a second time to participate in $\mathcal{M}_2$. Both memories will therefore continue to be generated by the network in superposition, and will not interact with each other unless a conjunction detector intended to detect part of one memory happens to be also triggered by part of the other memory.

The ability of a network of this kind to generate (recall) more than one memory concurrently (demonstrated in simulation in figure 3.4) is one of the features which distinguishes it from almost all other memory models.

3.2 The model

Having introduced the ideas behind our spiking memory model, we use the remainder of this chapter to define precisely the model, which we call the *Concurrent Recall Network*.

3.2.1 Network architecture

The model consists of a network of N neurons. Spikes are emitted by neurons and propagate, with some finite delay, along connections between neurons. Each connection terminates at a synapse with another neuron. When a spike reaches a synapse, the synapse becomes *active* for a fixed time t_{psp}. This models the duration of the post-synaptic potential.

Because t_{psp} is the same for all synapses, an equivalent view of spike transmission between neurons is to consider the spikes themselves to take the form not of delta functions but of binary top-hat functions with width t_{psp}. A synapse is then active for the duration of the impinging spike. For simplicity of explanation we favour this latter point of view in what follows.

3.2.2 Neurons

A neuron in this model is an information processing unit which performs a simple computation dependent on incoming spikes, and may, depending on the result of its computation, emit spikes on its output. Many varieties of spiking neuron models are used in the literature; we have already met integrate-and-fire neurons in chapter 2. Here we use a different neuron model: instead of maintaining a membrane potential which gradually accumulates input, we define the activity of the neuron as a memoryless function of its current inputs, which are binary values (1 during an incoming spike and 0 otherwise).

In particular, we model the input side of a neuron as a number of *conjunction detectors.* Each conjunction detector is endowed with S synapses (inputs), each of which receives input from exactly one neuron. A conjunction detector is said to be *active* when at least some number s of its S synapses are in the process of receiving a spike (thus, it detects the conjunction, or coincidence, of s spikes). Formally, we refer to this as an "s-of-S" operator, where we define an "s-of-S" operator, with inputs $x_i \in \{0,1\}$ ($i \in \{1 \dots S\}$), as

$$s\text{-of-}S(\{x_i\}) \equiv \begin{cases} 1 & \text{if } \sum_i x_i \geq s, \\ 0 & \text{otherwise.} \end{cases} \tag{3.4}$$

The activations of the neuron's D conjunction detectors are summed by the neuron, to give the combined input to the neuron:

$$a = \sum_{j=1}^{D} s\text{-of-}S(\{x_i\}^{(j)}) \tag{3.5}$$

where $\{x_i\}^{(j)}$ are the S synapses on the jth conjunction detector.

In the special case $s = S$, the s-of-S operation reduces to an S-input AND operator, which can also be thought of as a product in $\{0,1\}$. In this special case, the net input to the neuron takes the form of a binary sigma-pi unit [69]:

$$a = \sum_j \prod_i x_i^{(j)} \tag{3.6}$$

where $x_i^{(j)}$ is the ith synapse on the jth conjunction detector. In the case $s \approx S$ we can think of our neuron's input stage as *approximating* a sigma-pi unit.

The output side of a neuron is governed by a process which generates a spike whenever a (the combined input to the neuron) rises above a threshold d, unless a time shorter than t_{refr} has elapsed since the neuron previously emitted a spike in which case spike generation is suppressed. In other words, unless the neuron is in its post-spike *refractory period*, a spike is generated whenever d or more of its D conjunction detectors become active. In this research we focus primarily on the case $d = 1$. Output spikes generated by the neuron propagate along all outgoing connections from the neuron (each of which has an associated delay), eventually arriving at a synapse on a conjunction detector on another neuron.

For ease of visualisation, we sometimes describe the neuron in physiological terms, with each conjunction detector thought of as one branch in a *dendritic tree*, and the *soma* (cell body) combining the activations of each dendritic branch. *Axons* (outgoing connections) link the neuron to its counterparts. This view of the neuron is illustrated in figure 3.5. The use of this terminology is not intended to imply that the neuron model outlined in this section maps directly onto real neural hardware. For further discussion of the biological plausibility of this neuron model, see section 3.2.9.

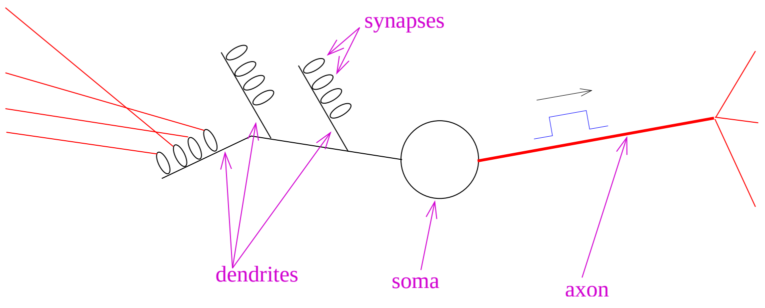


Figure 3.5: Illustration of a neuron in the Concurrent Recall Network using biological terminology. Each neuron has incoming connections grouped into dendritic branches, which, in this model, behave as coincidence detectors. The outputs of all dendritic branches are summed at the soma (cell body). If a threshold is reached, an outgoing spike is transmitted along axons (which impose time delays) before impinging upon other neurons.

3.2.3 Noise

Three types of noise may be incorporated into the model dynamics.

Firing reliability

p_{fire} is the probability that a neuron that is supposed to fire will actually do so. $p_{\text{fire}} = 1$ (the value used during simulations unless otherwise stated) corresponds to completely reliable firing.

Spontaneous firing

r_{spurious} is the rate parameter for a Poisson process governing spontaneous firing of each neuron, independently of the neuron's inputs. It is set to zero during simulations unless otherwise stated.

Temporal jitter

A connection between two neurons, i and j, has an associated time delay $\Delta_{i,j}$, set up by the training algorithm (section 3.2.5). The minimum value of $\Delta_{i,j}$ created by the training algorithm is Δ_{min}. Every time a spike travels along a connection between neurons, we perturb the time delay with Gaussian noise such that the actual delay experienced is a sample from a Normal distribution with mean $\Delta_{i,j}$ and standard deviation σ_{jitter}. By ensuring that σ_{jitter} is several times smaller than Δ_{min}, the probability that this could generate a negative delay can be made small. If it does happen, a delay of zero is used instead. σ_{jitter} is set to zero during simulations unless otherwise stated.

3.2.4 Definition of a memory

A memory $\mathcal{M}$ is defined as a set of m spikes $\{(n_i, t_i)\}$ ($i \in \{1 \ldots m\}$), each spike defined by the index n_i of the neuron that generates it and the time t_i at which it happens. In cases where memories are intended to repeat periodically, each memory also has a period T associated with it, with T larger than the time of the last spike in $\mathcal{M}$.

3.2.5 Training

The training rule examines the set of memories to be stored in the network and sets up time-delay connections between neurons' outputs and the inputs of other neurons' conjunction detectors. The aim is that after training the network will be able to replay the stored memories.

Two related training algorithms have been explored. We present here the first, which we call the *unlimited training rule* because it imposes no limit on the number of conjunction detectors which a neuron can possess. A second algorithm is explored in section 4.3.

Under the unlimited training rule, a memory $\mathcal{M}$ is added to the network by the following algorithm.

Algorithm 1 Unlimited training algorithm

```
for all spikes $(n_i, t_i) \in \mathcal{M}$ do
  Establish a new conjunction detector, $\gamma$, with $S$ inputs, on neuron $n_i$
  Let $\mathcal{P} = \mathrm{Q}(\mathcal{M}, n_i, t_i)$, i.e. $S$ spikes forming a subset of $\mathcal{M}$.
  for all spikes $(n_j, t_j) \in \mathcal{P}$ do
    Create a connection from neuron $n_j$ to an unused input on the new conjunction detector, $\gamma$, with propagation delay $t_i - t_j \mod T$.
  end for
end for
```

$\mathrm{Q}(\mathcal{M}, n_i, t_i)$ is a function that takes a memory $\mathcal{M}$ and the identity of a spike in that memory (specified by n_i and t_i), and returns a subset of $\mathcal{M}$ of size S. The spikes in this subset are the ones which will be used as the 'context' spikes to trigger a conjunction detector on neuron n_i in order to cause it to fire at time t_i. Several definitions of $\mathrm{Q}(\cdot)$ are used in this thesis, including:

$\mathrm{Q}_{\mathrm{nearest}}(\mathcal{M}, n_i, t_i)$ Returns the S spikes in $\mathcal{M}$ immediately preceding $t_i - \Delta_{\mathrm{min}}$. If periodic training is desired, "preceding" is interpreted with wraparound in the time domain, i.e. the last spike in the memory is treated as being the one "preceding" the first spike. Δ_{min} serves the purpose of enforcing a minimum propagation delay for connections.

$\mathrm{Q}_{\mathrm{uniform}}(\mathcal{M}, n_i, t_i)$ If periodic training is desired, this returns S spikes picked at random from a uniform distribution over all spikes in $\mathcal{M}$ other than those lying between $t_i - \Delta_{\mathrm{min}}$ and t_i. If the pattern is not to be treated as periodic, then only spikes prior to $t_i - \Delta_{\mathrm{min}}$ are available to select from.

If Q(·) is non-deterministic, for example when using $Q_{\text{uniform}}(\cdot)$, memories can be made more robust by repeating the training algorithm more than once. Each spike in a stored memory then has multiple contexts that can trigger it. In the experiments in this and the next chapter, the algorithm is applied only once unless otherwise stated.

More than one memory can be stored by running the training algorithm once (or more) for each memory. With unlimited training rule, the total number of conjunction detectors (and hence connections) in the network increases linearly with the number of stored patterns (assuming all patterns contain the same number of spikes). The *limited training rule*, described in the next chapter, does not have this property.

3.2.6 Recall

The network is said to be *recalling* a memory if the pattern of spikes being generated includes (to within a certain temporal tolerance) the desired memory as a subset. The degree of recall is not a binary measure, but rather we can assess the degree of recall through factors such as the number of spikes in the target memory that are present. If the network is trained on a periodic memory, then the pattern of activity induced in the network should repeat indefinitely. For non-periodic memories, the network should become quiescent upon reaching the end of the memory or memories being recalled.

Recall is triggered by externally stimulating neurons, causing them to fire at particular times. When the network has been trained using the training algorithm described in section 3.2.5 using $Q_{\text{nearest}}(\cdot)$ to select candidates for connections, then any s consecutive spikes in one of the learnt memories (where s is the number of simultaneously active synapses required to activate a conjunction detector — see section 3.2.2) will, in the absence of noise, be sufficient to trigger a spike. In general, this spike, in conjunction with $s-1$ of the artificially stimulated ones, will then trigger a subsequent spike, and so on, causing the entire remainder of the memory to be recalled in the manner outlined in the first part of this chapter. If the training was periodic, the entire pattern will then loop indefinitely. In practice, we stimulate the network with more than s spikes since it is possible that the minimum axon delay, Δ_{min}, can prevent certain memories from bootstrapping from s consecutive spikes alone.[3]

For other variants of the training algorithm, different subsets of spikes from training memories will be required to bootstrap the recall of a target spike pattern. For example, if $Q_{\text{uniform}}(\cdot)$ is used to select the 'context' spikes for each neuron, then stimulating s consecutive spikes from a stored memory is unlikely to result in any conjunction detectors being activated. Instead, recall can be triggered by stimulating some proportion of all spikes in the desired memory, spread throughout its duration.

Triggering concurrent recall of multiple memories is simply a case of stimulating the

[3]To see this, consider the case when $s = S$ and the interval between the sth and $(s+1)$th spike in a stored memory is longer than Δ_{min}. The spike triggered by the first s spikes will occur after the $(s+1)$th spike. The $(s+1)$th spike will remain untriggered, and so the conjunction detectors which rely on it as one of their inputs will not get activated, preventing the memory from bootstrapping.

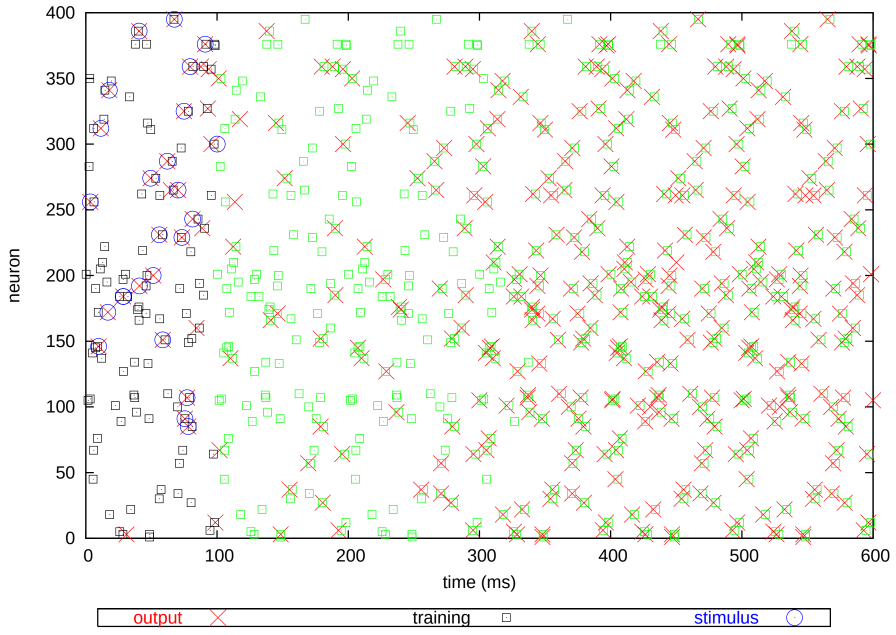


Figure 3.6: Pattern completion and robustness to temporal jitter. A network of 400 neurons was trained on a single periodic memory of 100 spikes (black squares) with period 100ms, using the algorithm of section 3.2.5 applied four times. Each conjunction detector had 3 inputs and activated if 2 or more of them received a simultaneous spike ($S = 3$, $s = 2$). The post-synaptic potential duration, t_{psp}, was 2ms. The 'context' spikes which trigger each conjunction detector were selected randomly from the training memory ($\text{Q}(\cdot) = \text{Q}_{\text{uniform}}(\cdot)$). The network was stimulated with 25% of the spikes in the stored memory, each temporally jittered according to a Normal distribution with zero mean and standard deviation 4ms (blue circles). The spiking activity of the network is shown by red crosses. Green squares show a periodic repetition of the training memory for comparison. In each subsequent 100ms period, the network reproduces a greater proportion of the spikes in the memory, gradually filling in the missing 75% of spikes in the original stimulus. The network also cleans up the temporal jitter in the stimulus spikes. Because spikes which arrive early, rather than late, will be the ones that get to trigger a spike, the recalled pattern stabilises with a slight time advance compared to a precise periodic repetition of the training memory.

network with some (or all) of the spikes from all memories which it is desired to recall. For example, if $\text{Q}_{\text{nearest}}(\cdot)$ was used during training, the network could be stimulated with a superposition of the first few spikes from each of the memories to be concurrently recalled.

3.2.7 Pattern completion and noise resilience

Three aspects of the Concurrent Recall Network provide a degree of robustness against missing, or temporally jittered, spikes. Firstly, the finite spike duration, t_{psp}, means that incoming spikes to a conjunction detector can still overlap even if they do not arrive exactly at the same time. This provides some tolerance and cleaning up of temporal jitter in spike times. Secondly, 'soft' conjunction detectors ($s < S$), which only require that a subset of their inputs receive a spike simultaneously in order to fire, can be used. Soft conjunction detectors provide resilience against a certain proportion of spikes being missing. Thirdly, multiple conjunction detectors can be created, each with a different triggering context, for each spike that is to be reproduced. As long as d of these redundant conjunction detectors activates, the neuron will still spike at the correct time. Redundant conjunction detectors provide further resilience to missing spikes.

In figure 3.6, we demonstrate completion of a stimulus consisting of 25 spikes from a stored memory of 100 spikes. Each stimulus spike has been temporally jittered according to a Normal distribution with standard deviation 4ms. The network completes the missing spikes and cleans up the temporal jitter.

3.2.8 Neural readout

The spiking activity of the Concurrent Recall Network consists of a superposition of all the spatiotemporal spike sequences which it is concurrently recalling. We can ascertain which memories are being recalled by comparing the actual spiking activity with the known training memories, either visually as is done in figure 3.4 or by automated procedures such as the ones described in the following chapters. It is likely that any biological system using the output of a Concurrent Recall Network would not need to 'separate out' the memories making up the activity of the Concurrent Recall Network. However, it is useful to verify that we can construct a neural system which can determine which memories are being recalled by a Concurrent Recall Network, should that be necessary.

One such neural 'readout' mechanism is to construct a layer of 'grandmother' neurons (see section 3.1.4). Groups of grandmother neurons are responsible for detecting whether the network is currently recalling each of the stored memories. Within each group, different grandmother neurons are responsible for detecting which temporal section of a memory has just occurred.

For example, in figure 3.7 a network of 1000 neurons has ten stored periodic memories, six of which are being concurrently recalled. Each stored memory has a different period. During the training of the network, 100 grandmother neurons were created. Each grandmother neuron operates in the same way as an ordinary Concurrent Recall Network neuron. In this example, each grandmother neuron has a single conjunction detector with 4 inputs. The 100 grandmother neurons are divided into ten groups of ten. In the first group, the first neuron is given time-delay connections from the outputs of four neurons which are expected to fire in the first tenth (by time) of the first memory. The second neuron is triggered by spikes in the second tenth, and so on. So if the first memory has a period of 100ms and is present among the spiking activity of the network, we expect the ten neurons in the first group to fire one after the other at 10ms intervals. Neurons in the other nine groups respond similarly to the other nine stored memories. At any point in time, the grandmother neurons indicate which memories are present, and also what phase in its period each memory has reached. In addition to demonstrating a neural readout mechanism for the Concurrent Recall Network, the activity of grandmother neurons provide a useful visualisation of its state.

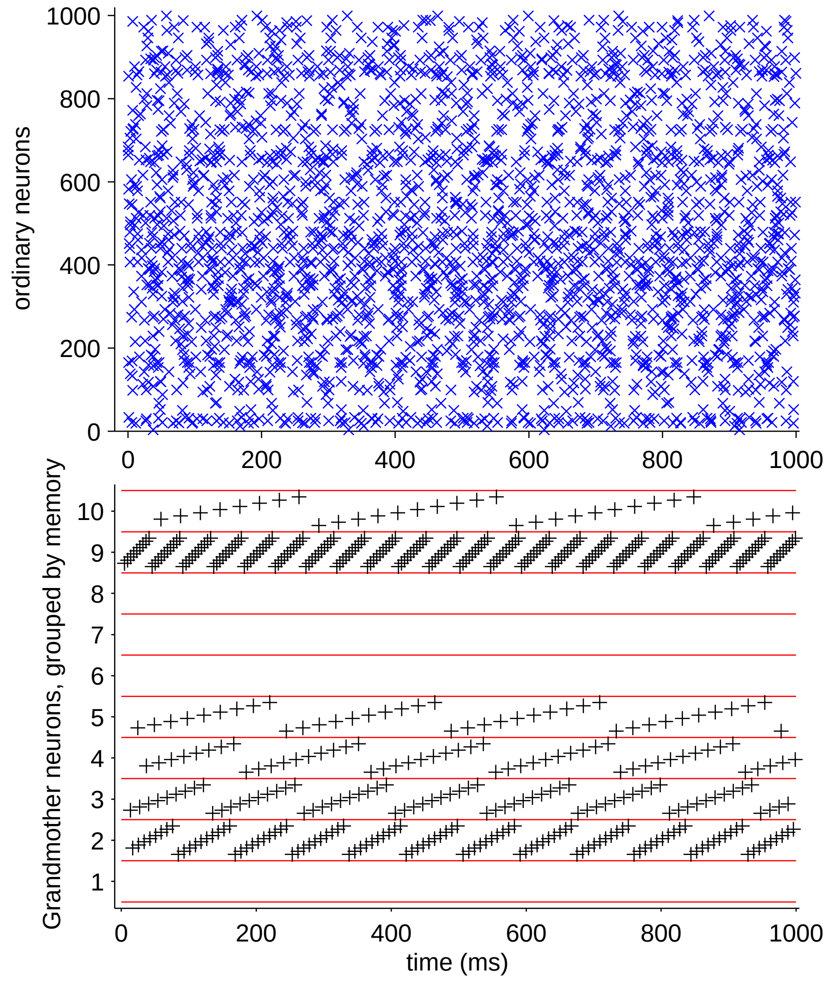


Figure 3.7: Spike raster plot demonstrating concurrent recall of periodic memories with non-equal periods and the use of grandmother neurons as a neural readout of the state of the network. A network of 1000 neurons was trained on ten periodic memories of 50 spikes each. Periods were chosen randomly between 30 and 300ms. Ten groups of ten grandmother neurons were used. Each group was trained to detect one of the ten stored memories, as described in the text. Six of the stored memories were stimulated at the start of the simulation. The top axes show the spiking times of the 1000 ordinary neurons; the bottom axes show the spiking times of the 100 grandmother neurons. Six groups of ten consecutive grandmother neurons, corresponding to the six patterns being recalled, are active. Within each group, the ten grandmother neurons fire in sequence, distributed evenly during the period of the associated memory.

3.2.9 Biological plausibility

As stated in the introduction to this thesis, our aim is not to closely model real biology. However, several aspects of the model mimic real neurons in the brain. Real neurons do have branched dendritic trees. These come in various types and complexities [21], and it is not unreasonable to model these complex structures with the simple 'Christmas tree' structure of figure 3.5.

The plausibility of dendrite branches acting as conjunction detectors is less clear. What is clear is that a biological neuron is a highly complex cell. Even a simple 'passive cable' model of dendrites leads to the conclusion that dendritic trees must be treated as an electrically distributed system, with the location of a synaptic input in the dendritic tree affecting both its effect at the soma and its interactions with other synaptic inputs [71, 80, 8, 55]. In reality, dendrites are not passive carriers of electrical charge, but contain active channels which respond to the local conditions inside the dendrite [46], providing the machinery to carry out further non-linear local computation within the dendritic tree. For example, both simulated models and experiment have demonstrated that voltage-gated ion channels can enhance the efficacy of incoming spikes which arrive nearby in time and space (i.e. on the same dendritic branch), lending weight to the idea that a dendritic tree could perform coincidence detection on multiple groups of inputs independently [56, 54, 89, 65]. It is therefore not implausible to suggest that an individual neuron may be able to carry out relatively complex non-linear computation on its inputs, similar to the computations of a Concurrent Recall Network neuron.

The use of delay lines is also not implausible. Although many axons connect between nearby neurons, resulting in propagation times close to 1ms, connections between brain areas can result in propagation times of the order of 10ms, and unmyelinated axons[4] can give rise to delays up to 100–200ms [51, 58]. Furthermore, synaptic input can take up to 10ms to propagate to the soma after having reached the dendritic tree [72], depending on the location of the synapse within the tree. Delays could also be instantiated by 'chained' connections through other neurons. We can therefore see that a variety of delay mechanisms are available to neural circuitry.

3.2.10 Similarity to existing models

Many neural network models have been created which perform recognition or generation of temporal sequences [49, 13, 88]. Like the Concurrent Recall Network, some models use a distribution of time delays to give the network access to information about its past state [30]. The Concurrent Recall Network is most similar to a construction known as a 'continuous-time Synfire braid' [15, 20].

A Synfire braid is a generalisation of the Synfire chain model [2, 15]. In a Synfire chain,

[4]Myelin is an insulative sheath found around some biological axons. One of its effects is to dramatically increase the speed of action potential propagation along the axon.

packets of synchronous spikes propagate between small pools of neurons within a larger population. Connections between neurons have a fixed delay and integer or real-valued connection weight. Each neuron in the $(n + 1)$th pool receives connections from all (or many) of the neurons in the nth pool. Neurons may participate in many pools, and several distinct pools may be active at any one time. Owing to the equal connection delays, the system effectively changes state synchronously with a fixed period equal to the connection delay, creating a 'chain' of pools of neurons which activate one after another. Since the number of neurons in each pool is typically much smaller than the total number of neurons, it is possible for multiple pools to be concurrently active, resulting in multiple chains progressing in parallel.

Each neuron is typically modelled as having a single internal potential, equal to the sum of the binary outputs of all neurons in the previous time-step in the chain multiplied by their corresponding connection weights. A non-local 'r-Winner Takes All' rule, designed to be a convenient implementation of adaptive global inhibition, sets the output of the neurons with the r largest potentials in a given time-step to 1, and all others to 0, thereby ensuring that exactly r neurons are active at any one moment. As a result, the system tends to re-synchronise the spikes in each volley, cleaning up any temporal jitter introduced into the connection delays [19, 28, 2]. Similar results are obtained when using alternative neuron dynamics including leaky integrate-and-fire and pure coincidence detector neurons [19, 28, 40].

A continuous-time Synfire braid is a Synfire chain that has non-equal, non-discretised delays which obey a consistency rule. The rule requires that all possible routes through the network between two neurons (up to some maximum number of 'hops') incur the same total delay. A recent paper by Izhikevich describes a similar construction of neurons and delay lines to generate precise spatiotemporal spike sequences, which he refers to as "polychronous" spikes [42]. The potential of each neuron in a Synfire braid as formulated in [15] is defined to be the sum of contributions from other neurons' outputs, weighted by the connection strength and delayed by the connection delay. A temporal tolerance is included in the sum, playing a similar role to that which t_{psp} plays in the Concurrent Recall Network. As in the Synfire chain, the neurons with the r highest potentials at any moment are active; all others are not.

A continuous-time Synfire braid is similar to the Concurrent Recall Network in that it uses real-valued delays between neurons and permits recall of spatiotemporal patterns of activity. However, it differs substantially in the construction of each neuron. A Synfire neuron performs a single summation of inputs (the binary outputs of other neurons multiplied by connection weights) and uses a global rule to select the r neurons with the highest activations. A Concurrent Recall Network neuron uses multiple non-linear sub-units (conjunction detectors) and uses a local rule to decide when to emit a spike.

3.3 Simulation of the Concurrent Recall Network

In order to investigate the various properties of the neural network model developed in this chapter, it was necessary to simulate it. In this section we briefly discuss the computer program written for this purpose.

3.3.1 Time-step versus event-driven simulation

Two approaches may be taken to the simulation of a temporal model: time-step and event-driven.

In a time-step simulation, time is divided into discrete steps. Simulating the system involves proceeding through the time steps in sequence, calculating at each step the current state of the system based on the previous state(s). The size of the time-step defines the temporal resolution of the simulation: the larger the time-step, the faster the simulation will run, but the more artefacts will be introduced into the simulated behaviour by the reduced resolution. A time-step approach was used in chapter 2 to model *mus silicium*.

In an event-driven simulation, there is no need for a discretisation of time other than that enforced by the finite precision of the floating-point arithmetic of the computer. The simulation environment does not progress step-by-step through time, but instead jumps ahead to the next *event* scheduled to occur. An event is any occurrence that changes the state of the model. For the Concurrent Recall Network, events include the firing of neurons and the arrival of spikes at synapses. Events cause the state of the system to be changed, and often result in the scheduling of other events for some future time. For example, the firing of neuron A which has a connection to neuron B (delay 4ms) and a connection to neuron C (delay 7ms) might result in two new scheduled events: one 4ms into the future and one 7ms into the future (ignoring any temporal jitter deliberately introduced). Because events are not forced to occur at times which are multiples of a coarse time-step, the temporal resolution of the simulation is the resolution to which the floating point value representing time is stored in the computer.

Event-driven simulations are most appropriate where all activity can be broken down into events that take place instantaneously, allowing the simulation to bypass all the time between events when nothing changes. The Concurrent Recall Network fits this requirement very well, since the state of synapses, dendrites and neurons are modelled with discrete values which change state instantaneously.

In performance terms, each method has its own advantages. An event-driven simulation does not have to laboriously simulate the periods of time between events when nothing is happening. On the other hand, there are computational overheads involved with managing the queue of scheduled events.

We decided to implement an event-driven simulation for the Concurrent Recall Network in order to allow experimentation with precise time delays without the significant quantisation errors that a time-step simulation would incur. The Java programming language was used.

Chapter 4

Capacity of the Concurrent Recall Network

In the previous chapter we described a neural network, which we call the Concurrent Recall Network, capable of concurrently recalling multiple stored spatiotemporal spike patterns. In this chapter we investigate the capacity and other properties of the model.

4.1 Definition of capacity

In a conventional associative memory such as a Hopfield network (section 3.1.1), the capacity is usually defined as the number of memories that can be stored in the network such that the nearest stable state of the network to a stored memory is the same as, or close to, the stored memory. In the case of the Hopfield network with random memories, the system can be treated as a spin glass [11, 29]. Such an analysis reveals several transitions in the behaviour of the system as the number of stored memories per neuron, n/N, is increased. At $n/N < 0.05$, the stored memories are stable states (perfect recall of all memories). For $0.05 < n/N < 0.138$, the memories are not themselves stable states, but have stable states nearby, with less than 1.6% of bits flipped. At $n/N = 0.138$, the memory fails catastrophically. This critical point, $n/N = 0.138$, is usually taken to be the capacity of the Hopfield network.

We will find that the Concurrent Recall Network behaves in a way amenable to a similar definition of capacity. If, for given parameter values, we increase the number of stored memories, then beyond a certain point the network catastrophically fails, just as the Hopfield network does beyond $n/N = 0.138$. We are however faced with the question, which does not apply to the Hopfield network, of how to define the capacity of a network which has the ability to recall more than one stored memory simultaneously. In this chapter we treat the number of patterns being simultaneously recalled as an additional parameter, defining capacity as the number of memories which can be stored, given that any p of them can be reliably concurrently recalled. In some cases it is also interesting to consider the converse question, namely "how many memories can be concurrently recalled given that a certain number of memories have been stored?”.

The capacity depends on the training rule. We consider two training rules: the unlimited training rule defined in the previous chapter, and the limited training rule which we define in section 4.3.

4.2 Unlimited training rule

We begin by investigating the performance of the network when using the *unlimited training rule* defined in section 3.2.5. Recall that under this training rule a new conjunction detector is added to the network for each spike in each training pattern.

4.2.1 Concurrent recall capacity

Simulations were performed using a network of $N = 1000$ neurons, with parameters $s = S = 3$, $d = 1$. These values (defined in section 3.2.2) correspond to the ‘binary sigma-pi’ limit mentioned in section 3.2.2. Each neuron has a number of conjunction detectors, any of which can trigger a spike; each conjunction detector has three inputs, all of which have to receive spikes simultaneously (such that the spikes, each of duration t_{psp}, overlap) to be activated. Because the unlimited training rule was used (with $\mathrm{Q}(\cdot) = \mathrm{Q}_{\text{nearest}}(\cdot)$), the number of conjunction detectors on a particular neuron is equal to the total number of times that neuron fires in all the memories stored so far. No noise was added to the network in this experiment.

Simulations were performed in which the number of stored memories, n, was gradually increased. Each memory consisted of 50 spikes distributed randomly with a uniform distribution over the 1000 neurons and a fixed 100ms period. For each value of n, the ability to simultaneously recall p of those memories was tested by stimulating the network with a superposition of the first five spikes from each of p memories randomly selected from the n stored memories, and simulating the network for five 100ms periods. Simulations in which the rate of spiking rose above ten times the $50p$ expected spikes per period were deemed to have experienced the ‘proliferation failure mode’ which will be discussed in section 4.2.2. For the remaining cases, the presence of each stored memory was tested for among the spikes that occurred in the final period of the simulation. Memories that had at least 80% of their spikes present (to within a 1ms tolerance) were deemed to have been recalled. A score was assigned to each (n, p) pair, being the fraction of stimulated patterns which were recalled, less the fraction of unstimulated patterns which were recalled (but should not have been). Runs failing due to proliferation were assigned a score of zero. In practice, it was found that the details of the metric used to measure the degree of success in each test were not important because the network either performs an excellent job at recalling all the target patterns and no others, or fails completely due to proliferation.

One data set from this experiment is shown in figure 4.1(a), which shows the score obtained for values of n and p between 1 and 2000. Figure 4.1(b) shows the average of 20 sets of results,

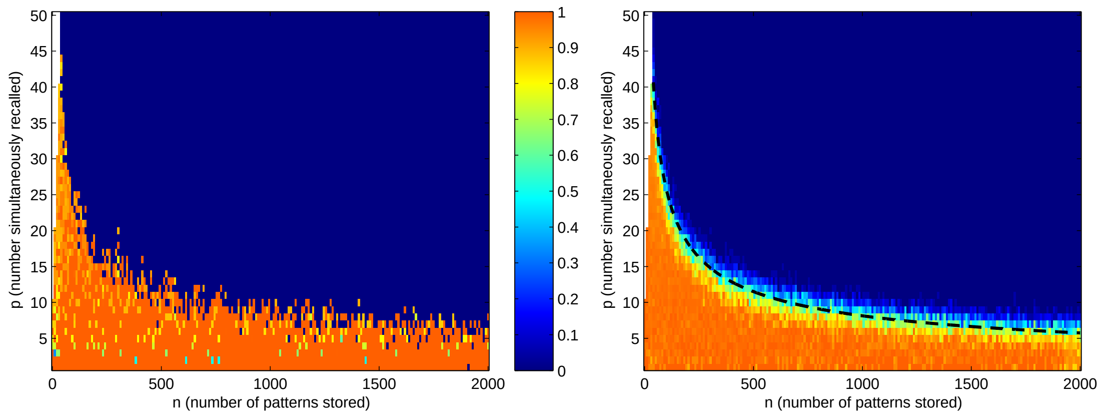


Figure 4.1: Performance of the Concurrent Recall Network using the unlimited training rule, for varying number of stored 50-spike memories (n) and number of concurrently recalled memories (p). The network contained 1000 neurons, and used conjunction detectors with $S = s = 3$. Colour indicates performance of the network, ranging from one (orange; perfect recall of all p memories) to zero (blue; complete failure), using the score defined in section 4.2.1. **(a)** shows an individual recall test at each (n, p) value; **(b)** shows an average over 20 such tests. It can be seen from (a) that performance does not gradually degrade as p is increased, but drops suddenly from close to one (perfect performance) to zero (complete failure, typically due to proliferation). The softer transition visible in (b) is due to trial-to-trial variability in the value of p at which the transition occurs. The dashed black line in (b) is a manually fitted curve of the form $p \propto n^{-1/2}$.

to reduce the random fluctuations in performance due to some patterns being easier for the network to recall than others.

The key result that we can obtain from the data in figure 4.1 is the relatively narrow boundary between the region of (n, p) parameter space in which all p stimulated memories are successfully recalled and the region in which the network fails completely. Fitting a curve to this boundary, as shown by a dashed line in the figure, we see that it is well fitted by the relationship $p \propto n^{-0.5}$.

In order to understand the behaviour shown in figure 4.1, we need an understanding of what is causing the network to fail to recall patterns in the areas where it scores poorly. The next section examines the network's failure mode.

4.2.2 Proliferation failure mode

In the case of the unlimited training rule with no noise, the behaviour of the network can be qualitatively summed up as follows. For low values of n (the number of memories stored) and p (the number stimulated for simultaneous recall), the output of the network is a close match to the superposition of all the stimulated training patterns. If one attempts to recall too many patterns simultaneously, or if the number of stored patterns is too high, the system fails catastrophically. An example of the spiking activity of the network during such a failure

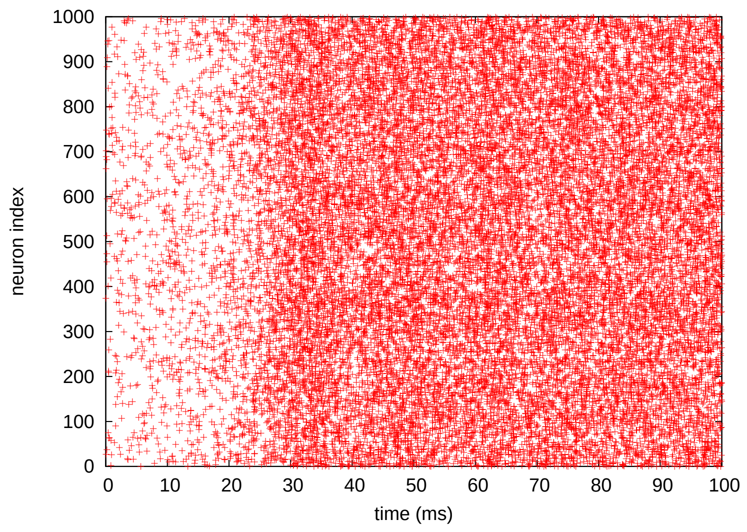


Figure 4.2: Spike rastergram showing proliferation failure mode. The network of 1000 neurons was trained on 85 memories of 50 spikes each. The network was stimulated with the first 6 spikes from all 85 patterns. Crosses denote the firing of neurons.

is shown in figure 4.2. We refer to this failure mode as *proliferation.* It occurs when spurious spikes (spikes which do not belong to any of the stimulated memories) are generated by the network, and these spurious spikes in turn cause more spurious spikes. Spurious spikes rapidly proliferate until the network is swamped with spikes, limited only by the refractory period of the neurons. Such a state does not convey any useful information and is slow to simulate due to the extreme number of spike events the simulation must process. The simulation software is therefore programmed to detect when the rate of spiking rises to an extreme level and will abort the simulation with an error condition.

4.2.3 Analytical prediction for the onset of proliferation

The relationship between the highest number of memories that can be concurrently recalled for a given number of stored memories (or vice versa) can also be explored analytically. In this section we deal only with the unlimited training rule.

Our approach in this section is to consider the conditions under which proliferation will develop. We begin by considering the probability of a neuron firing a spike due to 'accidental' triggering of one of its conjunction detectors by the spikes occurring in the network. For simplicity, we will limit our analysis to the case $s = S$, i.e. the conjunction detectors are like AND gates.

Let r be the rate of spikes occurring in the network. As with all rates referred to in this section, we define the rate to be the expected number of spikes (or other events) per neuron, per unit time. In the numerical results, rates are expressed in units of *per neuron, per millisecond.*

We consider the rate r to be made up from two contributions. The first contribution, r_{g}, is from the 'genuine' spikes which the network is supposed to be recalling, i.e. the stimulated memories. Since the unlimited training rule creates enough conjunction detectors to recall all stored memories, then, in the absence of noise, the stimulated memories are guaranteed to be present among any spurious spikes.[1] The rate of genuine spikes is obtained from the number of memories being simultaneously recalled, p, and the number of spikes in each, m:

$$r_{\mathrm{g}} = \frac{pm}{NT}. \tag{4.1}$$

where N, as before, is the number of neurons in the network and T is the period of each memory. Also contributing to r is the rate of spurious spikes, r_{sp}, which arise from chance activations of conjunction detectors. We therefore have

$$r = r_{\mathrm{g}} + r_{\mathrm{sp}}. \tag{4.2}$$

A conjunction detector has S inputs and (since $s = S$) only activates if all of them concurrently receive a spike. From the viewpoint in which spikes are considered delta functions (section 3.2), the equivalent statement is that spikes arrive at all s inputs within t_{psp} of each other. The rate of activations of the conjunction detector is therefore given by the rate at which spikes arrive at the first input, multiplied by the probability that, given a spike arriving at time t_1 on the first input, spikes arrive on the other $s-1$ inputs within t_{psp} of t_1 and within t_{psp} of each other. If we call this probability p_c, then the rate of activations of the conjunction detector is given by:

$$r_{\mathrm{cd}} = rp_c. \tag{4.3}$$

In the case $s = 2$, a spike is required to arrive at the second input within t_{psp} of time t_1, i.e. between $t_1 - t_{\mathrm{psp}}$ and $t_1 + t_{\mathrm{psp}}$. The probability of one or more spikes arriving during this $2t_{\mathrm{psp}}$ period is given by the probability of one or more events occurring from a Poisson process of mean $2t_{\mathrm{psp}}r$:

$$p_c = 1 - \exp(-2rt_{\mathrm{psp}}) \approx 2rt_{\mathrm{psp}} \qquad (s = 2). \tag{4.4}$$

In the case $s = 3$, not only do we require spikes to arrive at the two other inputs between $t_1 - t_{\mathrm{psp}}$ and $t_1 + t_{\mathrm{psp}}$, but we have the additional constraint that the two spikes must be within t_{psp} of each other, i.e. the constraints on t_2 and t_3 (the arrival time of spikes at the other two inputs) are $(t_1 - t_{\mathrm{psp}}) < t_2 < (t_1 + t_{\mathrm{psp}})$, $(t_1 - t_{\mathrm{psp}}) < t_3 < (t_1 + t_{\mathrm{psp}})$ and $|t_2 - t_3| < t_{\mathrm{psp}}$. Figure 4.3 illustrates the area in t_2, t_3 space that satisfies these constraints. The expected number of spikes per unit area in t_2, t_3 space is r^2, and the area of the constraint-satisfying

[1]For reasons of tractability, we ignore the possibility that a genuine spike is prevented from occurring because a previous spike has left a neuron in its refractory state at the relevant moment.

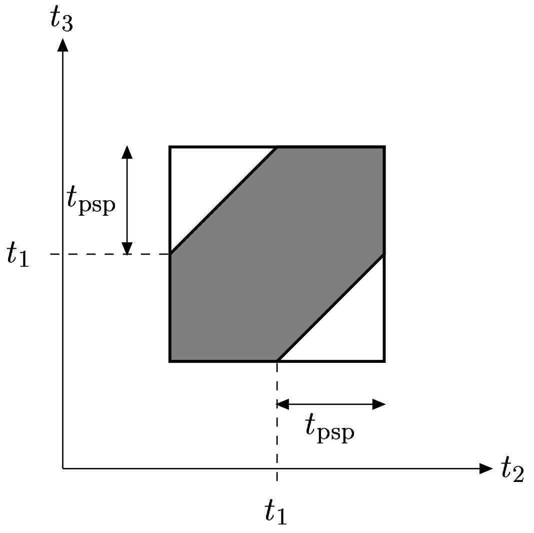


Figure 4.3: The shaded region shows the area in (t_2, t_3) space in which the constraints $(t_1 - t_{\text{psp}}) < t_2 < (t_1 + t_{\text{psp}})$, $(t_1 - t_{\text{psp}}) < t_3 < (t_1 + t_{\text{psp}})$ and $|t_2 - t_3| < t_{\text{psp}}$ are all satisfied, i.e. t_2 and t_3 are within t_{psp} of t_1 and of each other. The shaded area is 3/4 of the area of the enclosing square, which has area $(2t_{\text{psp}})^2$.

region in the figure is $\frac{3}{4}(2t_{\text{psp}})^2$, giving

$$p_c = 1 - \exp\left(-\frac{3}{4}(2t_{\text{psp}}r)^2\right) \approx \frac{3}{4}(2t_{\text{psp}})^2 r^2 \qquad (s = 3). \tag{4.5}$$

In the case $s > 3$, we require the region in $t_2, ..., t_s$ space that satisfies the constraints $(t_1 - t_{\text{psp}}) < t_i < (t_1 + t_{\text{psp}}) \forall i \in \{2, ..., s\}$ and $\max(t_2, ..., t_s) - \min(t_2, ..., t_s) < t_{\text{psp}}$. Generalising figure 4.3 to higher dimensions, the constraint-satisfying volume consists of an $(s-1)$-dimensional hypercube with side length t_{psp} located in one corner of the enclosing hypercube (of side length $2t_{\text{psp}}$), which is swept linearly up to the opposite corner. The volume of the inner hypercube plus the volume swept out is $(t_{\text{psp}})^{s-1}s$, and the expected number of spikes per unit volume in the $(s-1)$-dimensional hyperspace is r^{s-1}, resulting in a general expression for p_c:

$$p_c = 1 - \exp\left(-r^{s-1}(t_{\text{psp}})^{s-1}s\right) \approx r^{s-1}(t_{\text{psp}})^{s-1}s. \tag{4.6}$$

Consequently, the expected rate of activations of a conjunction detector due to random coincidence of spikes is

$$r_{\text{cd}} = rp_c \approx (t_{\text{psp}})^{s-1}r^s s. \tag{4.7}$$

We now consider the rate of spurious firing of each neuron, r_{sp}. Spurious spikes received on a neuron's inputs will in turn lead to a certain rate of spurious spikes being generated by the neuron. We can therefore define a function $f(r_{\text{sp}})$ which is the rate of generated spurious spikes. Since one neuron's output is another neuron's input, r_{sp} will evolve according to the iterative map

$$r_{\text{sp}} \leftarrow f(r_{\text{sp}}). \tag{4.8}$$

We now derive an expression for the function $f(r_{\text{sp}})$.

In the unlimited training rule, a conjunction detector is created somewhere in the network for each spike in each stored memory. The average number of conjunction detectors per neuron, D, is therefore

$$D = \frac{nm}{N}. \tag{4.9}$$

where $m = |\mathcal{M}|$ is the number of spikes in each of the n stored memories.

We assume $d = 1$, i.e. the neuron fires as soon as any one of its conjunction detectors is activated. The total rate of conjunction detector activations for the neuron is given by Dr_{cd}. To a first approximation, this is also the rate at which the neuron will fire:

$$f(r_{\text{sp}}) \approx Dr_{\text{cd}} = \alpha \cdot (r_{\text{g}} + r_{\text{sp}})^s \tag{4.10}$$

where we have substituted for r_{cd} from equation 4.7 and, for convenience, defined

$$\alpha \equiv (t_{\text{psp}})^{s-1} Ds. \tag{4.11}$$

We can improve our theoretical model by taking into account the neuron's refractory period, t_{refr} (section 3.2.2). If a conjunction detector activation occurs within t_{refr} following the neuron firing, it is rendered impotent and will not cause another spike. The correct rate of spurious firing is therefore given by subtracting from equation 4.10 the rate of impotent activations, which we can model as the rate of conjunction detector activations multiplied by the Poisson probability that the neuron fired within the previous t_{refr}, given that the neuron fires at rate r:

$$\begin{aligned} f(r_{\text{sp}}) &= Dr_{\text{cd}} - Dr_{\text{cd}} P(\text{activation is impotent}) \\ &= Dr_{\text{cd}} - Dr_{\text{cd}}\Big(1 - P(\text{zero firings in previous } t_{\text{refr}})\Big) \\ &= Dr_{\text{cd}} - Dr_{\text{cd}}\left(1 - e^{rt_{\text{refr}}}\right) \\ &= Dr_{\text{cd}} e^{-(r_{\text{sp}}+r_{\text{g}})t_{\text{refr}}} \qquad (4.12) \\ &= \alpha (r_{\text{g}} + r_{\text{sp}})^s e^{-(r_{\text{sp}}+r_{\text{g}})t_{\text{refr}}}. \qquad (4.13) \end{aligned}$$

Our improved theoretical model of the spurious spiking rate is therefore a dynamical system governed by the following iterative mapping:

$$r_{\text{sp}} \leftarrow \alpha \cdot (r_{\text{g}} + r_{\text{sp}})^s e^{-(r_{\text{g}}+r_{\text{sp}})t_{\text{refr}}}, \tag{4.14}$$

which has a fixed point $r_{\text{sp}} = r^*_{\text{sp}}$ defined implicitly by

$$r^*_{\text{sp}} = \alpha \cdot (r_{\text{g}} + r^*_{\text{sp}})^s e^{-(r_{\text{g}}+r^*_{\text{sp}})t_{\text{refr}}}, \tag{4.15}$$

or, neglecting the correction factor which takes account of the refractory period,

$$r_{\mathrm{sp}} \leftarrow \alpha \cdot (r_{\mathrm{g}} + r_{\mathrm{sp}})^s, \tag{4.16}$$

$$r^*_{\mathrm{sp}} = \alpha \cdot (r_{\mathrm{g}} + r^*_{\mathrm{sp}})^s. \tag{4.17}$$

As we will see in figure 4.4, this approximation is reasonable for the purpose of ascertaining whether a fixed point exists.

Figure 4.4 shows cobweb diagrams [82] for equation 4.14, for several values of r_{g}. Shown on the same plots, in green, is equation 4.10. We see that it is a good approximation to equation 4.13 in the range of r_{sp} values which correspond to the location of the fixed point.

In figure 4.4(a), r_{g} is small enough that a fixed point exists: the rate of spurious spikes, r_{sp}, converges and proliferation does not occur. In figure 4.4(b), r_{g} is at the critical value above which the fixed point disappears. The approximate critical value of r_{g} which we calculate in equation 4.20 below corresponds to the value of r_{g} for which the green curve (the approximation, equation 4.10) just touches the diagonal line. We can see from the figure that it will do so at a value of r_{g} fractionally lower than the value of r_{g} depicted in figure 4.4(b), since the green curve is already above the diagonal line in that figure.

In figure 4.4(c), r_{g} is greater than its critical value, which we will call $\hat{r}_{\mathrm{g}}$. The stable fixed point we saw in figure 4.4(a,b) disappears, and r_{sp} rises to much higher values, i.e. proliferation sets in. We discuss the behaviour of this theoretical model for values of r_{g} greater than the critical value in more detail in section 4.2.4. For now, we wish to derive an expression for the critical rate of genuine spikes, $\hat{r}_{\mathrm{g}}$.

Figure 4.4 shows that the maximum value of r_{g} for which r_{sp} reaches a stable fixed point when iterated starting from $r_{\mathrm{sp}} = 0$ occurs when the curve $f(r_{\mathrm{sp}})$ touches the line $f(x) = x$ without crossing it. Since the slope of the curve must be 1 at the the point where the curve touches the line, the critical value of r_{g} is determined by the condition that the solution of

$$\frac{\partial f(r_{\mathrm{sp}})}{\partial r_{\mathrm{sp}}} = 1 \tag{4.18}$$

is also a fixed point.

To allow us to determine an analytical solution, we use the approximation in equation 4.10, which we have seen is valid in the region of the fixed point of interest, for sensible parameter values. Equation 4.18 is then solved when

$$\begin{aligned} \frac{\partial}{\partial r_{\mathrm{sp}}} \Big(\alpha \cdot (r_{\mathrm{g}} + r_{\mathrm{sp}})^s \Big) &= 1 \\ \implies \alpha s (r_{\mathrm{sp}} + r_{\mathrm{g}})^{s-1} &= 1. \end{aligned}$$

Hence

$$r^*_{\mathrm{sp}} = \left(\frac{1}{\alpha s} \right)^{\frac{1}{s-1}} - r_{\mathrm{g}}. \tag{4.19}$$

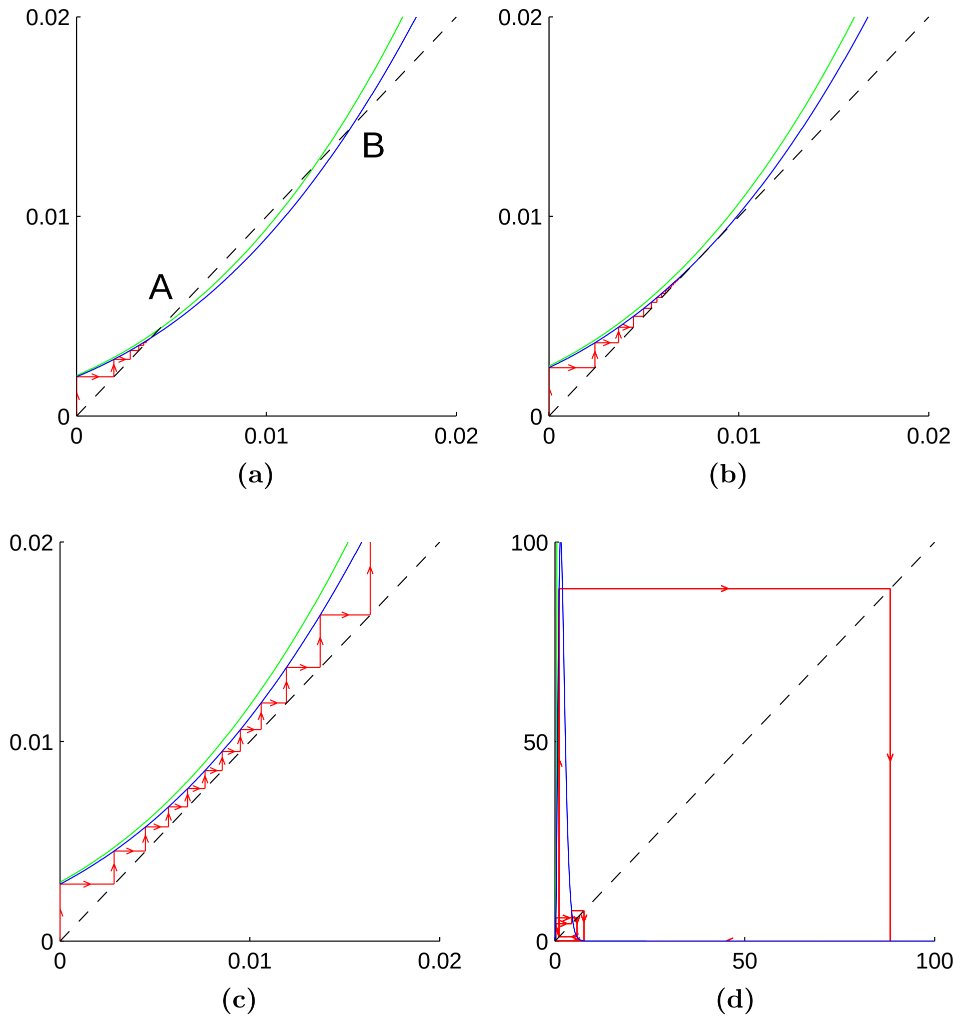


Figure 4.4: Cobweb diagrams for the iterative map in equation 4.14. The horizontal coordinate represents the current value of r_{sp}; the vertical coordinate represents the next value of r_{sp}, $f(r_{sp})$. The diagonal line shows $f(r_{sp}) = r_{sp}$. The blue curve shows equation 4.13; the green curve shows the approximation to it given by equation 4.10. We see that in the range of r_{sp} values containing the fixed point(s) in (a,b), the green curve is a good approximation to the blue curve. Red arrows show the evolution of r_{sp} as we iterate equation 4.14 starting from $r_{sp} = 0$. Other parameters correspond to the values used in the network simulated in figure 4.1, at $n = 1000$. **(a)** $r_g = 0.0150$. r_{sp} converges to a stable fixed point at A. **(b)** $r_g = 0.0161$. The blue curve just grazes the diagonal line. This value of r_g is the largest one for which a fixed point exists. The green approximation will graze the diagonal line at a fractionally lower value of r_g. **(c)** $r_g = 0.0170$. The fixed point has vanished: r_{sp} will grow to much larger values. **(d)** Same as (c), zoomed out by a factor of 5000. At high-enough values of r_{sp}, the blue curve eventually crosses back over the diagonal line, resulting in an alternative fixed point. In situations such as (c), then, r_{sp} does not diverge to infinity, but enters periodic solutions, taking implausibly high values of r_{sp} most of the time. This is discussed further in the text.

Substituting this value into equation 4.17, rearranging for r_g, and simplifying, we obtain the condition that r_g must obey for there to be a fixed point:

$$r_\mathrm{g} \leq \left(\frac{s-1}{s}\right)\left(\frac{1}{\alpha s}\right)^{\frac{1}{s-1}} = \hat{r}_\mathrm{g}. \tag{4.20}$$

Substituting for α, r_g and D from equations 4.11, 4.1 and 4.9, the condition becomes

$$\frac{pm}{NT} \leq \left(\frac{s-1}{s}\right)\left(\frac{N}{nms^2(t_\mathrm{psp})^{s-1}}\right)^{\frac{1}{s-1}}. \tag{4.21}$$

Rearranging, we obtain

$$n \leq \frac{1}{s^2}\left(\frac{s-1}{s}\right)^{s-1}\left(\frac{T}{pt_\mathrm{refr}}\right)^{s-1}\left(\frac{N}{m}\right)^{s}. \tag{4.22}$$

Finally we are in a position to predict the relationship between the maximum number of patterns that can be simultaneously recalled without triggering the onset of proliferation, and n, the number of stored memories. From equation 4.21,

$$p_\mathrm{max} \propto n^{-1/(s-1)}. \tag{4.23}$$

In the case $s = 3$, $p_\mathrm{max} \propto n^{-1/2}$, which agrees with the curve manually fitted to figure 4.1.

Unfortunately, although this model correctly predicts the relationship $p_\mathrm{max} \propto n^{-1/2}$, the constant of proportionality that it predicts, using the parameter values for the simulation in figure 4.1, is too large by a factor of four. Possible reasons for this discrepancy between the prediction of the analytical model and observed results are discussed in section 4.2.5.

4.2.4 Behaviour of the theoretical model for $r_\mathrm{g} > \hat{r}_\mathrm{g}$

Although it appears from figure 4.4(c) that r_sp will diverge to infinity if $r_\mathrm{g} > \hat{r}_\mathrm{g}$, we see in figure 4.4(d), which depicts the same value of r_g but is plotted on a different scale, that this is not the case. At r_sp values two orders of magnitude larger than those shown in figure 4.4(c), the blue curve reaches a maximum and eventually returns below the diagonal line, limiting the maximum value r_sp can obtain. This rate-limiting effect is caused by the neurons' refractory periods.

The maximum value that r_sp can obtain, according to figure 4.4(d), is approximately 100. This corresponds to each neuron firing 100 times per millisecond, which is clearly impossible: the absolute maximum firing rate of each neuron is $1/t_\mathrm{refr}$, i.e. 0.5 times per millisecond. We believe this discrepancy to be due to the fact that the theoretical model in equation 4.13 does not adequately model the true nature of the Concurrent Recall Network at high spiking rates. Two particular failings of the theoretical model at high firing rates are the approximation in equation 4.6 and the fact that it models the refractory period in a soft, probabilistic manner, whereas in fact the refractory period is a hard limit on firing rate. Equation 4.3 is also not valid for firing rates close to $1/t_{\text{refr}}$.

Nonetheless, the theoretical model of equation 4.14 captures the qualitative behaviour that the firing rate cannot diverge to infinity, as it would if we used only the approximation in equation 4.16. It is interesting to note that the theoretical model predicts oscillating behaviour for r_{sp} when $r_{\text{g}} > \hat{r}_{\text{g}}$. 'Proliferation oscillations' — bouts of proliferation separated by periods of relative calm — are a phenomenon which have been observed occasionally in simulations of the Concurrent Recall Network. These could be qualitatively explained by the behaviour of our theoretical model in the overloaded regime $r_{\text{g}} > \hat{r}_{\text{g}}$ (figure 4.4(d)): r_{sp} spends some time bouncing around relatively low values, then every so often it jumps up to a high value before being returned to low values.

4.2.5 Limitations of the analytic model

In addition to the limited validity at high spiking rates discussed in the previous section, we note here some other limitations of the analytical model.

The analytical model assumes that the stimulated memories are always correctly recalled, resulting in a fixed number of 'genuine' spikes. However, a high level of spiking in the network will mean that neurons spend a significant proportion of the time in a refractory state, during which they would not be able to fire a 'genuine' spike at the correct time. This adds an additional layer of complexity which we do not attempt to model.

We note also that the analytical model we used cannot model the rate at which r_{sp} evolves over time. We presented the evolution of r_{sp} as an iterative map with discrete updates; in fact r_{sp} will evolve continuously in a manner which depends on the distribution of time delays of connections.

Finally, r_{sp} is of course a statistical approximation to the true nature of the network, which consists of discrete events. Fluctuations in the instantaneous rate of spurious pikes could cause r_{sp} to jump to a value to the right of the unstable fixed point labelled 'B' in figure 4.4(a). Once in that state, it would no longer converge to the stable fixed point, labelled 'A'. Our analytical model makes no attempt to model this.

4.2.6 Capacity as a function of number of neurons

One of the key properties of a neural network memory is how its performance scales as the number of neurons is increased. To determine this property for the Concurrent Recall Network using the unlimited training rule, simulations were run in which the number of neurons, N, and the number of stored patterns, n, was varied. Exploration was initially restricted to simultaneous recall of seven memories ($p = 7$). Figure 4.5 is typical of the outcome of such simulations, showing the success rate (the score defined in section 4.2.1) as a function of the number of patterns stored, for various numbers of neurons in the network. The line for $N = 1000$ corresponds to a horizontal cross section through figure 4.1(b) at $p = 7$. Figure 4.5

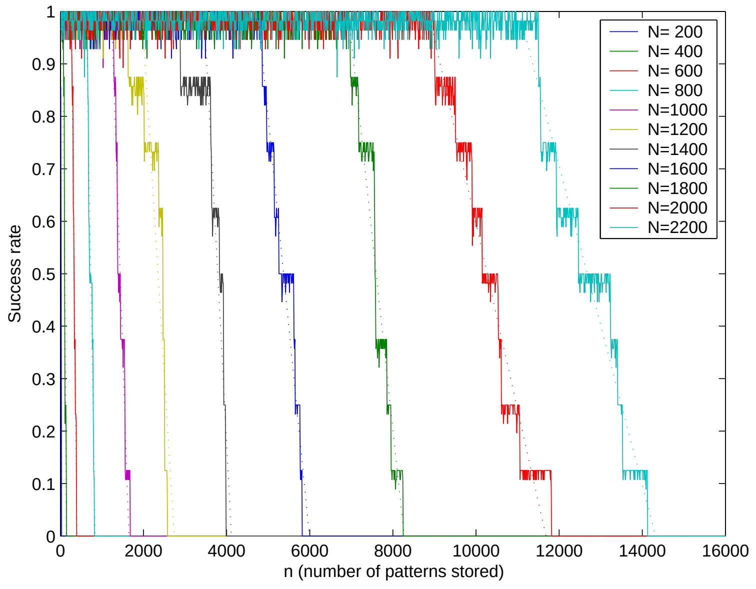


Figure 4.5: Performance of the network as a function of the number of stored patterns, n, for various numbers of neurons, N. Each memory contains 50 spikes. The network used the unlimited training rule ($S = s = 3$) and was stimulated with the first six spikes from 7 of the stored memories. The vertical axis shows the average over 8 trials of the success score defined in section 4.2.1.

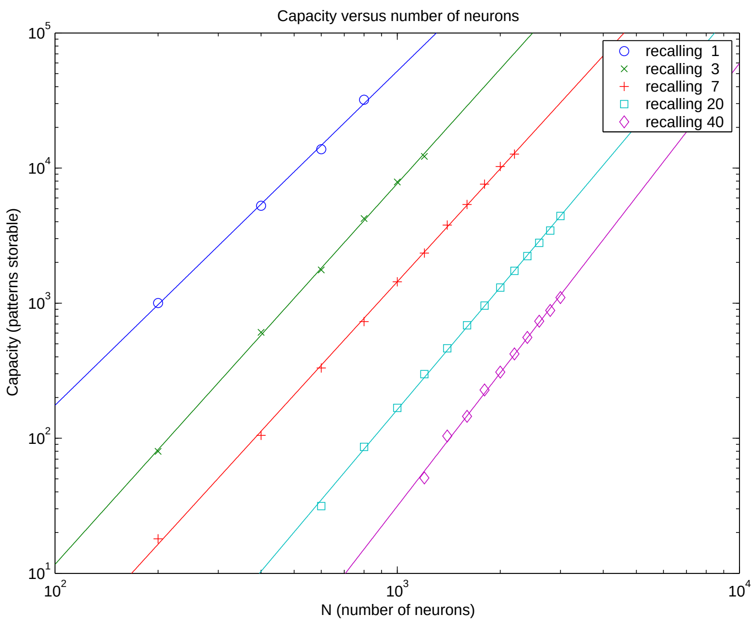


Figure 4.6: Capacity of the network versus number of neurons (unlimited training rule). Capacity is defined as the largest number of memories which can be stored if the network is still to be able to perform the required recall task. The red data points are derived from the fits to the data in figure 4.5 as described in the text; other data points come from similar experiments with different values of p, the number of memories being concurrently recalled.

shows scores averaged over eight runs; this averaging, combined with the fact that scores tend to be close to zero or one, is the origin of the steps at a spacing of 1/8 in score which can be seen in some curves. For example, the lowest step, close to a score of 1/8, corresponds to the cases in which the network successfully recalled the required 7 test memories in only one out of the eight runs. The graph shows that the network tends to perform close to optimally for values of n up to some N-dependent critical value, beyond which performance rapidly drops to zero when proliferation is triggered.

Data such as those in figure 4.5 were further processed by fitting to each curve a piecewise linear function which takes the value 1 for values of n below a parameter n_1, then linearly decays to reach a value of zero at $n = n_2$, and takes the value zero for values of n above n_2. The fits are shown with dotted lines in figure 4.5. The value of n corresponding to the point at which the fitted line crosses a score of 0.5 (i.e. $(n_2 - n_1)/2$) was taken as the capacity, C, of the network in the particular configuration corresponding to that line.

The capacity values obtained in this manner for various values of N and various numbers of simultaneously recalled memories are plotted in figure 4.6, and are all well fitted by the relationship $C \propto N^{2.7}$. This power law is in fair agreement with the analytical prediction: from equation 4.22, the capacity (i.e. the maximum value of n before the onset of proliferation) is proportional to $(N)^s$, i.e. N^3. The small discrepancy in the exponent is probably due to the imperfections in our analytical model already mentioned.

At face value, an exponent larger than one is a surprising result, since we would not expect the capacity of a network to grow faster than linearly in the number of neurons (recall that for the Hopfield network, the capacity, if defined as the number of memories which can be almost perfectly recalled, is linear in the number of neurons). However, the use of the unlimited training rule in these experiments means that as more patterns are stored in the network, the computational complexity of the network hardware is itself increased, since each additional memory stored adds an additional m conjunction detectors somewhere in the network. It is therefore not quite fair to compare this capacity result directly to a more conventional model which has a fixed quantity of hardware available. A Concurrent Recall Network with such a constraint is explored later, in section 4.3.

4.2.7 Capacity dependence on conjunction detector parameters

Two of the Concurrent Recall Network's most fundamental parameters are the parameters s and S, which determine the configuration of the conjunction detectors in each neuron. In order to determine the effect of these parameters on the capacity of the network, the experiment described in the previous section was repeated with various settings of s and S. Results are shown in figure 4.7. Straight lines have been fitted to the data and their slopes indicated on the graphs. It can be seen that the slope of the capacity curve depends on s, the number of inputs to each conjunction detector required to be simultaneously active. The nature of the dependence of the capacity curves on s and S can be understood in terms of the proliferation failure mode.

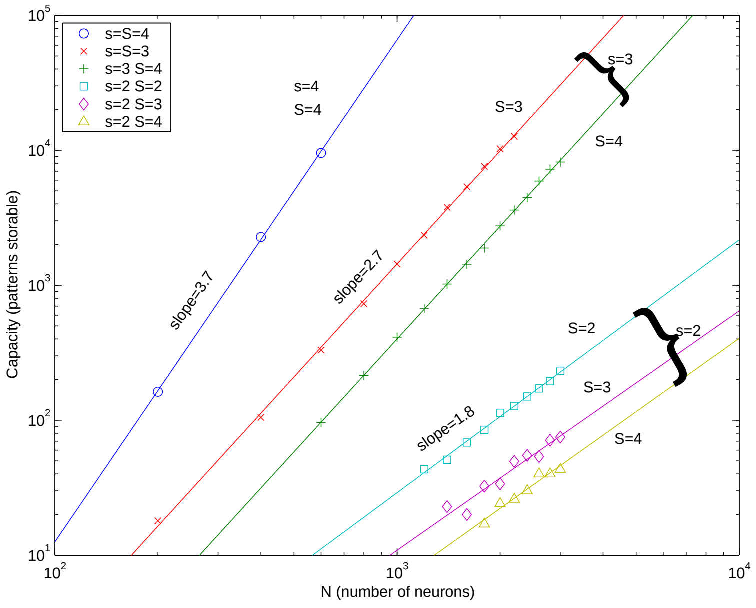


Figure 4.7: Dependence of capacity curves on conjunction detector configuration. As for figure 4.6, one set of points ($S = s = 3$) is derived from the fits in figure 4.5. The other sets of points are derived from similar simulation in which s and S were varied. The number of patterns being simultaneously recalled is seven.

In the $S = s$ case, our analytical model from section 4.2.3 predicts the capacity to be proportional to $(N)^s$. The experiment summarised in figure 4.7 found the actual exponents 1.8, 2.7 and 3.7 for $S = s = 2$, 3 and 4 respectively. The agreement in exponents is fair, but not perfect. Section 4.2.5 outlines some weaknesses in the analytical model which may be to blame.

The analytical model developed in section 4.2.3 applies only for $S = s$. We can, however, qualitatively understand the other cases displayed in figure 4.7. For a given value of s, a higher value of S means that a conjunction detector has more opportunity to detect accidental coincidences on its inputs. Hence, proliferation occurs at a lower density of spikes, and the capacity is reduced.

4.3 Limited training rule

In the previous section we examined the performance of the Concurrent Recall Network when using the unlimited training rule. That training rule is somewhat unusual in that it involves adding a new conjunction detector to the network for each spike in each memory to be stored. A more conventional approach is for the training rule of a neural network to have available to it a fixed quantity of hardware (e.g. neurons and connections or potential connections between them). In this section we describe a training rule for the Concurrent Recall Network which satisfies that constraint, and assess its performance.

4.3.1 Definition of the limited training rule

The limited training rule operates on a similar principle to the unlimited training rule described in section 3.2.5. The difference is that whereas the unlimited training rule starts with no conjunction detectors available and adds new ones as memories are stored, the limited training rule starts with a fixed number, D, of conjunction detectors available on each neuron. Formally, the limited training rule is defined by the following algorithm:

Algorithm 2 Limited training algorithm

```
while unused conjunction detectors remain on neurons which participate in any of the
training patterns do
  Pick a target spike $(n_i, t_i)$ at random from all spikes in all training memories
  Let $\mathcal{M}$ be the training pattern containing $(n_i, t_i)$
  if neuron $n_i$ has any unused conjunction detectors then
    Identify an unused conjunction detector, $\gamma$, on neuron $n_i$
    Let $\mathcal{P} = \mathrm{Q}(\mathcal{M}, n_i, t_i)$, i.e. $S$ spikes forming a subset of $\mathcal{M}$.
    for all spikes $(n_j, t_j) \in \mathcal{P}$ do
      Create a connection from neuron $n_j$ to an unused input on the new conjunction
      detector, $\gamma$, with propagation delay $t_i - t_j \mod T$.
    end for
  end if
end while
```

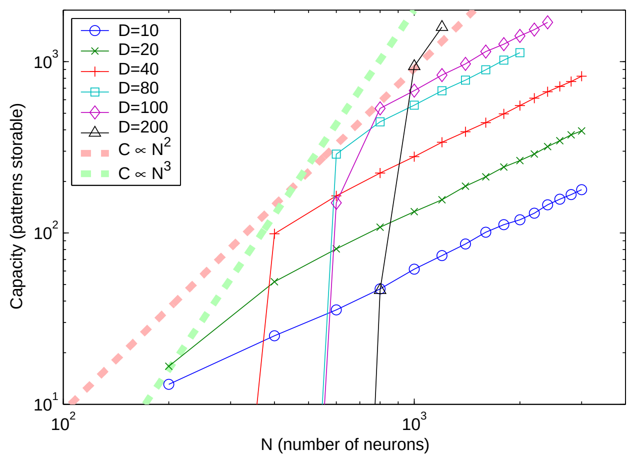
As in section 3.2.5, $\mathrm{Q}(\mathcal{M}, n_i, t_i)$ is the function which selects which S spikes in a given pattern $\mathcal{M}$ are to be used as trigger spikes for the spike at time t_i on neuron n_i.

In the following sections we examine the capacity of the Concurrent Recall Network under the limited training rule just defined, using $\mathrm{Q}(\cdot) = \mathrm{Q}_{\mathrm{uniform}}(\cdot)$ (see section 3.2.5).

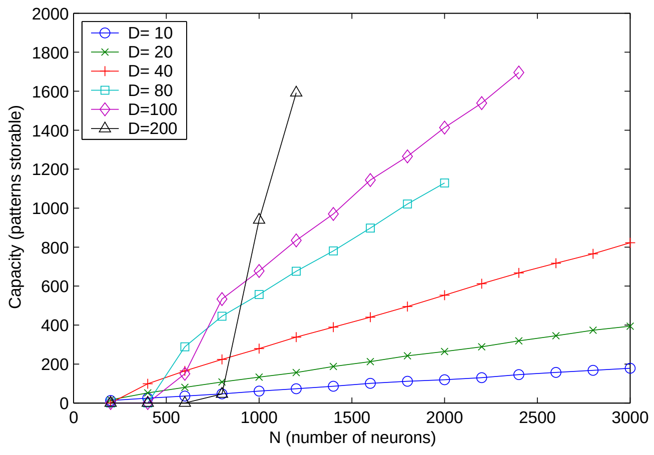
4.3.2 Capacity of the Concurrent Recall Network with limited training rule

Figure 4.8 shows the capacity-versus-N curve for the network using the limited training rule. As with the previous capacity graphs, each data point on the graph is determined by an automated fit to an intermediate dataset similar to the one in figure 4.5. Parameters were the same as for figure 4.6: seven patterns were being recalled concurrently, the conjunction detectors were configured to have $s = S = 3$ and each memory contained 50 spikes. The figure shows the capacity for various numbers of conjunction detectors per neuron, D.

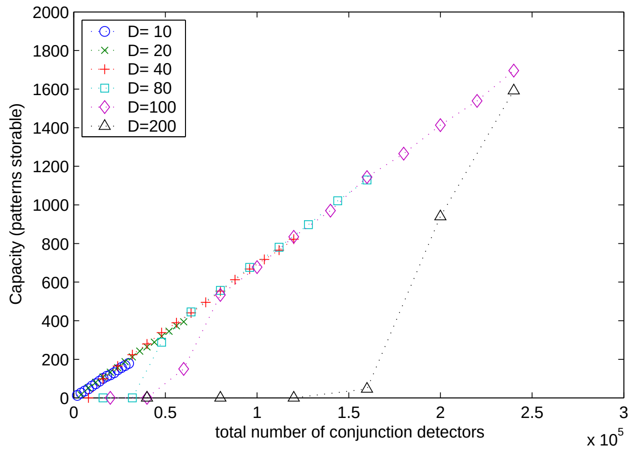
We see from figure 4.8(a,b) that, for large values of N, the limited training rule produces a linear relationship between the capacity and the number of neurons, i.e. a constant capacity per neuron. Providing a greater number of conjunction detectors per neuron increases the capacity of the network. Figure 4.8(c) shows that the majority of the data points fit closely a linear relationship between the capacity and the total number of conjunction detectors on all neurons, ND, with a constant of proportionality corresponding to 0.0078 memories (0.35 spikes) per conjunction detector (for the particular parameter settings used).



(a)



(b)



(c)

Figure 4.8: Capacity under the limited training rule with various numbers of conjunction detectors per neuron (D), $s = S = 3$, 7 patterns were being simultaneously recalled. Runs which aborted due to proliferation are plotted as having a capacity of zero. Data points are joined by lines to guide the eye only. **(a)** Capacity as a function of number of neurons. The graph is shown on a log-log scale for ease of comparison with previous figures; however the relationship between capacity and N on the straight sections of the curves is in fact linear. The thick dashed lines are discussed in the text. **(b)** shows the same data plotted on linear axes. **(c)** plots the same data against the total number of conjunction detectors rather than the total number of neurons.

However, the data displayed in figure 4.8 do not all follow the linear relationship between capacity and number of conjunction detectors identified above. The capacity drops below the linear trend for small values of N (the point of departure from the linear trend appears to be D-dependent). In this experiment, runs which aborted due to proliferation were assigned a capacity of zero. The fact that most of the data series plotted drop quickly to zero after they leave the linear $C \propto ND$ relationship suggests that it may be proliferation that causes this deviation and some other failure mode which limits the capacity in the linear regime.

The failure mode in the linear regime will be considered in section 4.3.3. We now investigate the behaviour of proliferation under the limited training rule.

The analytical model for the onset of proliferation that we developed in section 4.2.2 can be applied to the limited training rule with only one modification. In the unlimited training rule, the average number of conjunction detectors per neuron, D, was given by nm/N. Here, it is an external parameter. Substituting for r_g and α in equation 4.20, we obtain the following condition which must hold for proliferation to be avoided:

$$\frac{pm}{NT} \leq \left(\frac{s-1}{s}\right)\left(\frac{1}{Ds^2(t_{\text{psp}})^{s-1}}\right)^{\frac{1}{s-1}}. \tag{4.24}$$

Note that now D is a constant with no n-dependence, the condition for the avoidance of proliferation no longer depends on n, the number of memories stored. Treating s, m and T as constants, we find that values of N smaller than a critical value $\hat{N}$, given by

$$\hat{N} \propto D^{1/(s-1)}, \tag{4.25}$$

are liable to induce proliferation.

Equation 4.25 suggests that proliferation will occur under the limited training rule if N is smaller than a D-dependent critical value. That is exactly the behaviour we observe in figure 4.8. Intuitively, the smaller N is, the higher the average rate of firing, since the same number of spikes are squeezed into a smaller population of neurons. Also, the larger the number of conjunction detectors, the more opportunities there are for spurious coincidences to occur. We therefore expect both decreasing N and increasing D to make proliferation more prominent. Both of these are apparent in figure 4.8.

We can compare our theoretical result (equation 4.25) with the experimental data by plotting on figure 4.8(a) the boundary beyond which proliferation is predicted to take hold. By combining equation 4.25 with our observation that $C \propto ND$ in the linear, non-proliferating regime, we obtain a curve which we can plot on the figure:

$$\hat{C} \propto \hat{N}D \propto \hat{N}\hat{N}^{s-1} \propto \hat{N}^s. \tag{4.26}$$

Thus, in the case of $s = 3$, plotting the line $C \propto N^3$ onto figure 4.8(a) should, if the analytical model matches observed behaviour, provide a line which intersects each of the capacity curves at the point where it stops following the linear trend as N decreases. The thick dashed green line in figure 4.8(a) shows $C \propto N^3$, with the constant of proportionality adjusted by hand. $C \propto N^2$ is also plotted for comparison.

The data available in figure 4.8 is not sufficient to make a reliable judgement as to whether $C \propto N^3$ has exactly the correct exponent to match the observed capacity curves. Given the slight inaccuracy in the N-dependence of the proliferation analysis noted in section 4.2.6, it would not be surprising to find that our prediction for $\hat{N}$ is not exactly correct. It is, however, clear from figure 4.8(a) that an exponent between 2 and 3 will indeed specify a boundary which correctly demarcates the change of regime from extinction- to proliferation-limited.

4.3.3 Extinction failure mode

Examination of the spiking activity in simulations conducted in the linear part of the capacity curves reveals that, as suggested by the above analysis, it is not proliferation but a different failure mode that limits capacity in that regime. As the number of stored memories is pushed higher, there is no proliferation of spikes, but instead the spiking activity peters out into nothing. We refer to this as the *extinction* failure mode.

We can understand this alternative failure mode by considering the nature of the limited training rule. Whereas the unlimited training rule added an additional conjunction detector per learnt spike (thereby adding another opportunity for an accidental random coincidence of spikes to be detected), the limited training rule has a constant number of coincidence detectors. As more patterns are stored, the probability of accidental coincidences does not increase, and there is no reason for proliferation to occur. Instead, there comes a point when there are not enough conjunction detectors for the training memories to be stored sufficiently well to sustain themselves during recall. In this scenario, some (or all) of the memories will fail to be recalled even when stimulated.

4.3.4 Possible modifications to the limited training rule

The limited training rule as defined in section 4.3.1 makes no attempt to optimise the allocation of conjunction detectors on a particular neuron to spikes in target memories, but instead assigns conjunction detectors to target spikes at random. A strategy which might increase the capacity (in the linear, extinction regime) would be to allocate an equal number of conjunction detectors to each spike to be stored, to avoid the situation in which a memory fails to be stored adequately because a small number of its spikes happened to not be allocated a conjunction detector. Another strategy would be for a neuron to allocate a fixed number of conjunction detectors to each new spike it is asked to learn, rather than sharing them out over all spikes it has ever been asked to learn. When the neuron's supply of conjunction detectors is exhausted, it would recycle the conjunction detectors allocated to older memories. This should allow the network to always be able to recall the R most recently trained memories, with new memories gradually overwriting old ones. Both of these alternative strategies should increase the number of memories that can be stored before the extinction failure mode occurs.

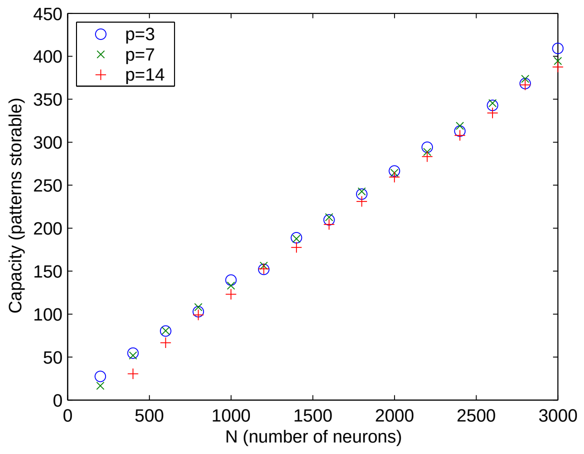


Figure 4.9: Capacity of the Concurrent Recall Network with limited training rule, showing the effect of recalling different numbers of memories, p, simultaneously. The number of conjunction detectors per neuron was 20.

However, the limited learning rule specified in section 4.3.1 has a more biologically plausible flavour to it, since it does not require any sophisticated planning of the training programme. We will therefore continue to use it in the following experiments.

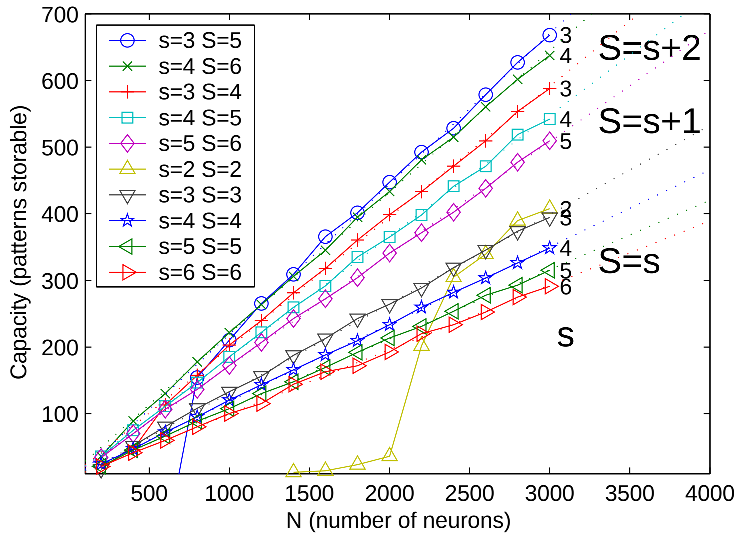
4.3.5 Capacity dependence on number of concurrently recalled memories

Figure 4.9 shows experimentally-determined capacity curves, using the limited training rule, for three different numbers of simultaneously recalled memories (p). 20 conjunction detectors per neuron were used. The figure shows that, broadly, the capacity of the network when using the limited training rule does not depend on the number of simultaneously recalled memories. This is consistent with our understanding of the extinction failure mode, which represents a failure to adequately *store* memories, and is not affected by how we attempt to *recall* them. Nonetheless, we would expect that there must be a value of p above which the average rate of spikes would be sufficiently high to cause the proliferation failure mode to occur.

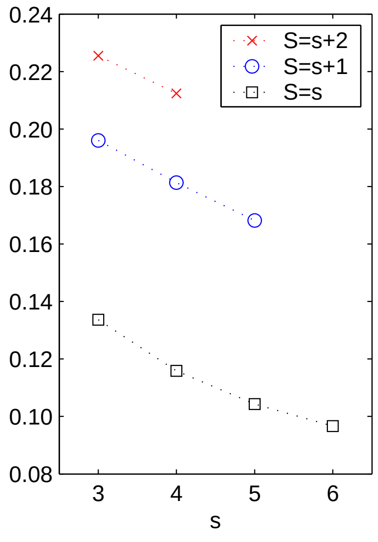
4.3.6 Capacity dependence on conjunction detector configuration

Returning to the $p = 7$ case, we can examine the effect of S and s on capacity, as we did for the unlimited training rule. Figure 4.10 shows the results of simulations for various values of these parameters. Here we can once again see two regimes. Most of the data points are consistent with a linear relationship between capacity and number of neurons, with the constant of proportionality depending on s and S. Values for the constant of proportionality are shown in figure 4.10(b). In a small number of cases, however, the capacity falls below this linear relationship, due to the onset of proliferation.

In the extinction (linear) regime, we can see two main effects. The dominant effect is that, for a given value of s, larger S gives better performance. But for a given value of S, smaller s is better. These observations make sense given the nature of the extinction failure mode.



(a)



(b)

Figure 4.10: (a) Capacity of the Concurrent Recall Network with limited training rule, showing the effect of conjunction detector configuration type. The number of conjunction detectors per neuron was 20. Dotted lines show straight lines fitted to the linear sections of data. **(b)** Gradients (vertical axis) of straight line fits in (a), as a function of s (horizontal axis) and S (see legend).

'Softer' conjunctions which have lots of inputs but require fewer of them to be simultaneously activated are more likely to continue operating even in the absence of some of the spikes they expect to be triggered by. So even when some spikes are missing completely because there were not enough conjunction detectors to encode them, memory retrieval may be able to continue.

Once again, we see that for lower values of N, some of the curves become affected by the proliferation failure mode and 'drop off' the linear trend. High values of S and low values of s are the worst affected for the same reason that they alleviate the extinction problem: 'softer' conjunction detectors are more easily triggered accidentally by other memories and therefore more likely to result in a proliferation of spurious spikes.

4.4 Conclusion

By simulation with randomly-generated training memories, and theoretical analysis, we have obtained capacity results for the Concurrent Recall Network under both the unlimited training rule and the more biologically plausible limited training rule. We have also examined two failure modes of the network which limit its capacity: proliferation and extinction.

Chapter 5

Line Attractors with the Concurrent Recall Network

5.1 Introduction

In chapter 3 we introduced an associative memory model which recalls stored spatiotemporal spike sequences using spiking neurons, conjunction detectors and delay lines. The network can stably regenerate any of the stored memories when stimulated with a noisy or incomplete version of the memory. We can formulate such a system as an *attractor network* [32] by considering each stored memory to be a point in a high-dimensional space containing all spatiotemporal spike sequences. When we stimulate the network with a noisy version of a memory, we are initialising it at a point in this space close to one of its stored memories. The dynamics of the network should then tend to move the state closer to the stored memory, until it reaches a stable state. Each stored memory in a non-overloaded network is an individual *attractor state* or *fixed point* in the space of memories. We can visualise *point attractors* as a local minimum in an energy landscape[1] into which the state of the network will 'roll' if placed nearby (figure 5.1(a)).

The attractor concept can be extended from zero-dimensional point attractors to one-dimensional *continuous attractors* [75]. If we can create a valley in the energy landscape, and the bottom of the valley does not slope, then we have created a *line attractor* (figure 5.1(b)). When initialised at a state anywhere along the line attractor, the network will remain in that state, since the energy landscape is flat there. If initialised a short distance from the attractor, it would descend to the bottom of the valley and come to a stable location somewhere along the line attractor. A line attractor can be thought of as a continuous trail of fixed points lying along the bottom of the valley.

A network with continuous attractors can store not just individual memories (point attractors in the high-dimensional space of all possible memories), but *continua* of memories.

[1]The energy landscape is a helpful introduction to the attractor concept. However, we note that it is not possible to define an energy function for all attractor networks.

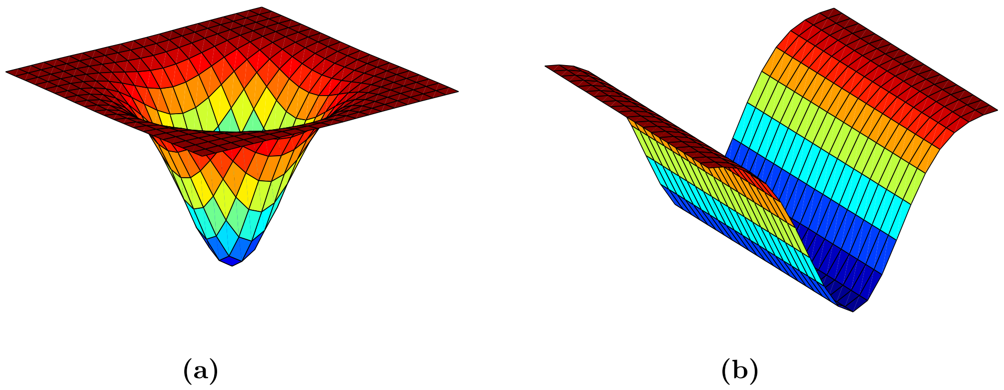


Figure 5.1: Illustration of the energy landscape for **(a)** a point attractor, and **(b)** a line attractor. A two-dimensional system is shown; in a typical attractor network the number of dimensions will be equal to the number of neurons in the network.

Continuous attractors have several uses. For example, an organism could use the state of a line attractor to internally represent its location along a one-dimensional route, the position of its limbs, head or eyes [73, 74], or for short-term memory of any continuous (real-valued) variable. Similarly, a two-dimensional *attractor map* can be used as a 'cognitive map' of the organism's physical location in two-dimensional space [70, 85, 63].

Most existing models of neural line attractors use the firing rate of one or more neurons to represent the continuous value being stored [16, 73]. In this chapter we investigate line attractors built using the Concurrent Recall Network of the previous two chapters. Spike timing, not firing rate, is used to store a representation of a continuous variable.

5.2 Mapping position to a spatiotemporal spike sequence

To build a line attractor based on the Concurrent Recall Network, we need a way of mapping a position (the distance along the line) onto a spatiotemporal spike sequence, thereby defining a one-dimensional line through the high-dimensional space of all possible spatiotemporal spike sequences. For this purpose, we use the following function:

$$\phi_i(x) = \alpha_i + \beta_i x \mod 2\pi. \tag{5.1}$$

ϕ_i is the phase within a cycle of fixed period at which the ith neuron fires, and x is a real number giving the distance along the line attractor from an arbitrary origin lying on the line. Each neuron in the resulting spike sequence fires exactly once per cycle. Thus, α_i determines the phase of neuron i in the firing pattern corresponding to the position $x = 0$, and β_i is the rate at which the same neuron's firing phase changes as we move along the line. Figure 5.2 shows the firing times of ten neurons as a function of x, and some corresponding spike sequences.

In neuronal hardware, the mapping of real-valued stimuli onto firing phase can be accom-

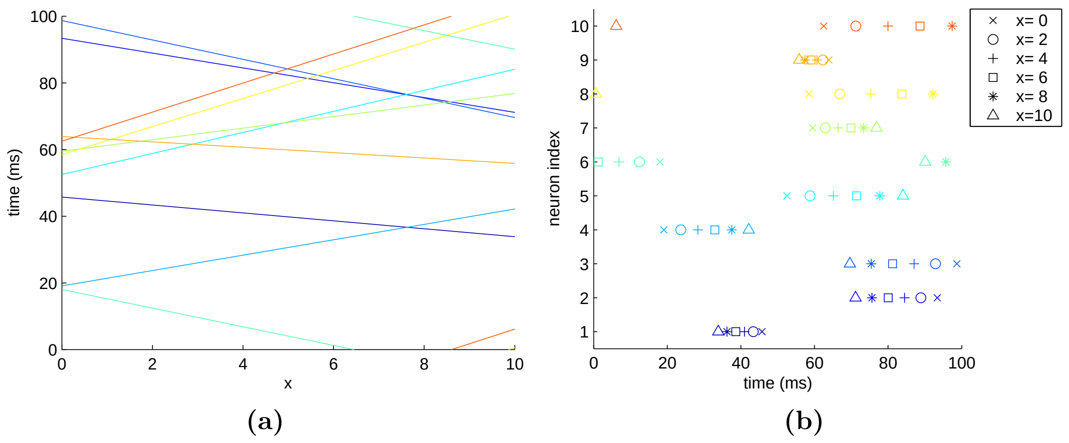


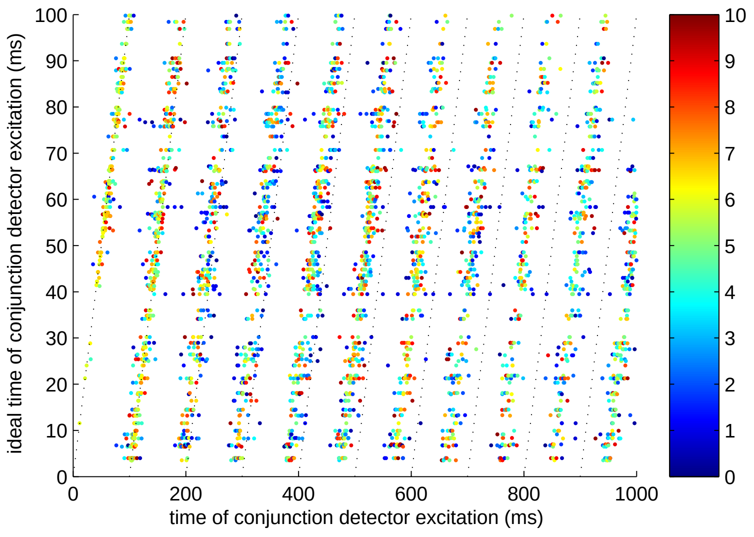
Figure 5.2: Mapping of a real value onto a spatiotemporal spike sequence. **(a)** Firing times (i.e. $\phi_i(x) \cdot \frac{T}{2\pi}$ where T is the period, 100ms), plotted against x, the position along the attractor, for ten neurons. Each intercept α_i was randomly chosen from a uniform distribution between 0 and 2π, and each β_i uniformly between $\pm\pi/10$, corresponding to a slope of up to $\pm$5ms per unit distance. **(b)** Spike rastergram showing, overlaid, spatiotemporal spike patterns for six values of x shown in the legend. Line colours in (a) correspond to the point colours in (b). The slope of each line in (a) determines the amount (and direction) by which the corresponding neuron's spiking time in (b) shifts as x is varied.

plished through the use of sub-threshold oscillations in neurons' internal potential [33, 60]. There is evidence of correlation between the phase of firing of 'place cells' in the rat hippocampus relative to a global 7-12Hz 'theta' rhythm and the location of the rat within the cell's place field [78, 64, 17, 67, 25]

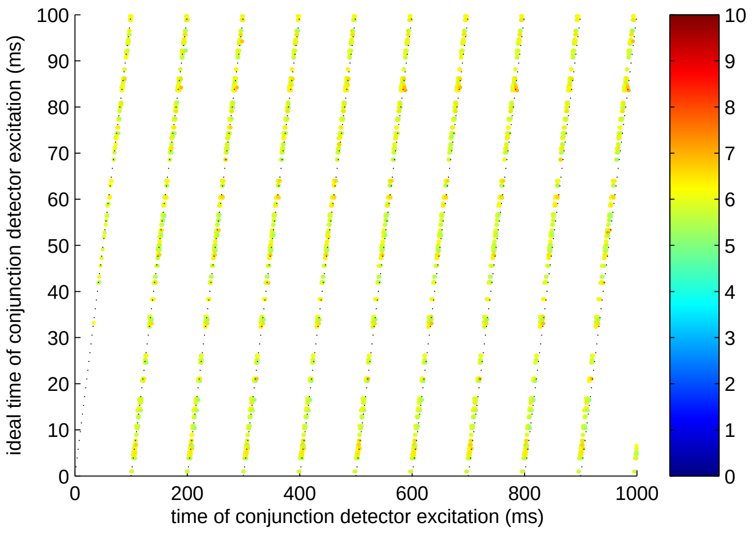
5.3 Storing an entire line attractor

The method chosen to store a line attractor memory in a Concurrent Recall Network is based on the limited training rule. As defined in section 4.3, the limited training rule allocates each conjunction detector in the network the task of reproducing a randomly-chosen spike from all the spikes that the conjunction detector's neuron is responsible for, in all the memories to be learnt.

The learning rule is modified for the storage of a line attractor in the following way. We denote distance along the line attractor by x, with the extent of the line limited by $x = x_{\text{min}}$ and $x = x_{\text{max}}$, and define a mapping from x to a spatiotemporal spike sequence using equation 5.1. For each conjunction detector available in the network, we generate a random position along the desired line attractor, x_{train}, with $x_{\text{min}} \leq x_{\text{train}} \leq x_{\text{max}}$ and use equation 5.1 to generate $\mathcal{M}_{\text{train}}$, the spatiotemporal spike sequence corresponding to that position. The conjunction detector is then given the task of generating its neuron's spike in $\mathcal{M}_{\text{train}}$, using delay lines from any S of the other spikes in $\mathcal{M}_{\text{train}}$. (Recall that S is the number of inputs to each conjunction detector, and s is the number of those which must be concurrently receiving a spike in order to activate the conjunction detector.) The procedure is very similar to that



(a)



(b)

Figure 5.3: Data from simulations of a Concurrent Recall Network line attractor. **(a)** A failed attempt, where the learning rule had no restriction on the β_i values of neurons used as the inputs to a conjunction detector; **(b)** A successful attempt where the learning rule rejected proposed conjunction detectors with input neurons having similar β_i values. In both cases, the network of 100 neurons ($S = s = 3$, $D = 150$, $d = 1$) was trained on a continuous memory lying on a line parameterised by $x = 0 \dots 10$, and was stimulated in the first period (0–100ms) with the spike sequence corresponding to $x = 6$. Each point on the graph indicates the activation of a conjunction detector. The horizontal coordinate of each point on the graph is the time at which the activation happened; the vertical coordinate is the time at which it should ideally have happened, i.e. the time at which the neuron on which the conjunction detector lives should ideally fire according to the spike sequence for $x = 6$. The colour of each point indicates x_{train} for the conjunction detector. In an ideal world, each point would lie on the dotted line (actual activation time equals ideal activation time) and most points would be coloured yellow (only conjunction detectors trained to respond to spike sequences close to $x = 6$ should respond).

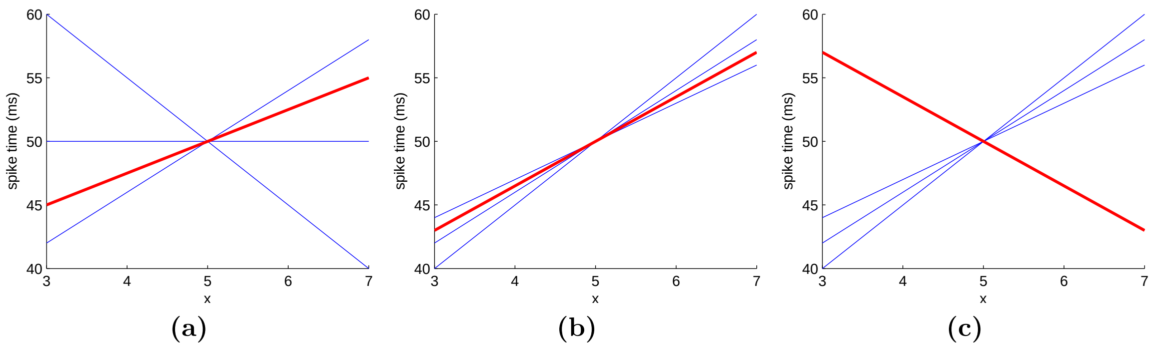


Figure 5.4: Effect of the range of β_i parameters on a conjunction detector's behaviour. The β_i parameters for the three neurons which feed a three-input conjunction detector are shown (as in figure 5.2(a)) by the slope of three blue lines. The lines are shown intersecting at $x = 5$ since the conjunction detector was trained at that position. The slope of the red line shows the β_i parameter for the conjunction detector's own neuron. **(a)** The three input β_i's are dissimilar: a small change in x will result in the input spikes being substantially separated in time. This conjunction detector will only respond to a very small region around $x = 5$. **(b)** The three input β_i's are similar. The input spike times all vary similarly as x changes, so the conjunction detector will still activate over a wide range of x values, with its activation time shifting together with the input spikes. In this case, the red line, showing how the neuron's firing time *should* vary with x, concurs with the input neurons. The conjunction detector will generalise correctly to a wide range of positions x. **(c)** As (b) except that the conjunction detector's own neuron's ideal firing time changes with x differently to the input spikes. This is undesirable: the conjunction detector (and hence the neuron) will respond to a wide range of x values but, rather than correctly tracking the correct spiking time as x varies, the phase of its output will move in the opposite direction to what it should ideally do. Only at $x = 5$ exactly would the output spike be correctly timed.

defined in section 4.3 for storing n individual memories, except that rather than distributing the available conjunction detectors between n discrete memories, we distribute them among the infinitely many memories along the attractor.

The hope is that the network can recall any point along the line using a handful of conjunction detectors which were trained on positions x_{train} lying close to that point. Two nearby points on the attractor would be recalled using overlapping, but not identical, pools of conjunction detectors. Each neuron participates in storing positions spread along the length of the attractor, making this, as in the case of the individual memories used in the previous chapter, a distributed memory.

Initial trials were conducted by training the network in the manner described and then stimulating it with the spike pattern corresponding to a particular position along the line attractor. It was found that the pattern of spiking in the network degenerated, over a few periods, into a pattern that did not correspond well to any position along the attractor, and eventually petered out into nothing. Figure 5.3(a) shows data from a simulation in which 'position coherence' is lost in this way.

A problem found to be inhibiting successful operation was the fact that some conjunction detectors fired, at incorrect times, for a wide range of stimulated positions along the line attractor. The reason for these spurious spikes can be understood in terms of the β_i values of the neurons from which a particular conjunction detector makes connections, as is illustrated in figure 5.4. Since β_i is randomly chosen, a typical conjunction detector which has, for example, three inputs will receive input from neurons with three different β_i values. The neuron on which the conjunction detector lives will also have its own β_i value. If the three input β_i values are significantly different from each other (figure 5.4(a)), then the particular temporal configuration of input spikes that the conjunction detector recognises will occur only in a narrow range of positions along the line attractor: moving away from this position will cause the relative spike times to shift by different amounts. But if the three input β_i values happen to be similar in value (figure 5.4(b,c)), then moving along the line attractor corresponds to shifting all three spikes by roughly the same amount and in the same direction. There will therefore be a much greater range of positions, x, for which the *relative* timing of the three spikes is sufficiently close to what they are at the conjunction detector's 'target' position to cause the detector to activate. If the conjunction detector's own neuron's β_i value is also similar to the three input β_i's (figure 5.4(b)), then, as x varies, the ideal 'conjunction time' will shift together with the input spikes, and hence together with the actual conjunction time. Such a conjunction detector *generalises* well, since it responds correctly over a wide range of x values. In contrast, if the conjunction detector's own neuron's β_i value is dissimilar to the input β_i values, then the ideal conjunction time does not remain close to the actual conjunction time once x diverges from x_{train}. This situation, illustrated in figure 5.4(c), will disrupt the system by causing misplaced spikes.

To reduce this effect, the training rule was modified such that it only permits conjunctions to be formed whose feeding neurons' β_i values are not all similar to each other.[2] This modification proved to be of great benefit, resulting in a system that correctly recalls any position along the line attractor. Data from an example simulation are shown in figure 5.3(b).

By preventing conjunction detectors that respond to a wide range of x values, the rule described above eliminates the problem case, figure 5.4(c), but also eliminates case (b) which has the merit of generalising well. We believe that a rule which allows case (b), i.e. does allow similar input β_i values if the output β_i value is also similar, should perform better. However, in simulations this was not found to be the case. The reason for this remains an open question.

In this chapter, we do not use a biologically-plausible training rule. One can, however, envisage a Hebbian learning rule which strengthens the importance of candidate conjunction detectors that are consistent with spatiotemporal spike sequences played to the network during a training phase. If, during the training phase, the spikes sequences presented to the network are those corresponding to many different x-values, then we believe that the net-

[2]In the remainder of this chapter, the precise criterion used is as follows. If the proposed inputs to a conjunction detector come from neurons with indices $i_1, i_2 \dots i_S$ then the proposal is only accepted if $|\beta_{i_{j+1}} - \beta_{i_j}| \geq \beta_{\text{crit}} \;\forall\; j \in \{1 \dots (S-1)\}$. The value of β_{crit} used is 0.025π, one quarter of the range from which β_i values are drawn.

work would tend to learn conjunction detectors involving similar β_i values on their input and output neurons, i.e. corresponding to the example in figure 5.4(b). For the same reason that conjunction detectors with similar β_i values generalise well when used generatively, the same conjunction detectors should be the ones which are consistent with the training data most frequently, and hence will be boosted by a Hebbian-type rule. This suggests a possible biologically-plausible mechanism for selection of β_i values.

5.4 Position readout

In order to measure the dynamics of the line attractor, and in order for its line attractor properties to be useful, we need a way of determining to which position along the attractor (if any) the network's spiking activity corresponds. We have taken two approaches to this. Firstly, we need a quantitative measure of the nearest point on the line attractor to the network's state and how distant it is from the line. Secondly, we might like a biologically plausible 'neural' readout of the network state without resorting to external analysis of the spiking activity. A neural readout of this type, based on the grandmother neurons of section 3.2.8, is introduced in section 5.8.

Our quantitative measure of 'distance' (in the high-dimensional space of possible spike sequences) between the actual pattern of spikes in a particular 100ms period and the ideal spike pattern corresponding to a particular position along the line attractor is

$$d = \sum_i d_i \tag{5.2}$$

where

$$d_i = \begin{cases} 1 - \cos\left((t_i^{(\text{target})} - t_i^{(\text{actual})})\frac{2\pi}{T}\right) & \text{if } t_i \text{ exists} \\ 1 & \text{otherwise.} \end{cases} \tag{5.3}$$

Here, T is the period, $t_i^{(\text{target})}$ is the time of the spike on neuron i in the target pattern and $t_i^{(\text{actual})}$ is the first firing time of the same neuron in the actual spiking activity (which is limited to a single period in duration). This distance measure is periodic in the difference between the actual and target firing times of each neuron, so that, for example, a spike at $t = 99$ms is considered 'close' to a spike at $t = 1$ms if the period is 100ms.

The second case in equation 5.3 means that a missing spike is penalised to the same degree as a spike that is 90 degrees out of phase with its correct time in the target pattern. Since the target pattern is an ideal pattern corresponding to a line attractor position, it is guaranteed to have exactly one spike per neuron. If no spikes are present at all in the actual spiking activity, the distance measure is equal to the number of neurons.

To find the closest line attractor position to a given period of spiking activity, we minimise, with respect to x and Δ_{offset}, the distance (according to the measure defined above) between the actual spiking activity and the spike pattern corresponding to position x, shifted in time by Δ_{offset}. A systematic evaluation over a coarse grid of values of x and Δ_{offset} is performed first, with the lowest distance found used to initialise a numerical optimiser, to avoid falling into local minima.

5.5 On-attractor dynamics

In an ideal line attractor, every point along the attractor is a fixed point of the system's dynamics. In this ideal world, when placed into a state which lies at an arbitrary point on the attractor, the network would remain in exactly that same state indefinitely. For the Concurrent Recall Network, this would correspond to repeating periodically exactly the same spike sequence.

In reality, slow drift along a line attractor is to be expected: not every point along the line will be a perfect fixed point of the system's dynamics. As a result, we expect that when placed in an arbitrary on-attractor state, the system will remain at (or very close to) that state on a short time-scale, but eventually will drift along the attractor until it reaches a true local attractor strong enough to prevent any noise in the system from causing it to escape.

Figure 5.5 shows the behaviour of the Concurrent Recall Network used to generate figure 5.3(b) when stimulated with spike patterns corresponding to eleven different positions along the attractor, $x = 0 \ldots 10$. On short timescales (on the order of one second, or 10 recall periods), the network stays close to the point at which it was initialised, demonstrating that it does indeed behave as a line attractor. On longer timescales, the network state drifts slowly. Rather than performing a random walk, the overlaid multiple trials show that the network's state tends towards recalling spike sequences corresponding to certain values of x — the local attractors described above. This is exactly how we expect a non-ideal line attractor to behave.

In the Concurrent Recall Network line attractor, one factor contributing to small imperfections of the attractor is the distribution of points along the attractor at which conjunction detectors were trained. At any given location along the line, x_{current}, the corresponding spike sequence will activate a pool of conjunction detectors, whose values of x_{train} will be spread a little way either side of x_{current}. If the 'centre of mass' of these points is not exactly at x_{current}, we would expect the state of the network to gradually shift towards the centre of mass. A truly stable fixed point, then, will be one with a higher than average density of conjunction detectors trained around it. This explains the 'repulsion' from the ends of the attractor ($x = 0, 10$) seen in figure 5.5. Since no conjunction detectors were trained on positions $x < 0$, the centre of mass of conjunction detectors triggered by the stimulus pattern at $x = 0$ corresponds to a value of $x > 0$. The state of the network is therefore pulled away from the end of the line.

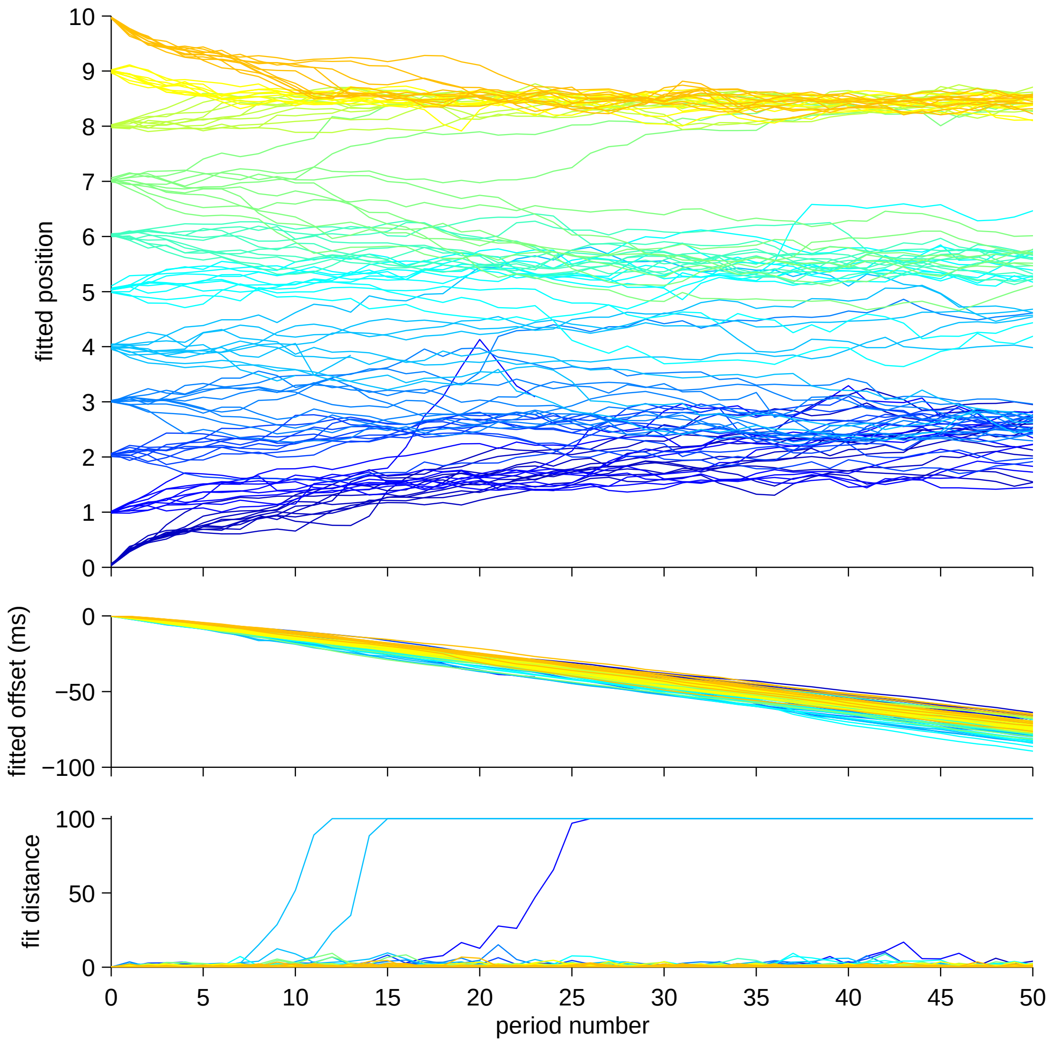


Figure 5.5: Fitted line attractor positions and spike offsets. A Concurrent Recall Network with 100 neurons and $S = s = 3$ was trained on a continuous line memory between $x = 0$ and $x = 10$. The network was stimulated with one period of spike activity corresponding to an integer location along the attractor, and was simulated for 50 periods (5000ms). 10 trials at each of the 11 starting positions are shown. Values of β_i were selected from a uniform distribution between $\pm 0.1\pi$, corresponding to a maximum shift in a spike's time of half a period as x varies from 0 to 10. Temporal jitter with standard deviation 0.5ms was applied to each axon transmission. **Top axes:** the line attractor position closest to the actual spiking activity in each period of simulation. **Middle axes:** The overall temporal offset of the fitted spike position. The offset grows increasingly negative because the recall runs slightly faster than the training patterns. **Bottom axes:** The distance of the fitted spike pattern from the actual spike pattern, showing the goodness of fit. In three out of the 220 trials, activity petered out during the simulation, resulting in a fit distance rising to 100 (the value obtained if no spikes are present).

5.6 Smart neurons: looking into the future

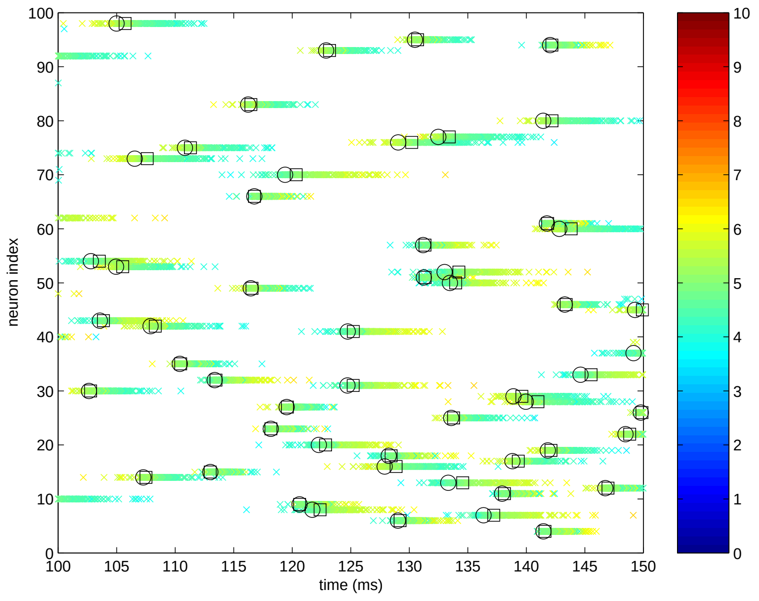
As demonstrated in figure 5.5, the Concurrent Recall Network we defined in chapter 3 is capable of storing continuous line attractor memories after only a slight modification to its training rule.

In the following sections, we look at the behaviour of the line attractor when it is initialised at a location that is not exactly on the line. The network simulated in the previous section was found to not perform particularly well in this situation. Furthermore, the number of conjunction detectors per neuron required hand-tuning to allow the network to recall even unperturbed locations along the line attractor: too few and the recall would peter out into nothing; too many and spikes would proliferate. A desire to overcome these limitations led to the enhancements described in this section.

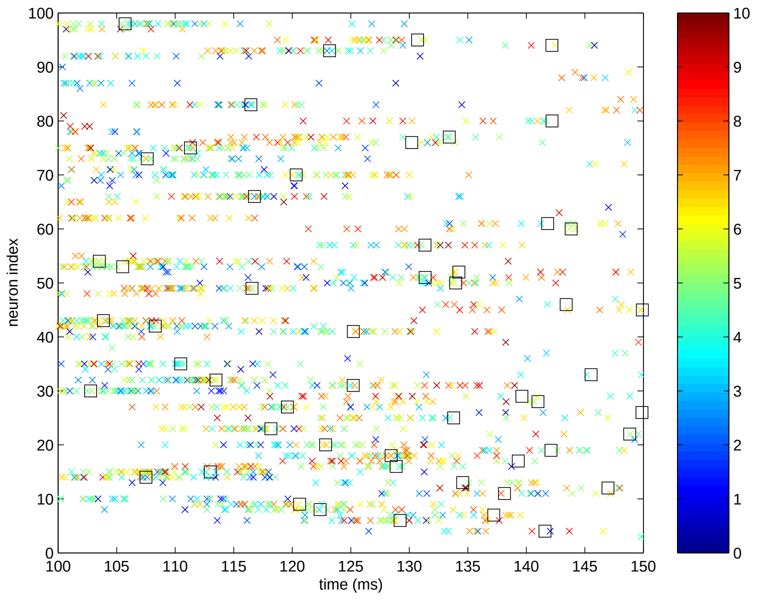
In order to make the Concurrent Recall Network respond better to stimuli that have been perturbed away from the attractor line, ways were sought to 'soften' its response to allow it to cope better with noisy stimuli. In all simulations of the Concurrent Recall Network performed in previous sections of this thesis, the parameter d defined in section 3.2.2, namely the number of conjunction detectors which must activate together in order to cause a neuron to generate a spike, has taken the value of 1. We now increase this value, with the hope that it will result in more resilience against noise without creating a system more prone to generating spurious spikes. Rather than fire as soon as any individual conjunction detector is activated, the neuron can take a 'poll' of all its conjunction detectors and only fire if a certain number are in agreement that a suspicious conjunction of input spikes has occurred. We assign a larger number of conjunction detectors to each neuron (since we need to have a larger population within which to conduct the poll). We also need to consider the distribution over time of conjunction detector activations for a given neuron, in order to understand when the neuron should fire.

Figure 5.6 shows a 50ms extract of the first period of recall following one period of stimulation with a spike pattern corresponding to position $x = 5$. Each neuron had 2000 conjunction detectors. For each conjunction detector that was activated during this section of simulation, the plot shows which neuron it was on (vertical axis), the time of activation (horizontal axis) and the position x_{train} on the line attractor which was originally used to generate its training template (colour). Each neuron tends to experience a burst of conjunction detector activations at around the time it is supposed to fire. In spite of the restrictions on the β_i values of the neurons feeding a conjunction detector (detailed in section 5.3), all conjunction detectors activate for a small range of x values, and their time of activation varies with x in a way that depends on their host neuron's β_i value. This explains why, in figure 5.6(a), each neuron sees a spread of conjunction detector activation times within each burst. For neurons with a positive β_i, conjunction detectors with $x_{\text{train}} < 5$ activate early and those with $x_{\text{train}} > 5$ activate late; for those with negative β_i, the situation is reversed.

Black circles on the figure show the actual firing times of the neurons, which were config-



(a)



(b)

Figure 5.6: Times (horizontal axis) of conjunction detector activations for each neuron (vertical axis) between $t = 100$ms and $t = 150$ms. In the 100ms preceding this period, the network was stimulated with **(a)** the spike pattern corresponding to $x = 5$; **(b)** the spike pattern corresponding to $x = 5$ with each spike independently perturbed according to a Gaussian distribution with standard deviation 8ms. Each neuron has 2000 conjunction detectors and fires if 20 of them are concurrently activated. Colour denotes the line attractor position at which the activating conjunction detector was created during training (x_{train}). Large circles indicate the firing times of neurons; large squares indicate the ideal firing times for $x = 5$. $t_{\text{psp}} = 4$ms; other parameters are as in previous figures in this chapter.

ured with a fixed threshold $d = 20$, meaning that twenty conjunction detectors were required to be concurrently active to trigger a spike. As can be seen from the fact that the actual firing times are relatively close to the ideal firing times (black squares), this parameter setting manages to capture approximately the right moment during each burst of conjunction detector activations. However, when the stimulus pattern is noisy, as in figure 5.6(b), the total number of conjunction detectors successfully activated in each burst is much smaller. In order to enable the neuron to fire at the correct time we would need a high value of d in the low-noise scenario and a low value of d in the high-noise scenario. No single value of d copes well in both scenarios.

To overcome this problem, we use an enhanced neuron mode, which we refer to as the *smart neuron*. Just like the neuron model defined in section 3.2.2, the smart neuron has conjunction detector sub-units. The difference comes in the method used to decide when to fire a spike. Ideally, we would like a neuron to fire in the middle of a burst of conjunction detector activations since, as seen in figure 5.6, some of them will arrive early and some late. Firing in the middle of an extended burst of conjunction detector activations provides a second level of noise clean-up beyond that already performed by the individual conjunction detectors. However, smart neurons are required to detect the middle of the burst without knowing how many conjunction detector activations to expect in each burst, since that will depend on how noisy the spiking activity is. We achieve this by detecting, inside each smart neuron, the moment when the *rate* of activations of its conjunction detectors is highest.

Such a method poses something of a barrier to physical implementation. Even with perfect information about the timing of the conjunction detector activations which have occurred so far, a neuron cannot tell whether the current rate of activations is going to be the maximum of the entire burst. To do so it would need to be able to peek into the future to see whether the rate increases later on in the burst. Fortunately, the design of the Concurrent Recall Network provides an opportunity for a neuron in the network to effectively see into the future, without actually violating causality.

Spikes generated by a neuron in the Concurrent Recall Network propagate away from the neuron along time-delay axons. If we subtract a fixed time, Δ_{smart}, from every axon's delay, but introduce a time delay of Δ_{smart} for spike generation *within* the neuron, then we will not have changed the behaviour of the network other than imposing a minimum effective axon delay of Δ_{smart}. However, we have given the neuron an extra time Δ_{smart} to play with.

In the smart neuron, we use this extra time as follows. The neuron monitors the rate of conjunction detector activations. In the simulations, the rate is defined as the number of conjunction detector activations in the previous millisecond; a softer rate measure would work equally well. If the rate reaches a maximum (in the sense of highest-so-far) value, the neuron initiates the process of firing, which means it will actually generate a spike Δ_{smart} into the future. In the time until the spike actually occurs, the neuron continues to monitor the rate of conjunction detector activations. If it reaches an even higher value, then the time until the planned spike occurs is reset to Δ_{smart}. According to this procedure, the neuron will fire

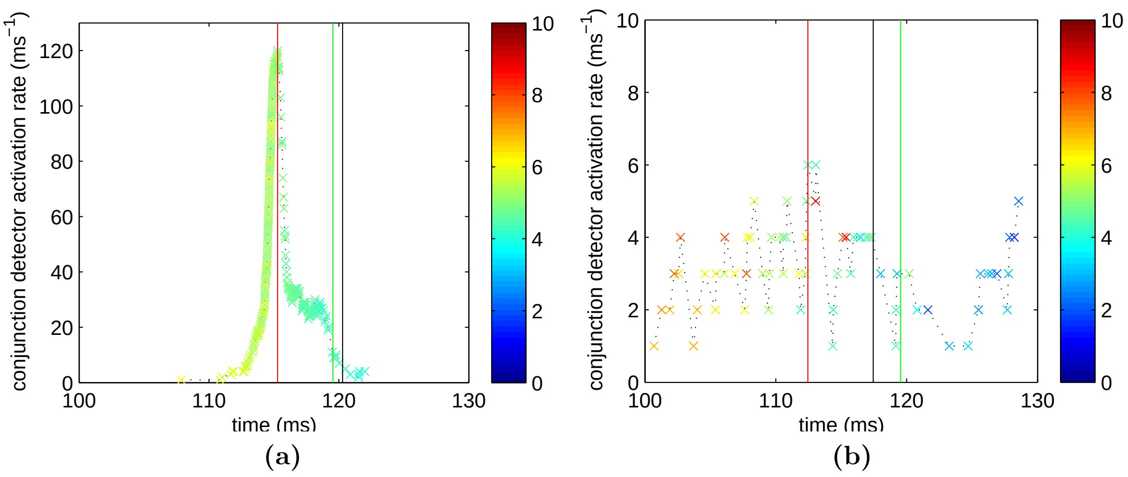
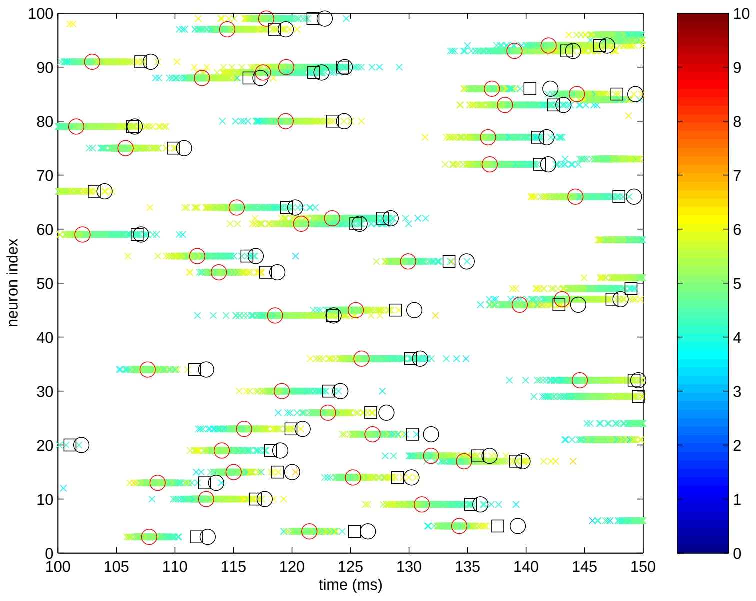


Figure 5.7: Smart neurons. Each plot shows the rate, as a function of time, of conjunction detector activations on a particular neuron with 2000 conjunction detectors. Each point is a new conjunction detector activation; the colour of the point denotes x_{train} for the conjunction detector, as in figure 5.6. The network was stimulated (between 0 and 100ms) with **(a)** the spike sequence corresponding exactly to $x = 5$; **(b)** the $x = 5$ spike sequence with each spike additionally perturbed according to a Gaussian with standard deviation 8ms. Note that the two plots use different scales on the rate axis. The vertical lines show: time at which the neuron detected a maximum conjunction detector activation rate and initiated a delayed firing (red); time of actual firing, 5ms later (black); ideal firing time of neuron according to the $x = 5$ spike sequence (green). Note that in (b) there are several best-so-far conjunction detector activation rates prior to the time denoted by the red line. At the first of these, the neuron initiated a delayed spike to occur 5ms later, but rescheduled it (resetting the delay to 5ms) at each subsequent peak. The red line shows the time of the first such maximum which was not superseded by a better maximum within 5ms.

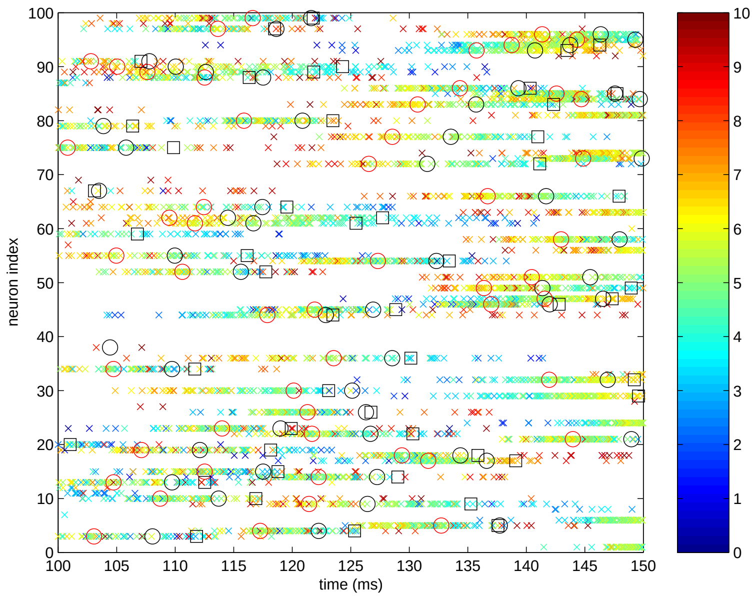
exactly Δ_{smart} after the actual maximum conjunction detector activation rate, unless the rate as a function of time is multi-peaked even on a timescale of Δ_{smart}. Since all axonal delays are reduced by Δ_{smart}, the effective firing time is exactly coincident with the maximum in conjunction detector activation rate. After firing, the neuron resets its record of the 'best-so-far' rate and enters a refractory period during which the conjunction detector activations are ignored.

Figure 5.7 illustrates how an individual smart neuron operates. Figure 5.8 shows the smart neuron rule operating on a complete network, showing how it enables neurons to fire at the correct moment (in the middle of the burst of conjunction detector activations) in both a low-noise (large number of activations) and high-noise (low number of activations) scenario. In the figures, and in the remainder of this chapter, we use $\Delta_{\text{smart}} = 5\text{ms}$. We also impose a minimum conjunction detector activation rate of 5ms^{-1}. Below this rate, a delayed spike is not initiated. This prevents stray 'noise' activations which are not part of coherent burst of activations from causing the neuron to fire.

The smart neuron model described in this section is somewhat more sophisticated than the neuron model defined in section 3.2.2. We do not have a proposed biological implementation



(a) Initialised at $x = 5$



(b) Initialised at $x = 5$ plus σ=8ms perturbations

Figure 5.8: As figure 5.3, except for the following changes. Each neuron fired according to the 'smart neuron' rule defined in section 5.6, with a minimum conjunction detector activation rate of 5ms^{-1} and spiking delay of 5ms. As before, large black circles denote actual neuron firing times. A red circle is placed 5ms before each black circle to show the moment at which the neuron decided to schedule a delayed firing.

of the smart neuron, however we note that the task performed by the smart neuron — finding the maximum rate of input events on a timescale of the order of milliseconds — is a local computation (in space and time). It is also worth recalling that the Concurrent Recall Network is able to operate as a line attractor using only the neuron model of section 3.2.2, as shown by figure 5.5. We have adopted the smart neuron to improve the range of perturbations and noise with which the network is able to cope.

The use of smart neurons achieves a similar effect to that achieved in some models by a dynamically-adjusted inhibition, or 'active threshold' [10]. An active threshold is the approach implicitly adopted in the Synfire chains model [2, 15], in which the number of neurons active is constrained to be the same in each time-step. In the Concurrent Recall Network, an active adjustment of the firing threshold of each neuron to ensure that it fires approximately r times per period, where r depends on the number of memories being recalled, might allow it to pick out the r largest peaks in its input. This could provide an alternative method of scale-independent detection of peaks to that offered by smart neurons.

5.7 Off-attractor dynamics

In section 5.5, we saw how a Concurrent Recall Network line attractor behaves when stimulated (initialised) with a state lying already on the attractor. In this section we investigate stimulating the network with a state which is close to line attractor.

Figure 5.9 shows, on the left, the ability of the network to return to the attractor after stimulation with a perturbed version of the spike pattern corresponding to the position $x = 5$ on the attractor. The right-hand panel shows the position on the attractor where the network state ended up. The network is generally able to recover spike patterns up to a certain perturbation strength, corresponding in this case to each spike being displaced by a normal distribution with standard deviation of around 15ms (as before, the period is 100ms). As expected, simulations in which the network is started further from the attractor tend to end up at a state on the attractor further from $x = 5$.

5.8 Driven movement along the attractor

As discussed in section 5.1, a line attractor can be used to store a real-valued variable such as an organism's position along a one-dimensional route, or the angle of one of its limb joints. A more sophisticated application of a line attractor is as an *integrator*. For example, while a line attractor can be used to *store* a representation of an organism's (one-dimensional) position in space, more useful than this is the organism's ability to integrate its motion, thereby *computing* its position from a starting position and its time-velocity history. *Neural integrators* are also believed to play a role in motor tasks: integration abilities enable the conversion of an 'extend' or 'contract' signal into the limb's current position, either for the purpose of maintaining a mental record of limb position or for informing the motor system

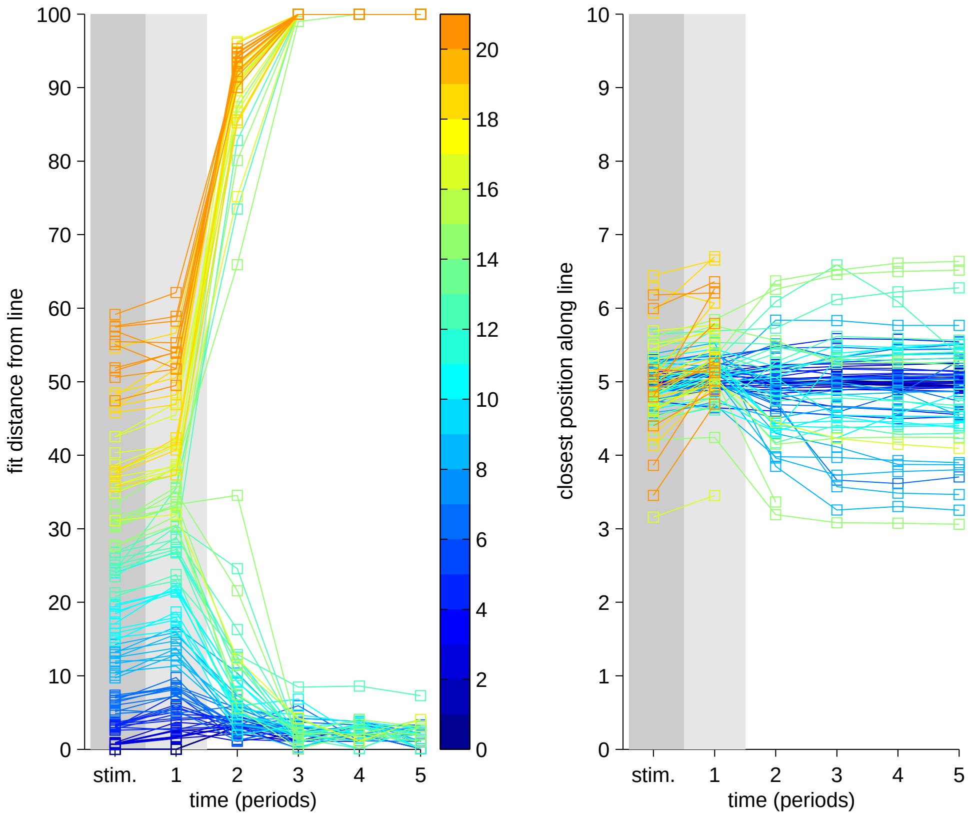


Figure 5.9: Ability of the network to recover from spike sequences which are not on the trained line attractor. The network of 100 'smart' neurons, each with 2000 conjunction detectors ($S = s = 3$), was trained on a line attractor extending over $x = 0 \ldots 10$. For each of the 110 trials shown, the network was stimulated for one period (100ms) with a perturbed spike sequence corresponding to $x = 5$ but with each spike additionally displaced in time according to a Gaussian distribution with zero mean and standard deviation which varied between trials. Line colour depicts the standard deviation used for that trial; the colour legend shows standard deviations in milliseconds. In each trial, the simulation was run for five periods, and a best fit line position (right-hand axes) obtained for the noisy stimulus (labelled *stim.*) and the network activity in each period. Note that the period labelled *1* is the first period of simulation, containing the stimulus plus any additional spurious spikes introduced by the network during the course of the stimulus. The left-hand axes show the corresponding distance of the spiking activity from the best-fit line position, i.e. the distance of the network state from the line, measured according to equation 5.2. Points corresponding to distances of greater than 60 have been omitted from the right-hand axes, since the fitted position becomes meaningless if the fit is very poor. Temporal jitter with standard deviation 0.5ms was added to each axon transmission.

of the desired position [73]. In more general terms, if a line attractor stores a continuum of related memories (such as a commonly used route, or views of a three-dimensional object as a function of viewing angle), then it is useful to be able to mentally move to and fro along the attractor. Likewise, a line attractor could be used to store a temporal sequence; integration of a control signal that causes the recalled location to move along the line at a particular velocity implements the ability to recall temporal sequences at arbitrary speeds and with the ability to 'pause' recall at any point.

In this section, we demonstrate a simple mechanism to implement integration in a Concurrent Recall Network line attractor. Because each neuron's firing time varies linearly with line position (section 5.2), the difference between the spike pattern at $x = x_{\text{current}}$ and $x = x_{\text{current}} + \delta x$ is a small shift $\delta t_i = \beta_i \delta x$ in the firing time of each neuron. To move along the attractor, all that is required is to nudge the firing time of each neuron by δt_i each time it fires. Since δt_i will be negative in 50% of cases, 50% of spikes will require *advancing* in time. Delays and advances of spikes are trivial to implement in the smart neuron model discussed in section 5.6. Since each neuron has an internal delay of 5ms before a planned spike is actually emitted, advancing or retarding the spike by δt_i is simply a case of adding δt_i to this internal delay. In neurobiology, this could be performed by having a global electrical or chemical signal per line attractor. The efficacy of this global 'velocity control' signal would require modification at each neuron according to the neuron's value of β_i. If more than one line attractor were stored in the same set of neurons, then several velocity control signals would be required, each modified by the appropriate β_i for the line attractor.

Figure 5.10 demonstrates the mechanism described above for motion along a single line attractor. A line attractor is initialised at the spike pattern corresponding to $x = 2$. After four periods of recall, the velocity control signal is set to a constant, positive value, by perturbing each neuron's Δ_{smart} by a value proportional to its β_i (the maximum magnitude of this perturbation was 1.25ms). Consequently, the spike pattern of the network moves steadily along the attractor. Later in the simulation, the direction of travel is reversed.

The white crosses in the top axes of figure 5.10 are produced by the same fitting mechanism used in figures 5.5 and 5.9. The figure also introduces two other position read-out mechanisms. In the top axes, the colour of each block shows the distance (according to the measure defined in section 5.4) of the spiking activity in each 100ms period of simulation from the ideal spike patterns corresponding to integer positions $x = 0 \ldots 10$ along the attractor. The bottom set of axes gives the output of a neural readout mechanism. In section 3.2.8 we used a layer of grandmother neurons which, during training of the network, are given connections that cause them to fire at a particular phase within a particular memory being recalled. Thus, a group of 10 grandmother neurons may be trained to fire in response to each successive 10ms section of a particular memory with a period of 100ms. When the network is recalling that memory (possibly amongst others), we expect the group of grandmother neurons to fire in sequence, repeatedly. Here, we extend this scheme for use with a line attractor by creating grandmother neurons for the spike patterns corresponding to the eleven evenly-spaced attractor positions

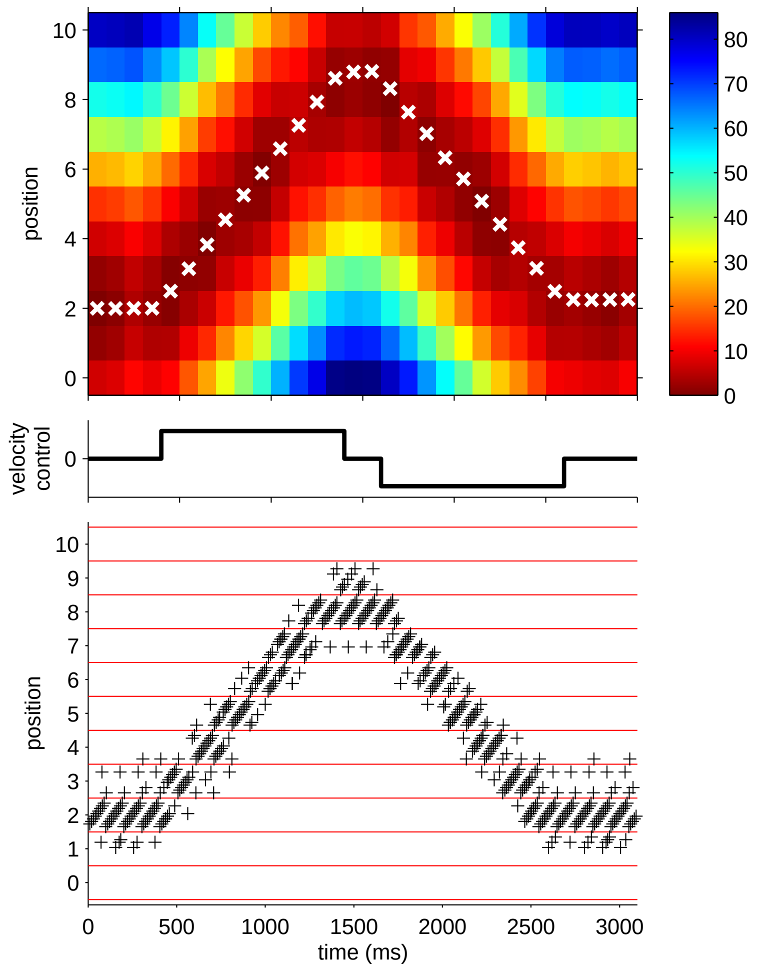


Figure 5.10: Motion along a line attractor, using a network with the same parameters as that of figure 5.9. The network was stimulated with the spike pattern corresponding to $x = 2$ in the first 100ms period, and was left to run unstimulated thereafter. **Top axes:** State of the network as a function of time. Colour denotes the 'distance' of the network state from line attractor positions $x = 0 \ldots 10$ (vertical axis) in each period. White crosses denote the best-fit position. **Middle axes:** Requested velocity along line attractor (arbitrary units) as a function of time. Each bump has a duration of 10 periods. **Bottom axes:** Grandmother neurons' output. Red lines separate groups of grandmother neurons trained to recognise patterns corresponding to 11 positions along the attractor, $x = 0 \ldots 10$. Within each group there are 10 grandmother neurons, trained so as to fire in sequence.

$x = 0 \ldots 10$. Tolerance in their conjunction detectors (originating from the finite post-synaptic potential duration) allows the grandmother neurons to respond to nearby spike patterns. As the state of the network migrates along the line attractor, we see the activity move from one group of grandmother neurons to another. Grandmother neurons of this type are one possible neural read-out mechanism.

The δt_i values applied in figure 5.10 correspond to $\delta x = 0.25$. We might therefore expect the attractor state to move by this amount in each period. In fact, the velocity of motion along the attractor is 2-3 times greater than that. This is explained by the fact that spikes which have been shifted by their δt_i activate conjunction detectors corresponding to position $x_{\text{current}} + \delta x$, i.e. already shifted by δx. The resulting spike from the conjunction detector's host neuron will then be further shifted. The shifts by δx will therefore accumulate at a rate greater than one per period.

When the integration 'control signal' is zero ($\delta t_i = 0$), the integrator becomes an ordinary line attractor and will behave as in figure 5.5, namely it will show a slow drift along the attractor. Drift in neural integrators is an established phenomenon, particularly well studied in the case of the oculomotor neural integrator, a neural circuit which controls eye position and is believed to be constructed from a firing-rate based network [57, 76, 73]. Unsurprisingly, visual feedback has been shown in humans and goldfish to reduce the rate at which eye position drifts [57, 31, 14, 79]. The design of the Concurrent Recall Network-based integrator suggests a particularly straightforward mechanism by which external stimuli, when available, could be used to 'anchor' the attractor state to a particular location. If an external stimulus (which varies with x in some way, including only being present at particular values of x) results in spikes with a fixed firing phase, then those spikes can be incorporated into the training of the network. The more of these spikes that are present during recall, the stronger will be the bias pulling the network state towards the position corresponding to the external stimuli spikes. This scheme has the useful property that the more evidence there is from the external world (the more stimulus spikes are generated), the more effect it has. If there is no external evidence, the attractor relies on its own ability to remember the current position.

5.9 Multiple line attractors

Having demonstrated that the Concurrent Recall Network can implement a line attractor, we turn to the question of whether such a network can truly live up to its name by storing not just one but several line attractors and concurrently recalling different states on each.

The scheme described in section 5.2 maps a real value onto a line within a high-dimensional space of possible memories. With two sets of α_i and β_i parameters, we can define two lines in the high dimensional space, which are unlikely to pass near to each other. We can therefore train a Concurrent Recall Network on two line attractors by distributing the available conjunction detectors along both lines. In the same way that the Concurrent Recall Network trained on individual memories can recall more than one memory simultaneously by outputting a superposition of the two spike patterns, so the network should be able to output a superposition of the spike patterns corresponding to position $x^{(1)}$ on one line attractor and position $x^{(2)}$ on the other. Ideally, each neuron will fire twice in each period, once with phase $\phi_i^{(1)}(x) = \alpha_i^{(1)} + \beta_i^{(1)} x^{(1)}$ and once with phase $\phi_i^{(2)}(x) = \alpha_i^{(2)} + \beta_i^{(2)} x^{(2)}$.

Less obvious is whether the technique used in the previous section to 'drive' the network along an attractor can be be applied in the multi-attractor case. Since in this situation each neuron fires multiple times per cycle, the technique of advancing or retarding each neuron's spikes will alter the timing of spikes belonging to all attractors being recalled. Even if it were desirable to move along all attractors at the same velocity, this would not be the way to achieve it since a neuron has a different value of β_i, and hence δt_i, for each stored attractor.

Fortunately, the natural ability of the Concurrent Recall Network to clean up noisy spike patterns rescues us from this problem. Suppose two attractors are stored, with β parameters $\beta_i^{(1)}$ and $\beta_i^{(2)}$ $(i = 1 \ldots N)$. The network is concurrently recalling position $x^{(1)} = x^{(1)}_{\text{current}}$ on the first attractor and $x^{(2)} = x^{(2)}_{\text{current}}$ on the second attractor. If we advance or retard all spikes by $\delta t_i \propto \beta_i^{(1)}$, then the spikes corresponding to attractor 1 will be moved into a pattern corresponding to position $x_1 = x_{\text{current}} + \delta x$. The spikes corresponding to attractor 2 will also be shifted by the same δt_i. But to the second attractor, these small perturbations are not correlated to $\beta_i^{(2)}$ and simply perturb the state slightly off the attractor in a random direction. Assuming the δt_i are not too large, the network will clean up the noise and maintain the position on attractor 2 at $x = x^{(2)}_{\text{current}}$.

Figure 5.11 demonstrates that a line attractor will indeed ignore δt_i's (or components thereof) which are not the appropriate ones for the set of β_i values corresponding the attractor. Two line attractors, both covering 10 units of length, have been stored in exactly the same network used in previous experiments detailed in this chapter. The network is stimulated with a superposition of two spike patterns, corresponding to one position on each attractor. During the simulation, the δt_i applied to each neuron is the sum of a δt_i for the first attractor and a δt_i for the second attractor: $\delta t_i = v^{(1)}\beta^{(1)} + v^{(2)}\beta^{(2)}$, where $v^{(1)}$ and $v^{(2)}$ are the 'velocity control' signals we wish to apply to the state of two attractors. The time-course of $v^{(1)}$ and $v^{(2)}$ are shown in the middle axes of the plot. The velocity schedule chosen demonstrates simultaneous movement of both attractors independently, as well as motion at different speeds by using different magnitudes of $v^{(1)}$ and $v^{(2)}$.

The upper pair of axes show the distance[3] of the spiking activity in each period from 11 positions along each attractor, and two sets of grandmother neurons are used as a neural readout.

[3]The distance measure defined by equation 5.2 was modified to allow for concurrent recall of multiple positions. Instead of comparing the *first* spike produced by a neuron with the ideal firing time of that neuron, we take the *closest* spike to the ideal firing time. This biases the distance measure towards low values if there are lots of spikes present. However, the grandmother neurons verify that the network is performing as desired.

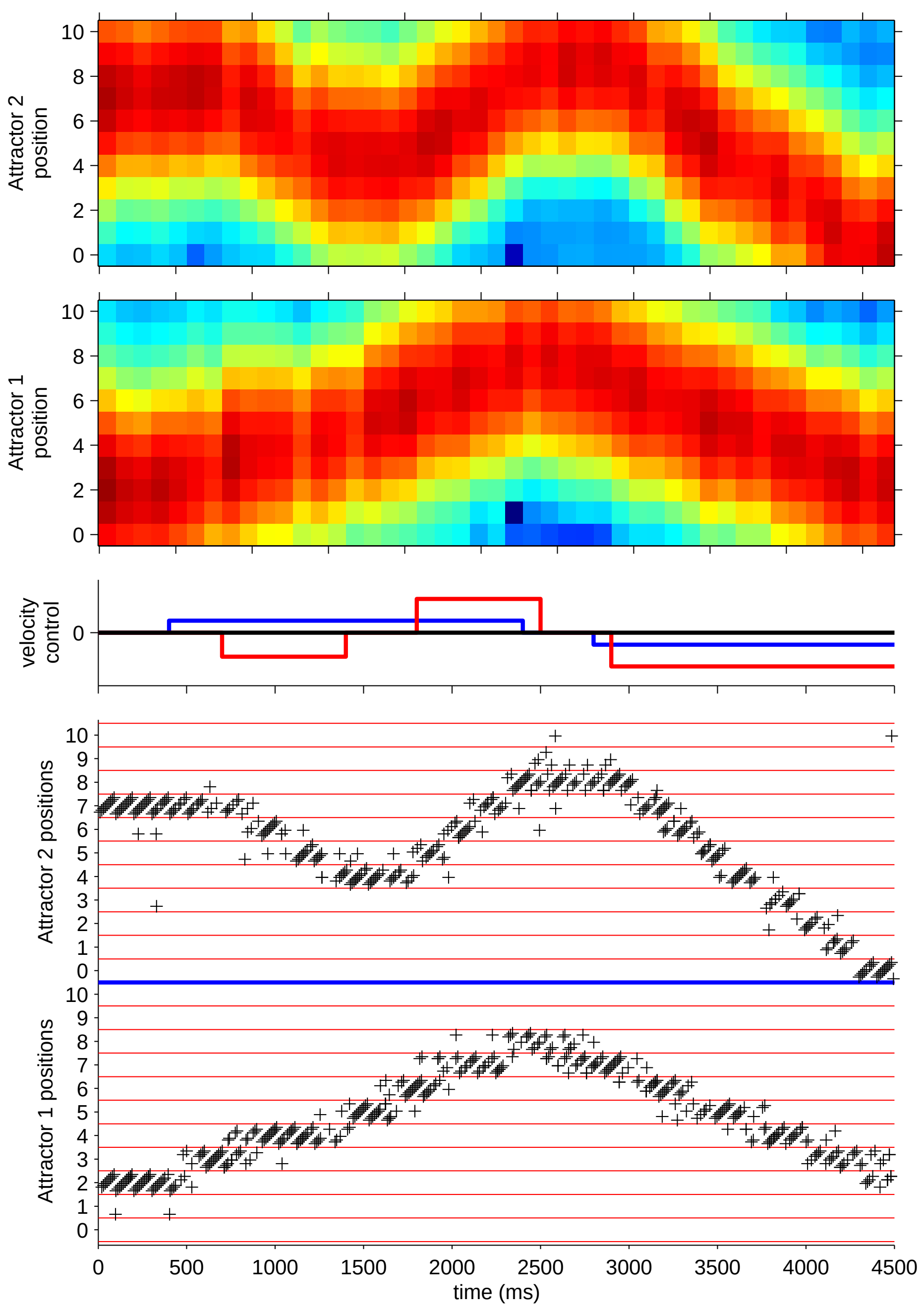


Figure 5.11: Concurrent recall of two attractors demonstrating independent, simultaneous motion. An identical network to that used in figure 5.10 was trained on two line attractors, parameterised by positions $x^{(1)}$ and $x^{(2)}$ (both taking values between 0 and 10). The network was stimulated for one 100ms period with the superposition of the spike sequences corresponding to $x^{(1)} = 2$ and $x^{(2)} = 7$. The top pair of axes shows a distance measure from positions along each attractor; the bottom pair of axes show the output of grandmother neurons trained on positions along each attractor. The middle axes show the motion schedule. The adjustment made to each neuron's firing delay was $\delta t_i(t) = v^{(1)}(t)\beta_i^{(1)} + v^{(2)}(t)\beta_i^{(2)}$. The blue curve is proportional to $v^{(1)}(t)$ and the red proportional to $v^{(2)}(t)$.

5.10 Conclusion

The Concurrent Recall Network defined in chapter 3 can be easily extended from an associative memory of multiple point attractors into an memory containing a line attractor. This transformation was achieved through a simple linear mapping of a real value onto the phases, with respect to an overall 100ms oscillation, of all the neurons in the network.

The capability of the line attractor memory was further enhanced by introducing a more sophisticated neuron model which detects the moment of the maximum rate of activations of a neuron's conjunction detectors. The delay-based architecture of the Concurrent Recall Network allows the neuron, without violating causality, to 'borrow' time to allow it to assess whether a maximum rate of conjunction detector activations has occurred, before firing.

The behaviour of the network when placed in states on or near to the line attractor is as expected for an imperfect line attractor. States on the attractor are stable on a short timescale but, on a longer timescale, drift along the attractor before reaching particularly stable locations. States within a certain perturbation distance of the attractor are attracted onto it; states at a greater distance from the attractor peter-out into inactivity.

A control signal can be applied to the network which causes the state to move along the attractor at an arbitrary velocity. We have therefore demonstrated a neural integrator built from spiking neurons which uses firing phase, rather than firing rate, to represent the stored value.

The Concurrent Recall Network is able to store multiple line attractors in a single population of neurons, each of which participates in all attractors. Positions on each attractor can be independently and concurrently recalled. Furthermore, each attractor can independently act as an integrator, despite sharing a single population.

Chapter 6

Conclusion

6.1 Summary and key insights

We have demonstrated two systems which use spiking neurons to perform recognition and memory tasks.

The first uses synchrony among weakly-coupled integrate-and-fire neurons to perform recognition of spoken words, and could equally well be applied to other temporal stimuli. An observer who recorded only the firing rate of neurons in the *mus silicium* network would observe a curious decaying firing rate in many neurons, but would fail to notice the key principle used by the network. Fractional changes in the time of individual spikes cause many neurons' spikes to fall into synchrony at the crucial moment of recognition. This network exploits the advantages and subtleties that temporal coding can offer over and above a purely rate-coded network. Furthermore, the network is built from biologically plausible hardware and uses a phenomenon widespread in nature — synchrony between weakly-coupled oscillators — to perform computation, thereby offering useful insights into how recognition of temporal patterns could be performed in the brain.

The second system explored (the Concurrent Recall Network) also makes use of the temporal coincidence of spikes, but in a more distributed manner. Whereas in *mus silicium* synchrony is set up among a relatively large number of neurons, in the Concurrent Recall Network delay lines are used to ensure that small numbers of spikes arrive concurrently at the 'coincidence detector' subunits which form part of each neuron. The network therefore makes use of *distributed coincidence detection* to recall memories in the form of spatiotemporal spike sequences. The system behaves as an autoassociative memory: noisy or incomplete stimuli are cleaned up and completed.

A spatiotemporal spike sequence — the spiking activity of a population of neurons over a period of time — is a naturally sparse representation which offers a unique advantage over a more traditional representation such as the firing rate of each neuron. If two firing rates are added, the information about the original firing rates is lost: only their sum is known. But if two spatiotemporal spike sequences are superimposed, the resulting spike sequence contains all the spikes from both the original sequences. The information about which of the two patterns each spike came from is lost, but it is still possible to detect the presence of both the original patterns. This property is exploited by the Concurrent Recall Network. Consequently, it is able to recall more than one stored memory concurrently, something impossible in most memory models. Chapter 4 showed how the capacity of the network, and its failure mode when overloaded, depend on the exact training rule used to store memories.

Having developed a spatiotemporal spike sequence memory, we saw in chapter 5 how it can be transformed into a *line attractor* which can store a continuum of related memories. A simple mapping of a real value (the position along the line attractor) to a spatiotemporal spike sequence is used, in which each neuron's firing phase in a 100ms cycle varies linearly with position along the attractor. Two nearby positions on the attractor correspond to spike sequences in which each neuron fires at nearly the same time. With this mapping defined, the available connections between neurons can be allocated in a way that distributes them along one or more line attractors, allowing the network to recall any point along an attractor. The ability of the network to recall multiple memories concurrently still applies in the line attractor case: simultaneous recall of positions along two line attractors was demonstrated. The performance of the Concurrent Recall Network as a line attractor was enhanced by using 'smart neurons', which exploit the delay-based network architecture to detect the time of peak rate of activations of their coincidence detector subunits without violating causality.

Finally, we saw how line attractors stored in a Concurrent Recall Network offer more than just the ability to recall locations along the attractor. By making a one-off adjustment to the time it takes each neuron to generate a spike, the network can be made to 'drive' along the attractor in a controlled fashion, thereby implementing a neural integrator.

6.2 Future directions

Several avenues for future research are apparent.

6.2.1 Biological plausibility

As stated in the introduction to this thesis, it was not the intention for the models developed to be constrained by known characteristics of real neurons. Nonetheless, having developed the theoretical models, it would be useful to investigate more throughly their possible biological instantiations.

6.2.2 On-line learning

Both of the spiking neural networks presented in this thesis use a training rule which examines the patterns to be learnt and designs the connections in the network accordingly. Preferable to this would be 'on-line' training which could learn stimuli on the fly using biologically plausible, local, rules. Some initial progress in this direction was made in the case of *mus* *silicium* (see appendix A). In the case of the Concurrent Recall Network, the training rules used in this thesis can be recast in a more biologically plausible framework by interpreting the creation of connections with certain delays as the outcome of a process of Hebbian *selection* of connections from a large population of candidate axons with a distribution of available delays. Further research is required to verify that a working on-line learning rule of this form can in fact be constructed.

Another possible area of study is the formation of spontaneous memories when the learning rule is active in the absence of external stimulation. We note two recent studies showing the spontaneous development of repeating spatiotemporal spike sequences in a large network of simulated biologically plausible neurons with spike-timing-dependent synaptic modification and a wide range of axonal propagation delays [43, 42].

6.2.3 Hierarchical implementation

Natural stimuli contain structure and associations at many different levels. To use vision as an example, it seems plausible that when learning to recognise a complex object, such as a face, it should not be treated as a single monolithic entity but as a collection of lower-level features (eyes, nose, mouth and so on). We would therefore like a way for collections of lower-level features to be in turn recognised as higher-level features, in a recursive or hierarchical manner. In the case of the Concurrent Recall Network, we believe this ability would follow naturally from a Hebbian delay selection learning rule of the kind alluded to above. Further research in this area would clearly be of benefit.

The task of understanding the brain is one of the greatest challenges facing science today. We hope the research described in this thesis has made a small contribution to this exciting task.

Appendix A

On-line Learning in mus silicium

We briefly describe here a graphical user interface (GUI) to a simulation of the *mus silicium* network described in chapter 2 which we developed for the purpose of researching an on-line training rule for the network.

The training rule used for *mus silicium* in chapter 2 is a process carried out before simulation of the network begins which, given the training data, designs connections in the network appropriately. The aim here is to investigate a learning rule that is part of the network itself and that would allow new memories to be learnt 'online' simply by presenting them to the network.

Figure A.1 is a screenshot of a GUI front-end to an underlying simulation of a simplified version of *mus silicium* with 25 pairs of W-neurons, 5 event types and 5 possible values of t_{decay}. The GUI was used to experiment with possible learning rules.

The learning rule under investigation in the screenshot has two main features. It uses a Hebbian-style rule which increments the strength of the mutual connection between two pairs of W-neurons each time they fire in near-synchrony, thereby increasing the probability that they will also synchronize next time the same (or similar) stimulus is presented. Such a rule depends on pairs of neurons which have similar firing rates occassionally 'accidentally' synchronizing so that the Hebbian rule has the opportunity to increase their mutual connection. In order to increase the probability of such 'accidental' synchronizations, the membrane potential of all neurons in the network is reset at random times during the simulation. Any two neurons which were being driven by similar input currents at the time of one of these resets would subsequently fire in synchrony until their input currents diverged.

An example is visible in the screenshot. The top set of axes shows the connection strength between two particular neurons. The connection strength increases substantially immediately following one of the global resets (vertical red lines) which occurs during a time when the neurons' input currents are similar.

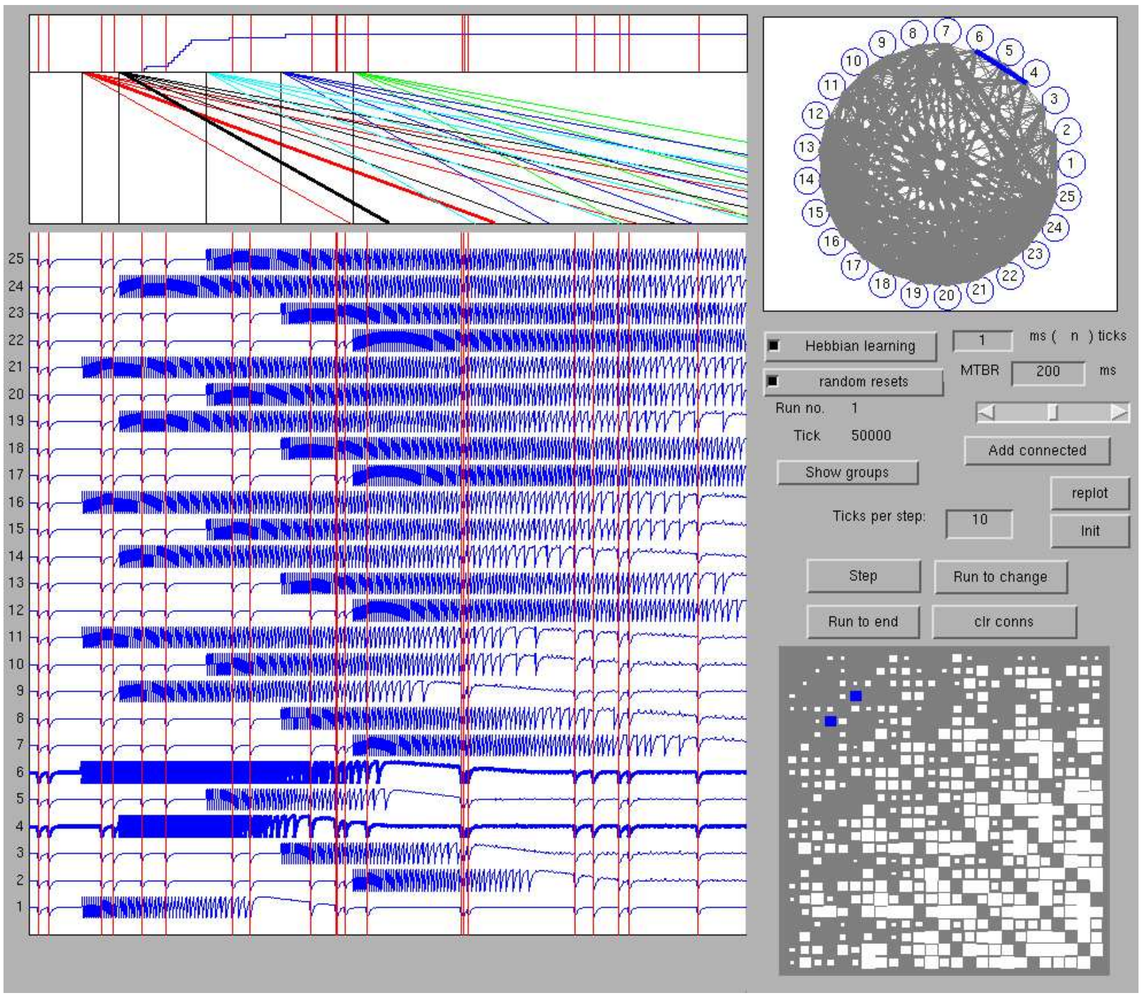


Figure A.1: Screenshot of a graphical user interface developed to investigate an online learning algorithm for *mus silicium*. See text for details of the learning rule. The axes on the left-hand side of the screenshot show, as a function of time, from top to bottom: (a) the strength of the connection between two particular W-neurons in the network; (b) the layer-A output currents driving the 25 W-neurons pairs in the network; (c) the internal membrane potentials of the 25 α-neurons. Thicker lines in (b) and (c) highlight the two neurons whose mutual connection strength is displayed in (a). Vertical red lines indicate times at which the membrane potentials of all neurons were artificially reset. On the right-hand side are diagrams showing the connections between all neurons.

Bibliography

[1] M. Abeles. Role of the cortical neuron – integrator or coincidence detector? *Israel Journal of Medical Sciences*, 18:83–92, 1982.

[2] M. Abeles. *Corticonics: Neural circuits of the cerebral cortex.* Cambridge University Press, 1991.

[3] M. Abeles, H. Bergman, E. Margalit, and E. Vaadia. Spatiotemporal firing patterns in the frontal cortex of behaving monkeys. *Journal of Neurophysiology*, 70(4):1629–38, 1993.

[4] M. Abeles and Y. Prut. Spatio-temporal firing patterns in the frontal cortex of behaving monkeys. *Journal de Physiologie (Paris)*, 90:249–250, 1996.

[5] E. Adrian. The impulses produced by sensory nerve endings: Part I. *Journal of Physiology (London)*, 61:49–72, 1926.

[6] E. Adrian and Y. Zotterman. The impulses produced by sensory nerve endings: Part II: The response of a single end organ. *Journal of Physiology (London)*, 61:151–171, 1926.

[7] E. Adrian and Y. Zotterman. The impulses produced by sensory nerve endings: Part III: Impulses set up by touch and pressure. *Journal of Physiology (London)*, 61:465–483, 1926.

[8] H. Agmon-Snir, C. E. Carr, and J. Rinzel. The role of dendrites in auditory coincidence detection. *Nature*, 393:268–272, 1998.

[9] B. Ahmed, J. C. Anderson, R. J. Douglas, K. A. Martin, and D. Whitteridge. Estimates of the net excitatory currents evoked by visual stimulation of identified neurons in cat visual cortex. *Cerebral Cortex*, 8(5):462–76, 1998.

[10] S. Amari. Characteristics of sparsely encoded associative memory. *Neural Networks*, 2:451–457, 1989.

[11] D. Amit, H. Gutfreund, and H. Sompolinsky. Storing infinite numbers of patterns in a spin glass model of neural networks. *Physical Review Letters*, 55:1530–1533, 1985.

[12] S. N. Baker and R. N. Lemon. Precise spatiotemporal repeating patterns in monkey primary and supplementary motor areas occur at chance levels. *Journal of Neurophysiology*, 84(4):1770–1780, 2000.

[13] G. D. Barreto and A. F. R. Araujo. A taxonomy for spatiotemporal connectionist networks revisited: The unsupervised case. *Neural Computation*, 15:1255–1320, 2003.

[14] W. Becker and H.-M. Klein. Accuracy of saccadic eye movements and maintenance of eccentric eye position in the dark. *Vision Research*, 13:1024–1034, 1973.

[15] E. Bienenstock. A model of neocortex. *Network: Computation in Neural Systems*, 6:179–224, 1995.

[16] C. D. Brody, R. Romo, and A. Kepecs. Basic mechanisms for graded persistent activity: discrete attractors, continuous attractors, and dynamic representations. *Current Opinions in Neurobiology*, 13:204–211, 2003.

[17] N. Burgess, J. O'Keefe, and M. Recce. Using hippocampal 'place cells' for navigation, exploiting phase coding. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 929–936, San Mateo, CA, 1993. Morgan Kaufmann.

[18] C. Carr. Timing mechanisms in the CNS. *Annual Review of Neuroscience*, 16:223–243, 1993.

[19] M. Diesmann, M.-O. Gewaltig, and A. Aertsen. Stable propagation of synchronous spiking in cortical neural networks. *Nature*, 402:529–533, 1999.

[20] R. Doursat. *Contribution à l'étude des représentations dans le système nerveux et dans les réseaux de neurones formels.* PhD thesis, Université Paris VI, 1991.

[21] J. C. Fiala and K. M. Harris. Dendrite structure. In T. McKenna, J. Davis, and S. F. Zornetzer, editors, *Single Neuron Computation*, Neural Nets: Foundations to Applications, pages 1–34. Academic Press, 1992.

[22] W. Gerstner and W. M. Kistler. *Spiking Neuron Models.* Cambridge, 2002.

[23] W. Gerstner, J. van Hemmen, and J. Cowan. What matters in neuronal locking? *Neural Computation*, 8:1653–1676, 1996.

[24] W. Gerstner and J. L. van Hemmen. Associative memory in a network of 'spiking' neurons. *Network*, 3:139–164, 1992.

[25] K. D. Harris, D. A. Henze, H. Hirase, X. Leinekugel, G. Dragoi, A. Czurkó, and G. Buzsáki. Spike train dynamics predicts theta-related phase precession in hippocampal pyramidal cells. *Nature*, 417:738–741, 2002.

[26] D. O. Hebb. *The Organization of Behavior.* Wiley, 1949.

[27] W. Heiligenberg. *Neural Nets in Electric Fish.* MIT Press, 1991.

[28] M. Herrmann, J. A. Hertz, and A. Prügel-Bennett. Analysis of synfire chains. *Network: Computation in Neural Systems*, 6:403–414, 1995.

[29] J. Hertz, A. Krogh, and R. Palmer. *Introduction to the Theory of Neural Computation.* Perseus Publishing, 1991.

[30] A. Herz, B. Sulzer, R. Kühn, and J. L. van Hemmen. Hebbian learning reconsidered: Representation of static and dynamic objects in associative neural nets. *Biological Cybernetics*, 60:457–467, 1989.

[31] K. Hess, H. Reisine, and M. Dursteler. Normal eye drift and saccadic drift correction in darkness. *Neuro-opthalmology*, 5(4):247–252, 1985.

[32] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the USA*, 79:2554–8, 1982.

[33] J. J. Hopfield. Pattern recognition computation using action potential timing for stimulus representation. *Nature*, 376:33–36, 1995.

[34] J. J. Hopfield and C. D. Brody. What is a moment? “Cortical” sensory integration over a brief interval. *Proceedings of the National Academy of Sciences of the USA*, 97(25):13919–13924, 2000.

[35] J. J. Hopfield and C. D. Brody. What is a moment? Transient synchrony as a collective mechanism for spatiotemporal integration. *Proceedings of the National Academy of Sciences of the USA*, 98(3):1282–1287, 2001.

[36] J. J. Hopfield and A. V. M. Herz. Rapid local synchronization of action potentials: Toward computation with coupled integrate-and-fire neurons. *Proceedings of the National Academy of Sciences of the USA*, 92:6655–6662, 1995.

[37] F. C. Hoppensteadt and E. M. Izhikevich. Thalamo-cortical interactions modeled by weakly connected oscillators: Could brain use FM radio principles? *BioSystems*, 48:85–94, 1998.

[38] C. Huygens. *J. Scavants*, XI:79–80, 1665.

[39] C. Huygens. *J. Scavants*, XII:86, 1665.

[40] K. Ikeda. A synfire chain in layered coincidence detectors with random synaptic delays. *Neural Networks*, 16:39–46, 2003.

[41] E. M. Izhikevich. Weakly pulse-coupled oscillators, FM interactions, synchronization, and oscillatory associative memory. *IEEE Transactions on Neural Networks*, 10(3):508–526, 1999.

[42] E. M. Izhikevich. Simple model of spiking network. 2004. Submitted to IEEE Transactions on Neural Networks. Available at `http://www.nsi.edu/users/izhikevich/publications/spnet.htm`.

[43] E. M. Izhikevich, J. A. Gally, and G. M. Edelman. Spike-timing dynamics of neuronal groups. *Cerebral Cortex*, 14(8):933–944, 2004.

[44] L. Jeffress. A place theory of sound localization. *Journal of Comparative Physiology and Psychology*, 41:35–39, 1948.

[45] F. Jelinek. *Statistical Methods for Speech Recognition.* MIT press, Cambridge, MA, 1999.

[46] D. Johnston, J. C. Mageea, C. M. Colbert, and B. R. Christie. Active properties of neuronal dendrites. *Annual Review of Neuroscience*, 19:165–186, 1996.

[47] P. König and A. K. Engel. Correlated firing in sensory-motor systems. *Current Opinion in Neurobiology*, 5:511–519, 1995.

[48] P. König, A. K. Engel, and W. Singer. Integrator or coincidence detector? The role of the cortical neuron revisited. *Trends in Neurosciences*, 19(4):130–137, 1996.

[49] S. C. Kremer. Spatiotemporal connectionist networks: A taxonomy and review. *Neural Computation*, 13:249–306, 2001.

[50] N. Kuwabara and N. Suga. Delay lines and amplitude selectivity are created in subthalamic auditory nuclei: the brachium of the inferior colliculus of the mustached bat. *Journal of Neurophysiology*, 69(5):1713–1724, 1993.

[51] K. H. Lee, K. Chung, J. M. Chung, and R. E. Coggeshall. Correlation of cell body size, axon size, and signal conduction velocity for individually labelled dorsal root ganglion cells in the cat. *Journal of Comparative Neurology*, 243:335–346, 1986.

[52] W. Maass and T. Natschläger. Networks of spiking neurons can emulate arbitrary Hopfield nets in temporal coding. *Network: Computation in Neural Systems*, 8(4):355–372, 1997.

[53] D. MacKay, S. Mahajan, E. Ratzer, J. Miskin, D. Ward, and S. Wills. Time-warp invariant computation with action potentials: Deductions about the Hopfield-Brody mouse. Available at `http://www.inference.phy.cam.ac.uk/mackay/HBMouse/EntryA.html`.

[54] M. Margulis and C.-M. Tang. Temporal integration can readily switch between sublinear and supralinear summation. *Journal of Neurophysiology*, 79(5):2809–2813, 1998.

[55] B. W. Mel. Information processing in dendritic trees. *Neural Computation*, 6:1031–1085, 1994.

[56] B. W. Mel. Why have dendrites? A computational perspective. In G. Stuart, N. Spruston, and M. Häusser, editors, *Dendrites*, pages 271–289. Oxford University Press, 1999.

[57] B. D. Mensh, E. Aksay, D. D. Lee, H. S. Seung, and D. W. Tank. Spontaneous eye movements in goldfish: oculomotor integrator performance, plasticity, and dependence on visual feedback. *Vision Research*, pages 711–726, 2004.

[58] R. Miller. Representation of brief temporal patterns, Hebbian synapses, and the left-hemisphere dominance for phoneme recognition. *Psychobiology*, 15:241–247, 1987.

[59] R. E. Mirollo and S. H. Strogatz. Synchronization of pulse-coupled biological oscillators. *SIAM Journal on Applied Mathematics*, 50(6):1645–1662, 1990.

[60] R. Müller, D. J. C. MacKay, and A. V. M. Herz. Associative memory using action potential timing. In G. Heinz, editor, *Proceedings of BioNet'96: Third Workshop 'Bio-informatics and Pulsepropagating networks'*, 1996.

[61] Mus silicium competition. `http://neuron.princeton.edu/~moment/Organism/`.

[62] Z. Nadasdy. Spike sequences and their consequences. *Journal de Physiologie (Paris)*, 94:505–524, 2000.

[63] J. O'Keefe and L. Nadel. *The Hippocampus as a Cognitive Map*. Oxford University Press, 1978. Available at `http://www.cognitivemap.net`.

[64] J. O'Keefe and M. Recce. Phase relationship between hippocampal place units and the EEG theta rhythm. *Hippocampus*, 3(3):317–330, 1993.

[65] A. Polsky, B. W. Mel, and J. Schiller. Computational subunits in thin dendrites of pyramidal cells. *Nature Neuroscience*, 7:621–627, 2004.

[66] Y. Prut, E. Vaadia, H. Bergman, I. Haalman, H. Slovin, and M. Abeles. Spatiotemporal structure of cortical activity: properties and behavioral relevance. *Journal of Neurophysiology*, 79:2857–2874, 1998.

[67] M. Recce. Encoding information in neuronal activity. In W. Maass and C. M. Bishop, editors, *Pulsed Neural Networks*. MIT Press, 1999.

[68] A. L. Roskies. The binding problem – introduction. *Neuron*, 24:7–9, 1999.

[69] D. E. Rumelhart, G. E. Hinton, and J. L. McClelland. A general framework for parallel distributed processing. In *Parallel distributed processing: explorations in the microstructure of cognition*, volume 1, chapter 2, pages 45–76. MIT Press, 1986.

[70] A. Samsonovich and B. L. McNaughton. Path integration and cognitive mapping in a continuous attractor neural network model. *Journal of Neuroscience*, 17(5):5900–5920, 1997.

[71] I. Segev and M. London. A theoretical view of passive and active dendrites. In G. Stuart, N. Spruston, and M. Häusser, editors, *Dendrites*, pages 205–230. Oxford University Press, 1999.

[72] I. Segev, M. Rapp, Y. Manor, and Y. Yarom. Analog and digital processing in single nerve cells: Dendritic integration and axonal propagation. In T. McKenna, J. Davis, and S. F. Zornetzer, editors, *Single Neuron Computation*, Neural Nets: Foundations to Applications, pages 173–198. Academic Press, 1992.

[73] H. S. Seung. How the brain keeps the eyes still. *PNAS*, 93:13339–13344, 1996.

[74] H. S. Seung. Continuous attractors and oculomotor control. *Neural Networks*, 11:1253–1258, 1998.

[75] H. S. Seung. Learning continuous attractors in recurrent networks. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems 10: Proceedings of the 1997 Conference*, volume 10, pages 654–660, 1998.

[76] H. S. Seung, D. D. Lee, B. Y. Reis, and D. W. Tank. Stability of the memory of eye position in a recurrent network of conductance-based model neurons. *Neuron*, 26:259–271, 2000.

[77] W. Singer. Neuronal synchrony: A versatile code for the definition of relations. *Neuron*, 24:49–65, 1999.

[78] W. E. Skaggs, B. L. McNaughton, M. A. Wilson, and C. A. Barnes. Theta phase precession in hippocampal neuronal populations and the compression of temporal sequences. *Hippocampus*, 6(2):149–172, 1996.

[79] A. A. Skavenski and R. M. Steinman. Control of eye position in the dark. *Vision Research*, 10:193–203, 1970.

[80] N. Spruston, G. Stuart, and M. Häusser. Dendritic integration. In G. Stuart, N. Spruston, and M. Häusser, editors, *Dendrites*, pages 231–270. Oxford University Press, 1999.

[81] S. Strogatz and I. Stewart. Coupled oscillators and biological synchronization. *Scientific American*, 269:102, 1993.

[82] S. H. Strogatz. *Nonlinear Dynamics and Chaos.* Perseus Books, Cambridge, MA, 1994.

[83] A. K. Sturm and P. Konig. Mechanisms to sychronize neuronal activity. *Biological Cybernetics*, 84:153–172, 2001.

[84] D. W. Tank and J. J. Hopfield. Neural computation by concentrating information in time. *PNAS*, 84(7):1896–1900, 1987.

[85] M. Tsodyks and T. Sejnowski. Associative memory and hippocampal place cells. *International Journal of Neural Systems (Supplement)*, 6:81–86, 1995.

[86] A. E. P. Villa, I. V. Tetko, B. Hyland, and A. Najem. Spatiotemporal activity patterns of rat cortical neurons predict responses in a conditioned task. *Proceedings of the National Academy of Sciences*, 96(3):1106–1111, 1999.

[87] C. von der Malsburg. Binding in models of perception and brain function. *Current Opinion in Neurobiology*, 5:520–526, 1995.

[88] D. Wang. Temporal pattern processing. In M. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 1163–1167. MIT Press, Cambridge MA, 2003.

[89] R. Wessel, J. Kristan, William B., and D. Kleinfeld. Supralinear summation of synaptic inputs by an invertebrate neuron: Dendritic gain is mediated by an "inward rectifier" K+ current. *Journal of Neuroscience*, 19(14):5875–5888, 1999.

[90] A. T. Winfree. Biological rhythms and the behavior of populations of coupled oscillators. *Journal of Theoretical Biology*, 16:15–42, 1967.