APPLICATION OF SPARSE DISTRIBUTED MEMORY TO THE INVERTED PENDULUM PROBLEM

A dissertation submitted to The University of Manchester
for the degree of Master of Science
in the Faculty of Engineering and Physical Sciences

2009

Thomas Sharp
School of Computer Science

Contents

Total of ≈18000 words.

List of Tables

List of Figures

Abstract

Sparse Distributed Memory is a neural network data store capable of emulating the animal abilities of recognition and association of similar stimuli. The memory is applied in this research to the task of learning to control an inverted pendulum given no knowledge of the simulation dynamics and feed-back only in the form of punishment upon failure. It was hoped that the application of a biologically plausible neural network to a learning problem common to many animals would provide insight into the operation of the brain and how to imbue electronic machines with certain intelligent characteristics.

A Sparse Distributed Memory variant was constructed from spiking neuron models (a novel aspect of this research) and initialised with random data. The memory was addressed with encodings of the state of the inverted pendulum and prompted to return data that was then decoded into motor actions. Every wrong action was punished with a shock that randomly permuted the data accessed by the last address. It was expected that following a learning period the memory would establish the correct mappings from address to data and also, due its associative abilities, would be able to return appropriate data for previously unseen addresses.

The research had limited success conditional upon artificial restrictions on the learning environment to simplify the problem. Experiments showed that the random permutation of data following a shock severely impaired the distributed storage mechanism of the SDM, limiting its storage capacity and largely negating its associative abilities.

Declaration

No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

Thanks are due to everyone in the School of Computer Science but especially to Steve Furber whose inspiration and support has been invaluable. Romain Brette and Dan Goodman deserve acknowledgement for the development and excellent support of the *Brian* simulator for spiking neural networks used extensively in this research. Finally, this work would have been impossible without the ceaseless love and support of my parents, for which thanks cannot be adequately expressed.

> The whole thinking process is still rather mysterious to us, but I believe that the attempt to make a thinking machine will help us greatly in finding out how we think ourselves.
>
> — Alan Turing, 1951

Chapter 1

Introduction

A pronounced disjunction exists between the abilities of the digital computer and the brain. Whereas organic ‘wetware’ is slow, imprecise and prone to mistakes, electrical engineering has produced machines capable of performing complex calculations at immense speed with insignificant error rates. It may seem, then, that the brain is somewhat stupid and its enormous complexity redundant. However, computer science has yet to replicate high-level functions of the brain such as learning, recognition and association with anywhere near the performance or efficiency found in nature. It may be argued that if the brain so outperforms the computer then the key to imbuing computers with these intelligent characteristics is to design systems in the likeness of the brain. This dissertation documents research into the application of one such system, Sparse Distributed Memory, to the task of learning the motor skills necessary to balance an inverted pendulum.

*

The inverted pendulum is a classic control problem in which an agent must balance a pendulum with a rigid rod by driving the cart upon which it is mounted back and forth. Given the problem in one dimension, knowledge of the physics involved and precise sensors and actuators, programming the cart to control the inverted pendulum is trivial. Without knowledge of the pendulum physics, however, the problem becomes significantly more difficult. Yet this is a similar situation to the one in which bi-pedal animals find themselves when learning to walk and they become quite adept at it.

The inability of a computer to learn to balance an inverted pendulum may be explained according to the teaching approach taken. A natural method of teaching that we might be familiar with—of laying out premises and giving hints until a logical conclusion is reached—fails in the sense that the 'hints' required by a computer quickly grow to become a formal specification of behaviour (a program) which requires no learning on the computer's part whatsoever. An alternative *brutal* approach[1] may be to expose the computer to every possible combination of premises in turn and punish it repeatedly until it derives the correct conclusion for each one by random guess. Imagining that the premises are floating-point numbers representing inverted pendulum positions and velocities and the conclusions are the motor responses necessary to keep the pendulum balanced, the brutal approach fails because of its basis in the intractable task of establishing the correct inference for every combination of premises.

The solution to this problem of intractability may be to establish groupings of similar inverted pendulum states. Then if an inference is correctly guessed for one of the states in a group it can be applied to all of the states in that group, thereby reducing the number of inferences to be guessed at. Grouping might be achieved by reducing the precision with which the inverted pendulum positions and velocities are represented but this would limit the degree of control that can be exerted over the pendulum and does not reflect on how animals demonstrate the ability to switch rapidly between fine and coarse control of a dynamical system, such as when reacting mid-gait to avoid tripping over an unexpected obstacle. Pentti Kanerva's Sparse Distributed Memory is a neural network model of data storage which is interesting here for its ability to group similar stimuli (pendulum states) with similar responses (motor actions). The observation that the structure of Sparse Distributed Memory is similar to those found in the cerebral cortex—an area of the brain implicated in motor control—suggests that it may be a suitable store for the inferences required to learn to balance the inverted pendulum.

*

This dissertation begins with the hypothesis that a brutal learning scheme may be the method by which animals acquire certain gross motor skills, noting that learning to ride a bicycle, for example, requires the repeated punishment of falling off before the correct handling is established. Testing this hypothesis requires a biologically plausible neural network capable of learning, provided by

[1]So named for the combination of brute-force and punishment employed.

Sparse Distributed Memory, and an appropriate abstraction of a control problem, given by an inverted pendulum simulation.

Despite the biological likeness of Sparse Distributed Memory most of the work undertaken to demonstrate its capabilities to date has used highly artificial neuron models which do not reflect the behaviour of neurons in the brain. Recent work in the School of Computer Science at The University of Manchester proposed adaptations to Sparse Distributed Memory—incorporating research by Simon Thorpe into how populations of neurons may encode information—which may make it possible to build an SDM variant from biologically plausible components. The first part of the research presented in this dissertation involves construction of a Sparse Distributed Memory variant from realistic computational neurons, which are examined in chapter 2. Sparse Distributed Memory itself, and the aforementioned work at Manchester, is examined in chapter 3 and chapter 4 documents the process of building the SDM.

The second part of the presented research involves development of a simulation environment for the inverted pendulum and methods to encode the pendulum states as neural inputs and decode the neural outputs as motor actions, all of which is discussed in chapter 5. The research concludes by testing the hypothesis that the brutal teaching approach is viable using Sparse Distributed Memory, as documented in chapter 6. Chapter 7 draws conclusions from the findings of the previous chapters and suggests further work to be conducted in the area.

Chapter 2

Computational Neuron Models

In this chapter the basic structure and function of the biological neuron is outlined and the reasons for creating and selecting abstract computational models are briefly discussed. A number of computational neuron models of varying complexity and biological plausibility are presented and the basic behaviour of each is demonstrated.

2.1 Biological Neural Structures

The human brain consists of more than 10^{11} neurons connected by a number of synapses in the order of 10^{15} [1]. Each component neuron in a network consists essentially of a number of dendrites, a cell body and an axon which forms synapses with the dendrites of a number of other neurons [7] (see figure 2.1). A neuron fires a spike in electrical potential along its axon when the sum of the electrical potentials received in the cell body from the dendrites exceeds some threshold. Signals traverse neural structures, passing from axon to dendrite wherever synapses exist between the two, by means of these spikes in electrical potential. From a particular neuron spikes are practically uniform in size, that is, all stimuli to a neuron which exceed its threshold produce identical spikes [16], a principle described as *all-or-nothing* firing. The bio-chemical mode of spike propagation also means that spikes remain constant in size as they pass along the axon and down any number of branches, unlike a signal in an electrical circuit which diminishes under such conditions.

Two aspects of spiking neural networks are interesting with respect to their

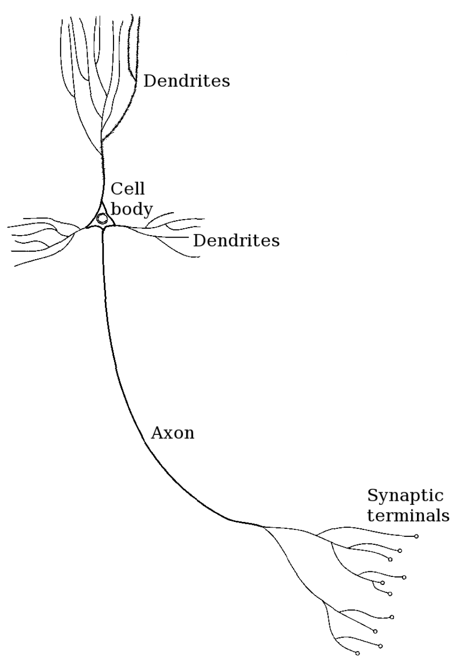

Figure 2.1: A pyramidal neuron. Adapted from [12].

potential information processing capacity. Perhaps obviously, network connectivity forms the basis for this capacity by defining the paths along which signals may travel. However, the notions of connection strengths between neurons (synaptic weights) and the dynamic variation of these strengths (synaptic plasticity) appear to be equally important, having been observed in cultured rat neurons [4] and used as a keystone to many computational models [20, 17, 9]. It has long been conjectured that synaptic plasticity is the mechanism by which animal learning occurs [13] and more recent studies would seem to support this idea [5]. The means by which information may be encoded or stored in neural networks, given synaptic plasticity and the apparently limiting all-or-nothing firing scheme, is among the central research areas of this dissertation. However, before examining this area in detail a number of computational abstractions of the biological neuron should be considered.

2.2 Computational Neurons

When modelling large neural structures abstract neuron models are necessary to manage the computational complexity of the network: despite the continued fulfilment of Moore's law, the most ambitious projects to model biologically plausible neural networks are still limited to simulating around 1% of the neurons in the brain in real-time [19]. Behavioural complexity is a further concern, in that the human designer of a network must be able to specify and understand the behaviour exhibited by neurons, a task that becomes more difficult for programmers as neuron models move from simple logic gates to objects with highly complex biochemical dynamics. The selection of particular neuron models in the following chapters reflects both of these issues.

The following neuron models vary in both computational and behavioural complexity but have in common a lack of spatial characteristics. They are described as point-neuron models, in that membrane potential (activation level) is calculated in the cell body (as the sum of the dendritic inputs) and is assumed to be the same throughout the neuron. This abstracts out some of the most complex biochemical details of the neuron including the propagation delay in the transmission of spikes along axons and dendrites.

2.2.1 The Perceptron

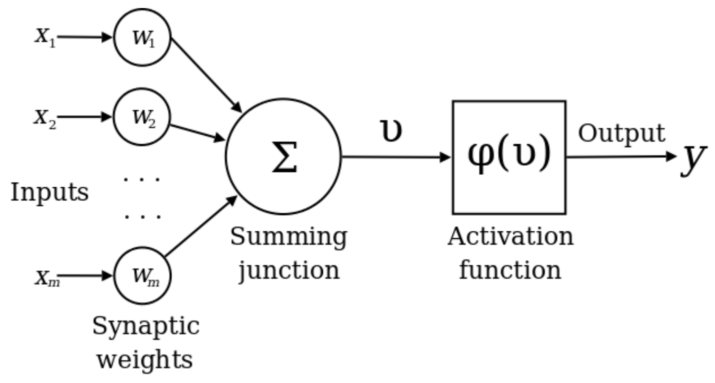

Figure 2.2: The perceptron neuron model.

The simplest abstraction of neuron behaviour is given by the perceptron developed by Rosenblatt [20] and subsequently described by Haykin [12]. Figure 2.2 shows the model structure, which identifies the key components of a biological neuron in that the inputs can be thought of as dendrites, the summing junction as the cell body and the result of the activation function as the firings along the axon. Inputs $x_1, x_2, \cdots, x_m$ are unipolar binary values (just as spikes are

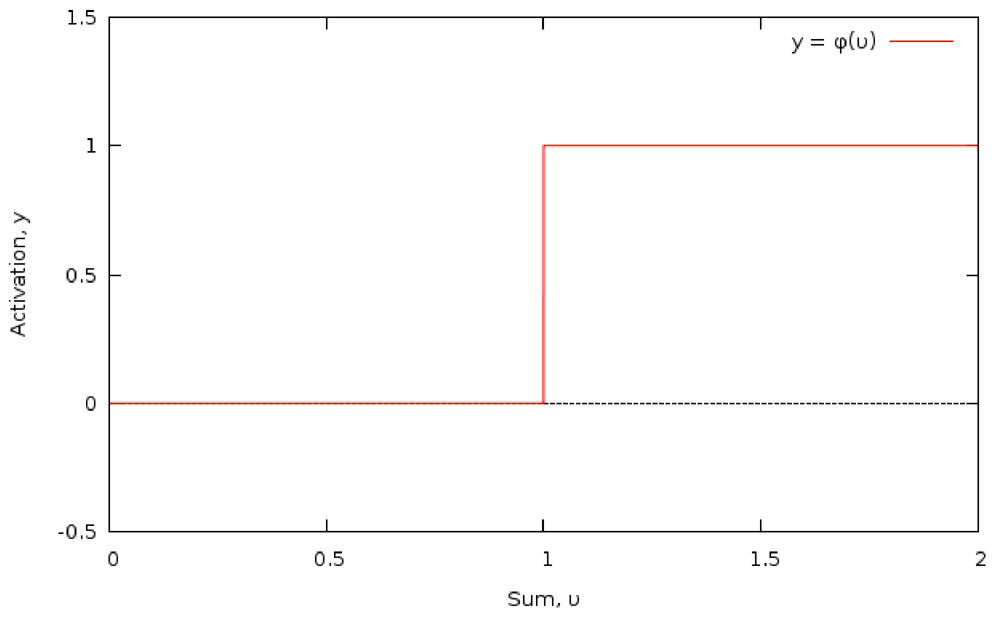

Figure 2.3: Activation of perceptron for $\theta = 1$.

all-or-nothing) and the synaptic weights may be either unipolar or bipolar binary values, depending on the simulation environment.

The perceptron operates in a globally synchronous network and at each time step the summing junction performs the operation

$$v = \sum_{i=1}^{m} w_i x_i \tag{2.1}$$

The activation function, which takes the result of the summing junction as a parameter, is given by a variation of the Heaviside step function which has a variable threshold controlled by θ.[1]

$$\phi(v) = \begin{cases} 1 & \text{if } v \geq \theta \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

Figure 2.3 plots the activation function for a threshold of 1, showing that when the weighted sum of the inputs to the neuron is greater than the threshold the activation is total and invariant (that is, the neuron fires) and otherwise the activation is 0.

[1]The threshold θ and the synaptic weights may from now on be referred to as the neuron parameters.

The perceptron is useful as an abstract demonstration of the operation of a neuron and is sufficiently powerful to implement a variation of Sparse Distributed Memory in a globally clocked network. However, whereas biological neurons demonstrate highly asynchronous behaviour the output of a perceptron at each clock cycle is, like a logic gate, directly dependent on the inputs that are high in that time step. To plausibly model biological neural networks a more complex relationship between dendritic inputs, activation level and axonal output is required.

2.2.2 The Leaky Integrate-and-Fire Model

The Leaky Integrate-and-Fire (LIF) model [10] is identical to the perceptron in structure but differs in function in that it employs a real-valued, rather than binary, activation level that is decoupled from the axonal output. The model is described by the differential equation

$$\dot{A} = \upsilon - A/\tau \tag{2.3}$$

$$\text{if } A \geq \theta \text{ fire and reset } A = 0 \tag{2.4}$$

where A is the activation level, υ is value returned by the summing junction and τ is the leaky time constant which controls the activation decay. Figure 2.4 shows the effect of applying a 5mV input (in place of input from the summing junction) to an LIF neuron with a 30mV threshold and a time constant of 10ms. From application of the voltage at $t = 10$ms the activation level climbs to the threshold, at which point the neuron fires a spike along the axon[2] and the activation level is reset to 0. This process repeats until $t = 75$ms when the input ceases and the activation level 'leaks' in exponential decay. The leaky nature of activation in the LIF model is further demonstrated by figure 2.5 in which an LIF neuron is presented spiking inputs at random times. Note that even in the absence of inputs large enough to cause the neuron to fire spontaneously, firing occurs at approximately $t = 20$ms and $t = 65$ms as a consequence of accumulated successive inputs, behaviour which is impossible to demonstrate with the perceptron.

[2]Spikes in a plot of LIF activation are apparent only by the immediate decay in activation level following that variable reaching the threshold. Spike-like shapes in the activation level of an LIF neuron, such as the one at approximately $t = 75$ms in figure 2.4, should not be confused with output spikes. The morphology of spikes propagated to efferent neurons may be arbitrarily defined but is often given by the Dirac delta function.

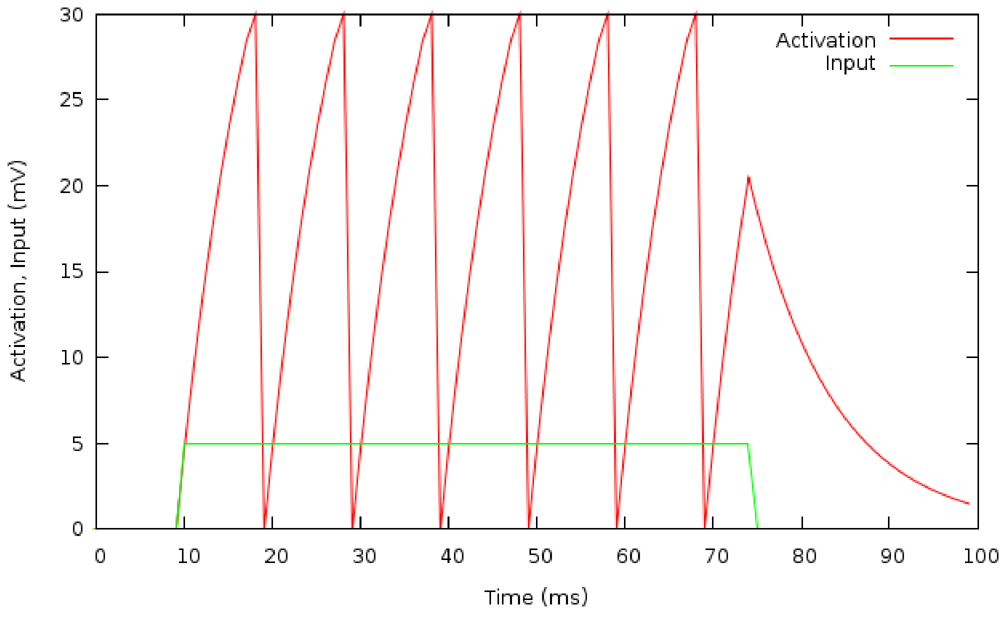

Figure 2.4: Activation level of an LIF neuron for $\theta = 30\text{mV}$ under steady input.

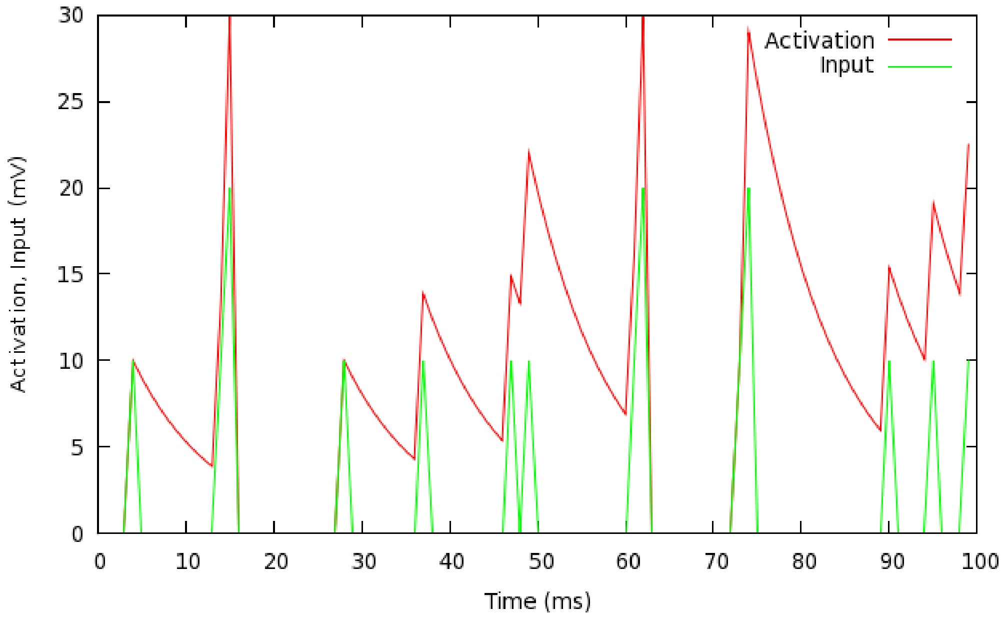

Figure 2.5: Activation level of an LIF neuron for $\theta = 30\text{mV}$ under spiking inputs.

2.2.3 The Rate Driven Leaky Integrate-and-Fire Model

Bose et al. employ a variation of the Leaky Integrate-and-Fire model in which dendritic input drives the first derivative of the activation level rather than the activation level itself in order to transmit coherent bursts of spikes through a multilayer neural networks [6]. The Rate Driven Leaky Integrate-and-Fire model is described by a simple extension to the LIF model

$$\dot{A} = R - A/\tau_A \tag{2.5}$$

$$\dot{R} = \upsilon - R/\tau_R \tag{2.6}$$

$$\text{if } A \geq \theta \text{ fire and reset } A = 0, R = 0 \tag{2.7}$$

and introduces some temporal separation between dendritic input and the peak of the activation level, as shown in figure 2.6. Here, a single input of 5mV is presented at $t = 10$ms which causes a spike in the level of the rate variable R. The rate variable in turn drives the activation level upwards until the former leaks to the point that $-A/\tau_A$ becomes the dominant term and the activation level begins to decay.

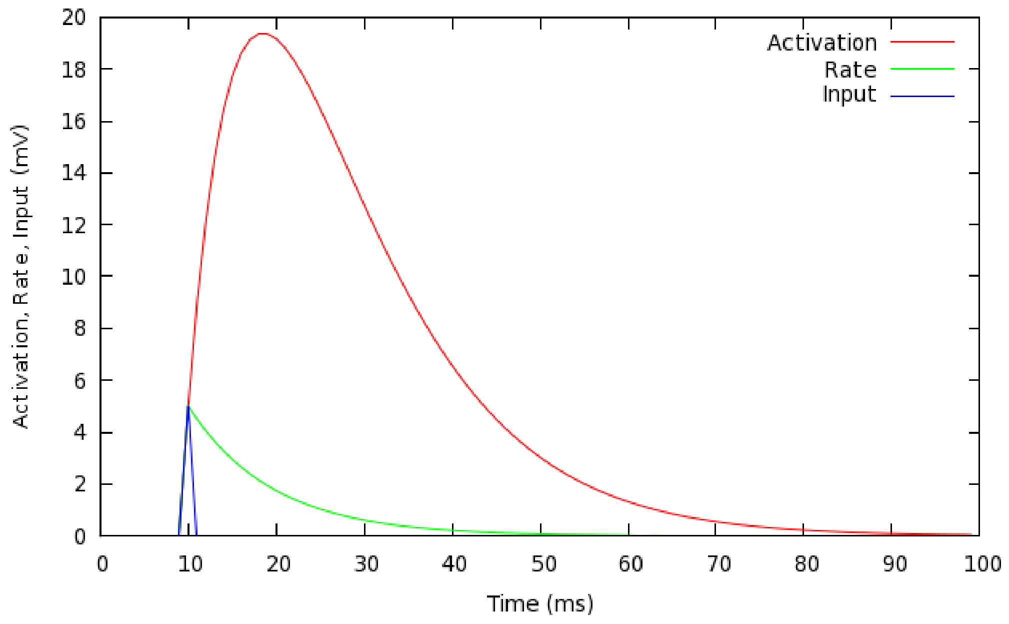

Figure 2.6: Activation level of an RDLIF neuron for $\theta = 30$mV given one input.

2.2.4 The Izhikevich Model

Biological neurons exhibit two characteristics which are absent from both of the above variations of the LIF model. Presented with a constant input voltage, biological neurons fire with increasingly less frequency as they *habituate* to the input. Biological neurons also experience a *refractory period* after firing, during which the neuron is incapable of firing regardless of further inputs [7]. The LIF model may be adapted to emulate these characteristics but the Izhikevich model exhibits both behaviours and is also capable of simulating a range of highly realistic spiking behaviours [14] whilst remaining computationally tractable [15]. The Izhikevich model is described by the coupled differential equations

$$\dot{v} = 0.04v^2 + 5v + 140 - u + I \tag{2.8}$$

$$\dot{u} = a(bv - u) \tag{2.9}$$

$$\text{if } v \geq 30\text{mV, reset } v = c, u = u + d \tag{2.10}$$

where (contrary to the previously described conventions) v is the activation, I is the value returned by the summing junction, u is a leaky *recovery* variable (responsible for habituation and refractory period) and a, b, c and d define the characteristics of the spiking patterns produced by the neuron.

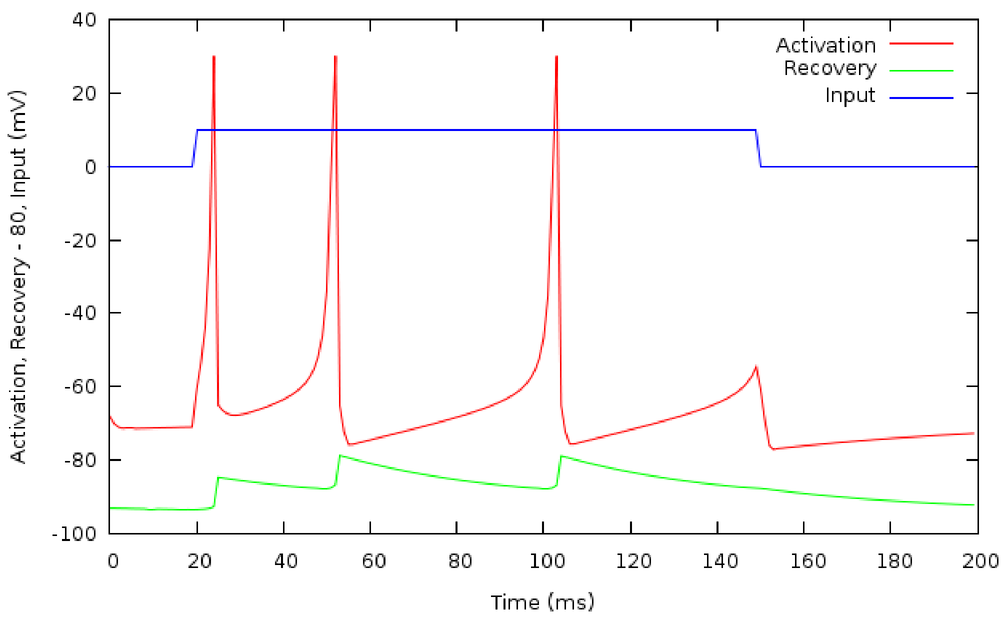

Figure 2.7: Activation and recovery (offset $-80mV$) levels of an Izhikevich neuron.

The activation of a regular spiking Izhikevich neuron when a 10mV input is applied is shown in figure 2.7. The Izhikevich model, unlike the LIF models, directly simulates the membrane potential of the axon of a neuron so that the voltages shown in figure 2.7 are exactly those received by efferent neurons (rather than simply a Dirac spike in the time step in which the activation level reaches the threshold) making for highly realistic network behaviour. Note that with each spike the recovery variable is incremented which causes the the interval between the second and third spikes to be greater than between the first and second. The recovery variable also enforces a refractory period so that the neuron cannot fire spikes in immediate succession, even with a powerful input current (not shown).

2.3 Summary

In this chapter the basic structure of the biological neuron and a number of computational neuron models have been presented. Note has been made of the issue of computational and behavioural complexity in neuron models and the behaviour of each of the presented models has been described. In the following chapter the operation of Sparse Distributed Memory is considered with reference to the simple perceptron model given here.

Chapter 3

Sparse Distributed Memory

This chapter introduces Sparse Distributed Memory as proposed by Pentti Kanerva's original thesis on the topic and then examines subsequent developments to the network. The first sections closely follow the chapters of Kanerva's thesis to best describe SDM and the closing sections discuss alterations to the network required for an implementation in spiking neural networks.

3.1 RAM and the Best Match Problem

Random Access Memory (RAM) is a form of instruction and data storage that consists of a number of memory locations, each uniquely accessed by a specific address, and a memory controller that contains an address decoder for each memory location. To access a memory location, an address is presented to the memory controller which in turn presents it to every address decoder in parallel. The decoder to which the presented address corresponds then activates its memory location, making it available for reading or writing.

Kanerva describes the problem of finding the best match to an arbitrary word from a given set of words as a task which is difficult in RAM [17]. The problem can be thought of as an abstraction of the problem of association or recognition of similar stimuli. For elaboration, let S be a set of size m of words, let Z be the word to match and let all words be from the language of words of size n over the alphabet $\{0, 1\}$, with the best match defined as the minimum Hamming distance between words.[1]

[1]Hamming distance will be used throughout this dissertation to compare (vectors of) binary numbers. Addresses or data will be considered *near* or *close* to one another if the Hamming

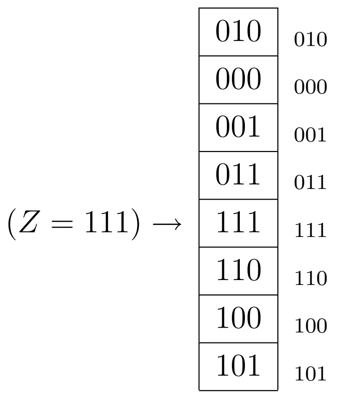

Figure 3.1: A fully occupied memory, showing memory contents in the outlined column and addresses in the subscript to the right.

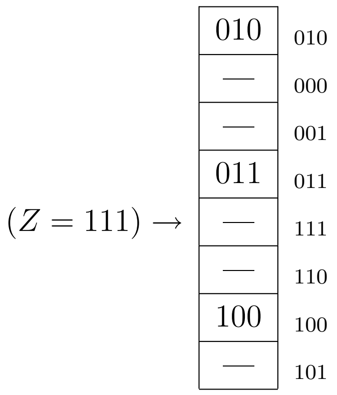

Figure 3.2: Memory when $m << 2^n$, showing a miss when addressing memory with some $Z \notin S$.

The problem is addressed by storing each word from S in RAM at the address pointed to by itself and the word to match is then used as an address to memory. When $m = 2^n$, memory is described as fully occupied and the best match to any Z is simply Z. Figure 3.1 shows full occupancy at $n = 3$ with the word Z addressing a memory location that contains itself. It should be noted that for the purpose of demonstration memory locations are arranged as a Gray code, so that the Hamming distance between two neighbours is exactly 1.

When $m = 2^n$ finding the best match is trivial but this is an unusual case. It is possible that $m << 2^n$ so that the contents of S are sparse in memory. Figure 3.2 shows $n = m = 3$ and the failure to find a best match when addressing memory with some $Z \notin S$, an occurrence which will be described as a *miss*.

To be able to find the best match to a word $Z \notin S$, the empty addresses around elements of S must be padded with duplicates of those elements. Figure 3.3 shows the padded memory and some $Z \notin S$ addressing the memory to find its nearest match.

Clearly, any padding algorithm must read or write every address in the address

distance between them is relatively small in the context in which they appear.

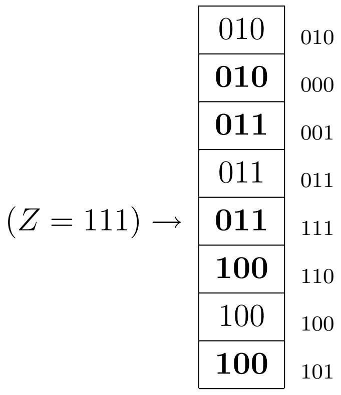

Figure 3.3: Memory padded with elements of S to facilitate finding the best match to Z.

space at least once so solving the best match problem in RAM with conventional address decoders requires time and space exponential in the size of the words in S.

3.2 The Best Match Machine

Having identified the complexity involved in solving the best match problem in RAM Kanerva proposes the best match machine in which conventional address decoders are replaced by perceptron-like neurons. A neuron with certain parameters may be used as an address decoder if each bit of a binary address is fed in parallel to the neuron inputs such that recognised addresses cause the neuron to fire and activate the *word-line* (or memory location) with which it is associated. Tuning a decoder neuron to a specific address involves setting the synaptic weights to -1 for the 0s in the address and $+1$ for the 1s and setting the threshold to the number of 1s in the address.

Neuron	Address	Weights	Threshold
A	{0,1,0}	{-1,+1,-1}	1
B	{0,1,1}	{-1,+1,+1}	2
C	{1,0,0}	{+1,-1,-1}	1

Table 3.1: Tuning decoder neurons to particular addresses.

The best match problem can now be solved with greatly reduced complexity. Table 3.1 shows the parameters of decoder neurons tuned to address the elements of S in figure 3.2. With these neurons and parameters the problem stands as it is in figure 3.2; presenting some $Z \in S$ to the decoder inputs causes the appropriate neuron to fire thereby making the match location available for reading; presenting some $Z \notin S$ causes no firing, resulting in a miss. As noted, the RAM solution to a miss is to pad memory. Decoder neurons make this process redundant: to associate many addresses to a single stored value (equivalently, to associate arbitrary words to a word of a given set) all that is necessary is to decrement the threshold of each neuron. Doing so increases the maximum Hamming distance between the tuned address and the presented input address for which a neuron fires. When the thresholds are decremented to a point such that a bank of decoder neurons form a total surjective function mapping any address to the nearest memory location containing an element of S, the best match problem is solved.

3.3 Examining Sparse Distributed Memory

The best match machine demonstrates a simple associative memory in which addresses within a certain Hamming distance of one another are associated to a particular data item. Kanerva's solution to the best match problem has an interesting corollary: if it is possible to map many addresses to a single memory location then it is not necessary for a memory location to exist for each address in the address space. Essentially, memory locations can be exist *sparsely* in the address space. Although the best match machine is designed to solve a problem which involves reading from memory, the machine's architecture equally permits writing to memory by storing data items in the nearest existing memory location to the given write address.

The best match machine illuminates another interesting property of neurons as address decoders. If lowering the threshold increases the range of addresses to which a decoder will respond then setting the threshold to negative infinity would cause the neuron to respond to all addresses. A bank of such decoders would fire in concert when presented with any inputs. This is obviously not useful itself but it serves to demonstrate the concept that just as many addresses may map to a single memory location, one address may map to many memory locations. By reducing the decoder thresholds beyond the point required to solve the best match problem, the ranges of addresses to which the decoders respond (the *response circles* or *access circles*) begin to overlap. Writing to one such overlapped address would cause copies of the written data to be *distributed* throughout memory.

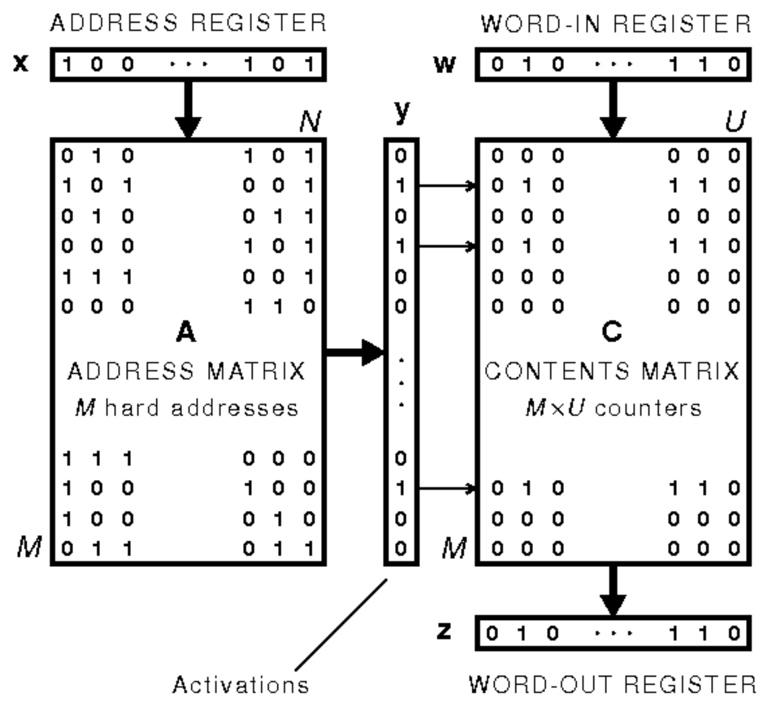

Figure 3.4: Writing to and reading from an SDM. Adapted from [18].

3.3.1 The Architecture of Sparse Distributed Memory

The architecture of Sparse Distributed Memory is shown in figure 3.4. Its structure is not dissimilar to RAM in that it consists of discrete components for decoding addresses (the *address matrix*) and storing data (the *contents matrix* or *data matrix*) but the key differences with RAM are that memory locations are sparse in the address space and the address decoder may activate many word-lines concurrently by firing multiple decoder neurons [17]. This architecture is employed by all of the SDM variants reviewed in this chapter so it is worth examining its operation in some detail.

Each of the M rows in the address matrix corresponds to a decoder neuron that receives the N address bit inputs in parallel. The decoders are tuned so that their response circles are evenly distributed throughout the address space, and their axons feed into the data matrix. Each of the U columns in the data matrix corresponds to a data neuron which has a synapse with the axon of every decoder neuron and a threshold set to half of the average number of decoders that are expect to fire during a write operation. The synapses between the decoder and data neurons are the bit storage locations of an SDM, having an equivalent role

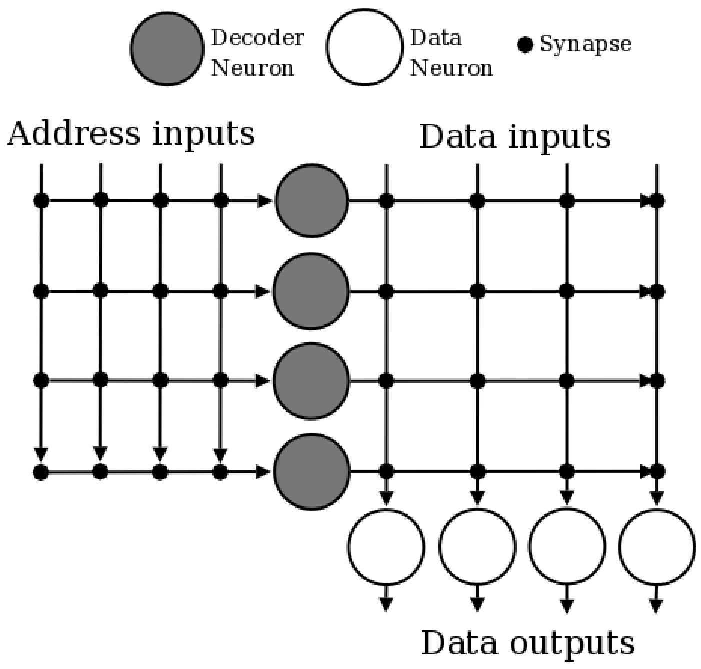

Figure 3.5: A CMM in the synapses between decoder and data neurons.

to the bistable logic gates used in conventional memory. The architecture can also be described as a correlation matrix memory [25], shown in figure 3.5 with the decoder axons feeding horizontally from the left, the data neuron dendrites running vertically downwards and the synapses at the intersections between the two. Data to be written to memory is presented one-bit-per-data-neuron into the trailing dendrites at the top of the figure (the absurdly small memory shown could take data items a maximum of four bits in size).

Figure 3.4 shows the consequence of a write to an initially empty memory. The presentation of an address x to the address matrix produces a *word-line vector* (the set of memory locations activated by firing decoder neurons) and the simultaneous presentation of a data item w to the data matrix sets the synaptic weights to 1 wherever a 1 in the input crosses an active word-line. The result can be seen to be the distributed storage of the data w in a number of locations.

A read is subsequently performed by presenting the same address x to the address matrix. The same decoders fire and these spikes propagate, where the synaptic weights are greater than zero, into the dendrites of the data neurons. Each data neuron sums the inputs and fires if the threshold is met [9] producing 1s in their respective locations in the output vector z.

3.3.2 Sparse Distributed Memory Performance

In a memory in which storage locations are sparse in the address space it may seem that data items are susceptible to corruption as writes to different close addresses contend for the use of the corresponding locations. This phenomenon occurs in the cache of a stored-program computer but is accommodated for in suitably large SDMs by distributed storage.

The performance of an SDM with respect to error free recovery of stored data depends heavily upon the occupancy of the memory [8]. Occupancy can be represented as the probability that a random bit is set to 1 after n writes

$$h = 1 - [1 - \frac{w}{W}\frac{d}{D}]^n \tag{3.1}$$

where $\frac{w}{W}$ is the average fraction of word-lines active during a write and $\frac{d}{D}$ is the average fraction of data word bits set to 1. For occupancies of less that 0.5 it has been shown that the quality of retrieved data (the similarity between written and read data, see equation 3.4) can be very high but above this occupancy the quality tends to decrease [9]. A consequence of this is that an SDM has its maximum information content (the number of bits which may be stored and retrieved) at an occupancy of exactly 0.5.

3.4 Adapting SDM for Biological Plausibility

Kanerva's thesis concludes with the compelling observation that the neural structures in Sparse Distributed Memory (the array of address decoders with long axons forming synapses with each of the data neuron dendrites) are remarkably similar to those found in the cerebral cortex. Given the role of the cerebral cortex in animal memory, Sparse Distributed Memory seems to be an ideal data store for research into biological approaches to learning and recognition tasks. However, the SDM variant described above is far from biologically plausible: the use of the perceptron as a neuron model and the resulting global synchronicity of the network does not imitate the complex neuron dynamics and asynchronous behaviour of biological networks. The research described in this section provides the basis for constructing an SDM variant from more realistic neuron models.

3.4.1 Population Coding

The all-or-nothing nature of neural spiking permits the output of a neuron at a particular time to be represented as a single bit, so the firing of a population of neurons may be represented by a vector of bits. Problems arise, however, when employing this full binary coding in a biologically plausible network. In an asynchronous environment neurons do not fire in perfect concert so a clean vector of 1s and 0s representing a data word cannot be expected at each simulation time step. It is also difficult to control the occupancy of an SDM variant using full binary codes as the number of decoder neurons which fire for a given address cannot be fixed [9].

N-of-M coding conveys information in the the subset of N neurons from a population M which fire. By fixing N for each volley of spikes it becomes possible to deal with network asynchronicity, in that a layer of neurons 'know' that a complete data word has arrived when exactly N spikes have been received [8]. A further improvement to N-of-M coding is rank-order N-of-M coding in which information is conveyed both by the subset of neurons that fire and the order in which they fire. It has been suggested that rank-order coding is a biologically plausible way of encoding information in populations of spiking neurons [22] and this approach has the advantages of encoding more information than full binary coding for certain values of N and M [9].

The information encoded by a rank-order N-of-M spike volley is given by

$$\log_2 \left(\frac{M!}{(M-N)!} \right) \text{ bits} \tag{3.2}$$

that is, the possible firing orders of N firing neurons divided by the possible firing orders of the $M - N$ non-firing neurons.

In the ordered N-of-M scheme the significance of each successive firing in a volley decreases monotonically.[2] Consequently, rank-order firings can be usefully represented by a significance vector in which the indices to the vector represent neurons and the values at those indices represent the significance of each firing [9]. Significance is calculated for each firing by multiplying the significance of the previous firing by a significance ratio σ. So, the following 3-of-7 code with significance ratio $\sigma = 0.9$ (in which neuron 4 fires first, neuron 7 fires second and

[2]For example, in a 2-of-3 code, the first firing encodes a one-of-three choice and the second firing encodes only a one-of-two choice.

neuron 6 fires third, with neurons 1, 2, 3 and 5 not firing at all) is represented

$$\{0, 0, 0, \sigma_1, 0, \sigma_3, \sigma_2\} = \{0, 0, 0, 1, 0, 0.81, 0.9\} \tag{3.3}$$

When scaled to unit length (such that $\sum_{i=1}^{N} \sigma_i^2 = 1$) the significance vector representation is useful for comparing the quality of codes read from an SDM with those originally written. The quality of a retrieved code y with respect to the original written code x is given by the dot product of the two vectors

$$\sum_{i=1}^{N} x_i y_i \geq \theta \Rightarrow X \approx Y \tag{3.4}$$

A dot product of 1 denotes an exact match, or perfect quality, and progressively smaller values denote increasingly fuzzy matches.

3.4.2 Employing Rank-Order N-of-M Codes

For rank-order N-of-M codes to be employed in multi-layered neural network such as Sparse Distributed Memory each neuron layer must be made sensitive to the rank-order of the inputs it receives and the N-of-M firing patterns of a population of neurons must be enforced to ensure that the number of neurons firing in each successive layer does not grow without bounds.

Furber et al. note that a population of neurons such as an address decoder can be made to be sensitive to rank-order codes using only binary synaptic weights [9].[3] By randomly setting i of the A address bit input weights of each decoder neuron to 1 and the rest to 0 each decoder may sample non-intersecting sets of the address spikes and so produce unique activation levels. Those decoders which sample the greatest number of spikes in an address will receive the largest input and therefore spike first so that a rank-order address is translated into a rank-order word-line vector. A small number of decoders may unavoidably sample the same subset of spikes in certain addresses, in which case they will fire simultaneously (in the same time step), but these occurrences have been shown to be infrequent for appropriate i-of-A.

[3]This is of concern when constructing large-scale neural networks because of the memory and coprocessor requirements for floating-point operations. Neuron models more complex than the perceptron usually require floating-point numbers to accurately represent activation level but by setting synaptic weights and thresholds to binary and integer values respectively resource requirements may be minimised.

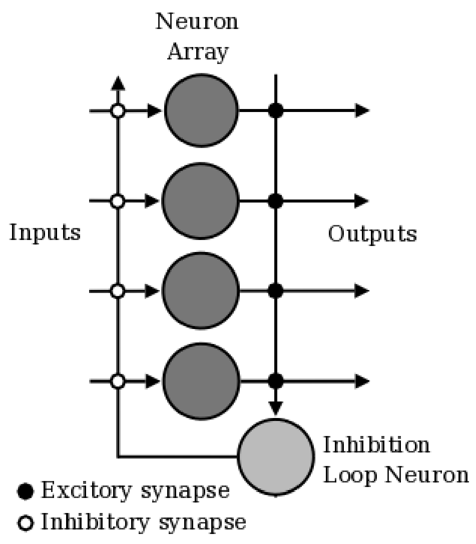

Figure 3.6: An inhibitory feed-back loop for enforcing N-of-M firing patterns.

Bose et al. address the problem of enforcing an N-of-M firing pattern by employing an inhibitory feed-back circuit on each layer of neurons in a network [6]. Figure 3.6 shows the structure of such a circuit. The inhibition loop neuron has a threshold set to N and receives excitatory input (that is, the synapses between the neurons have positive weights) from each of the neurons in the layer. When N spikes have been received by the inhibition loop neuron it fires inhibitory input into all of the neurons in the layer thereby lowering their activation levels and preventing further firings. For N-of-M codes to be enforced in the firings of an array of neurons, firings must occur asynchronously: no mechanism exists here to suppress excess firings if $N + 1$ or more neurons fire in concert. Even if such a mechanism did exist, it would be impossible to unambiguously select which N of a set of homomorphic spikes to propagate.

3.5 Summary

This chapter has outlined the structure and operation of Sparse Distributed Memory along with the modifications required to build an SDM variant in biologically plausible networks. The following chapter documents the process of a construction along these lines and analyses the resulting network.

Chapter 4

Constructing SDM from Spiking Neurons

This dissertation has so far reviewed a number of computational neuron models, a variant of Sparse Distributed Memory using the perceptron and work towards an SDM variant suitable for construction from spiking neuron models. This chapter details the process of constructing an SDM variant from such models, the problems involved and the verification of the expected behaviour, beginning with a brief overview of each area of the work.

Network Structure

Sparse Distributed Memory consists of an address decoder responsible for mapping addresses to word-lines and a data matrix containing the bit storage locations on the word-lines. The presented research began with the construction of an address decoder consisting of 4096 decoder neurons (a number borrowed from work on an SDM variant using rank-order codes by Furber et al. [9]) each of which was sensitive to some fraction of the inputs presented on parallel *address-lines* passing through every decoder. Neuron parameters were selected and an inhibitory feed-back circuit established to ensure that approximately 64 of the 4096 decoder neurons (an arbitrarily chosen fraction) fired in response to each rank-order 11-of-256 (numbers again suggested by Furber et al.) address delivered to the address decoder. The data matrix was appended to the address decoder by wiring the axons of the decoder neurons to the dendrites of 256 data neurons. A similar process of parameter selection and construction of an inhibitory feed-back loop was followed to produce rank-order 11-of-256 firing patterns in the data neurons.

Network Operation

Sparse Distributed Memory is effectively a 2 layer neural network: presentation of an address to the first layer (the address decoder) results in a number of firings into the second layer (the data matrix) which in turn fires depending on the weights between the 2 layers, producing the network output.

Biologically plausible neuron models introduce additional complexity to this process in that they may 'remember' previous network activity in the gradual decay of their activation levels. Consequently, network output may depend not only on the immediate input but also on historical inputs and internal activity. Work on exploiting this behaviour—so different from that of the logic found in CPUs—would certainly be interesting but it presents difficult complications to this research. To avoid interference between one input and the next an artificial clock was imposed with a period long enough for most neurons in the network to return to resting states. Inputs to the network were then restricted to the beginning of each clock cycle.

Such a clock is an arguably unlikely device to find in a biological neural network. Its use here is justified firstly by the observation that some grouping of spikes into waves must occur for rank-order codes to have meaning (there cannot be a first, second and third spike in a train of firings which has no discernible beginning or end) and secondly by an aversion to dealing with the complexity resulting from passing continuous spike trains through a spiking neural network.

Testing

Sparse Distributed Memory is a candidate learning agent for the inverted pendulum problem because of its associative capabilities, in this case because of a potential ability to recognise similarity between pendulum states through its address inputs and act accordingly via its data outputs. As such, testing of the completed SDM was primarily concerned with attempting to demonstrate these capabilities by presenting sequences of similar addresses and examining the data matrix output for corresponding similarity.

A number of metrics exist to further examine the performance of an SDM variant using rank-order codes [9] but of greater concern was the stated ability of biologically plausible neuron models to 'remember' network activity from one cycle to the next. To test for any adverse effects of this ability, identical addresses were delivered to the SDM at different times with the expectation that identical data should be produced if no inter-cycle interference was present.

Finally, long sequences of random addresses were presented to the SDM to test the expectation that resulting activity should be distributed, both spatially and temporally, evenly throughout the network.

Software Tools

Although Sparse Distributed Memory exhibits rather remarkable properties it is essentially a simple network as the figure (3.5) of a correlation matrix memory shows. Consequently, the construction of an SDM variant in spiking neural networks involved significantly more time experimenting with various neuron models and parameters than writing code. This does not, however, diminish the importance of the Brian package for Python to this work.

Brian is a spiking neural network simulator that permits a programmer to tersely specify neuron models, neuron groups and network connectivity in a Python program [11]. Using Brian it was possible to build the entire SDM and associated read, write and punishment methods in a single class of around 100 lines of code. Although Python is an interpreted language which is not currently well disposed to exploiting parallel architectures, Brian programs may boast speed near to that of an equivalent C program as a result of the careful optimisation of the simulator back-end and the use of vectorised computations from the NumPy package. Brian has been instrumental in the timely development of an efficient, stable and well-featured simulation that would likely have been infeasible with many other tools.

4.1 Building an Address Decoder

Building the address decoder involved selecting an appropriate neuron model from the ones described in chapter 2, creating an array of neurons sensitive to rank-order and implementing the feed-back inhibition circuit described in section §3.4.2 to enforce w-of-W decoder neuron firings. The latter feature is important given that the data neurons should have thresholds of approximately half the number of decoders that are expected to fire in each cycle (see §3.3.1) which is difficult to predict unless w is controlled. The inhibition mechanism requires that neurons fire largely asynchronously, so much of the work on the address decoder proceeded with this in mind.

4.1.1 Selecting a Neuron Model

Given an array of neurons which receive inputs in parallel, such as bank of address decoder neurons, it is sensible when employing rank-order codes to expect the neurons receiving the most input to fire first and those receiving less input to fire later. This informs the choice of neuron model in a rank-order SDM, as will be demonstrated.

Take an array of 10 conventional LIF neurons, indexed from 0 to 9, that receive 10 inputs in parallel somewhat like the decoder neurons shown in figure 3.5 in the previous chapter. Each neuron is sensitive to a number of inputs equal to its index[1] and every neuron has a threshold of 5mV. Now, if spikes are delivered synchronously on all 10 input lines then every neuron with an index greater than 4 will fire simultaneously and the remaining neurons will not fire at all so no information about the input sensed by each neuron will be encoded in the output. This causes two problems: it is impossible to unambiguously select the N most active neuron firings to enforce N-of-M firing patterns from one neuron layer to the next and (in an SDM address decoder) it is impossible to tell which storage locations were most closely related to the given address.

The Rate Driven LIF neuron model solves these problems by introducing a temporal separation between neural inputs and outputs which depends upon the strength of the input; strong input drives the activation level to the threshold more quickly than weak input. The RDLIF neuron model thus preserves in the firing times of an array of neurons information about the input sensed by each one. However, the RDLIF model introduces a degree of behavioural complexity which the LIF model does not express.

It is intuitive with the LIF model that a neuron with a threshold of xmV will fire immediately when the value produced by its summing junction is greater than or equal to xmV. This is not so with the RDLIF model as the value produced by the summing junction drives the derivative of the activation level rather than the activation level itself. It is certainly possible, knowing the inputs received and the leaky time constants used, to establish if and when an RDLIF neuron will fire but these calculations denote a departure from the logic-gate-like behaviour which makes perceptron-based networks so predictable. Further behavioural complexity is introduced by asynchronous inputs: the activation level decay demonstrated

[1]For example, neuron number 4 would have 4 of its synaptic weights set to 1mV and the rest to 0mV.

by any leaky model means that a number of inputs exceeding xmV presented over time may not cause a neuron with a threshold of that same value to fire, as the period between 25ms and 50ms in figure 2.5 shows.

The behavioural complexity of the RDLIF neuron model was deemed sufficiently rich, based upon the above reasoning, to be used in the SDM construction. In brief trials along the lines of the experiment above the Izhikevich neuron model expressed relationships between inputs and outputs which were very difficult to predict and so the model was deemed too complex for this initial attempt at SDM construction.

It should be noted here that a bug was introduced by the misplacement of brackets when programming the RDLIF neuron model in Brian. Consequently, the neuron activation level was calculated as

$$\dot{A} = (R - A)/\tau_A \tag{4.1}$$

instead of dividing A by τ_A and then subtracting the result from R. The primary effect of this mistake was to decrease the height of the peak in the activation level for a given input, which was unconsciously corrected for in the selection of neuron thresholds that produced the desired firing behaviour. Since this error was only spotted when writing up this chapter it has been promoted from a bug to a *feature* and persists for the rest of the dissertation.

4.1.2 Selecting Neuron Parameters

In order to specify fine control over the number of decoder neurons that fire with an inhibition loop the number of simultaneous neuron firings (firings in the same simulation time step) must be minimised. Selection of the following parameters was conducted with the intention of producing the richest asynchronous address decoder activity possible.

Leaky Time Constant

The first parameters selected for the decoder neurons were the leaky time constants used in their differential equations. An array of 10 RDLIF neurons was constructed, each sensitive to a number of inputs equal to their index as in the above demonstration. The neuron thresholds were set to 0.1mV to ensure that every neuron spiked and the spike times were recorded following synchronous

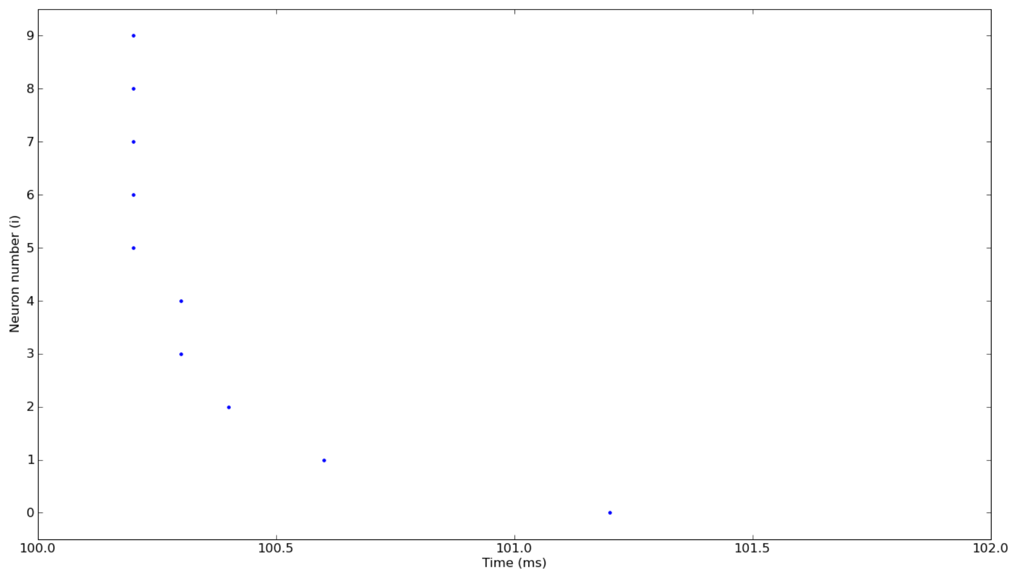

Figure 4.1: Firings of RDLIF neurons receiving i simultaneous inputs. $\tau = 10$ms.

delivery of spikes on all 10 input lines at $t = 100$ms. Figure 4.1 shows the distribution of firing times for leaky time constants of $\tau_A = \tau_R = 10$ms. The 10 neurons fired across a period of 1ms on only 5 distinct time steps, which is far from the smooth distribution that was hoped for.

Figure 4.2 plots the distribution of firing times for $\tau_A = \tau_R = 100$ms, showing firings across a period of approximately 10ms, with each neuron firing on a distinct time step.

These trials demonstrate a limitation of the method by which Brian calculates discrete approximations of the neuron differential equations. By default Brian recalculates network state in 0.1ms time steps so that multiple neurons firing within this period will appear to do so simultaneously. Adjusting the granularity of the simulation time steps, say to 0.01ms, would reduce the occurrence of simultaneous firings but would also incur a tenfold increase in computational cost. For this reason, the leaky time constants for the RDLIF decoder neurons were fixed at 100ms and the default simulator time step granularity was used.

Address Bit Sampling

To decide the number of address bits that should be sampled by each neuron (that is, the number of synaptic weights that should be 1) an address decoder was constructed. 256 address-lines ran through an array of 4096 RDLIF decoder

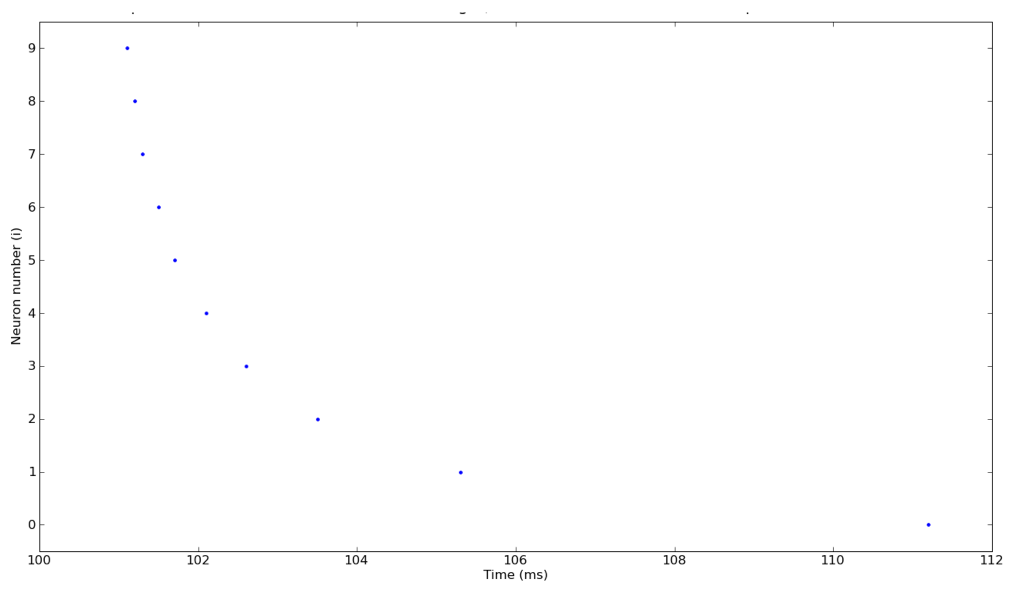

Figure 4.2: Firings of RDLIF neurons receiving i simultaneous inputs. $\tau = 100$ms.

neurons with thresholds set to 0.1mV and leaky time constants were fixed at the value arrived at in the previous trials.

A number of trials were conducted in which a proportion of the synaptic weights (i-of-A) between the address-lines and decoders were set to 1 at random and the number of distinct time steps on which decoders fired in response to a randomly-generated synchronous 11-of-256 address was recorded. The activation level of any neuron that fired was reset to -1000mV to prevent any repeated firings.

Table 4.1 shows the effect of varying the number of inputs to which decoders are sensitive on the number of distinct bands in which they fire. Only 5 repetitions of each variation were performed, after which the standard deviation was seen to be so small that further repetitions seemed unnecessary. The decoders fired on a maximum of around 11 distinct time steps (when they sampled 128 inputs) which may be because the only factor determining firing times was the number of inputs received: those decoders receiving 11 inputs fired in the first band, those receiving 10 fired in the second and so on. Figure 4.3 clearly shows the banding in the firing times of decoder neurons sensitive to 128-of-256 inputs when presented an address at 100ms.

	No. of bands	
i-of-A	Mean	Std dev
1	1.0	0.000
2	2.0	0.000
4	2.8	0.447
8	3.4	0.547
16	4.4	0.547
32	6.2	0.447
64	8.0	0.707
128	10.6	0.547
192	9.2	0.447
224	7.0	0.000
256	1.0	0.000

Table 4.1: The effect of i-of-A on banding in decoder neuron firing times.

Address Presentation and Decoder Threshold

To attempt achieve to a richer spread of firing times, asynchronous rank-order addresses with 2ms delays between each input spike were introduced. Figure 4.4 shows the decoder firing times in response to an address presented at 100ms. Banding is still clearly evident and may be attributed to the low threshold that causes those neurons sensitive to inputs to fire immediately after receiving them. Too many neurons fire in the very first bands to effectively select w of them and suppress the rest.

The effect of increasing the decoder neuron threshold to 1mV can be seen in figure 4.5 and a very rich spread in firing times is apparent with only soft banding. This is likely to be a consequence of the delay between inputs and firings in the RDLIF model: say two neurons receive the same input, then their activation levels begin to climb and failing any other inputs they will fire simultaneously but if, during this delay between input and firings, one of the neurons receives further inputs then its activation level will climb more rapidly and it will fire sooner. This behaviour can only be expressed when the neuron threshold is sufficiently high that the time taken for the activation level to reach it is greater than the time between input spikes.

Ensuring rich asynchronicity in the firing of data neurons requires that the decoder neurons fire not only asynchronously but also at greater intervals than

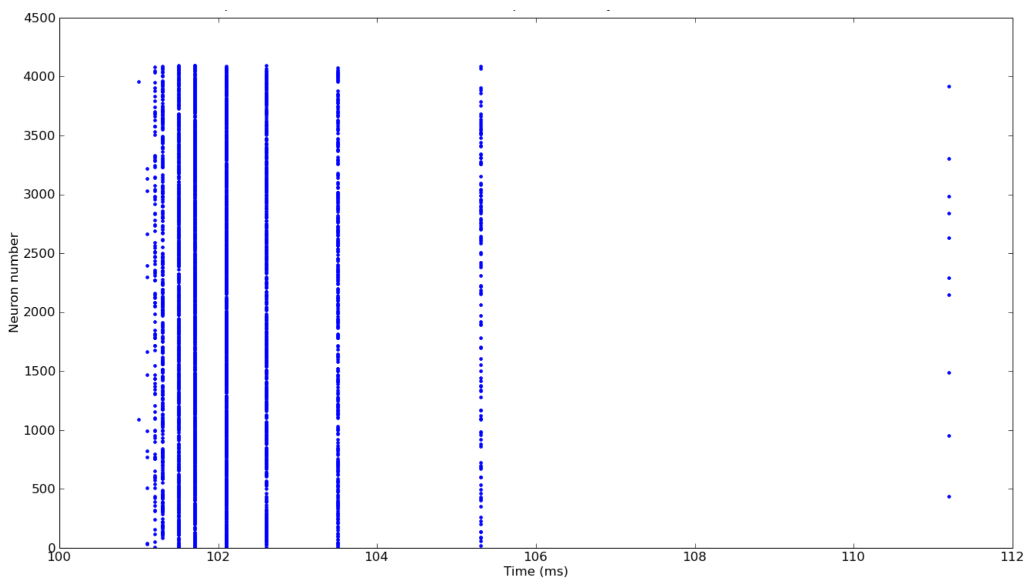

Figure 4.3: Decoder firings in response to a synchronous address.

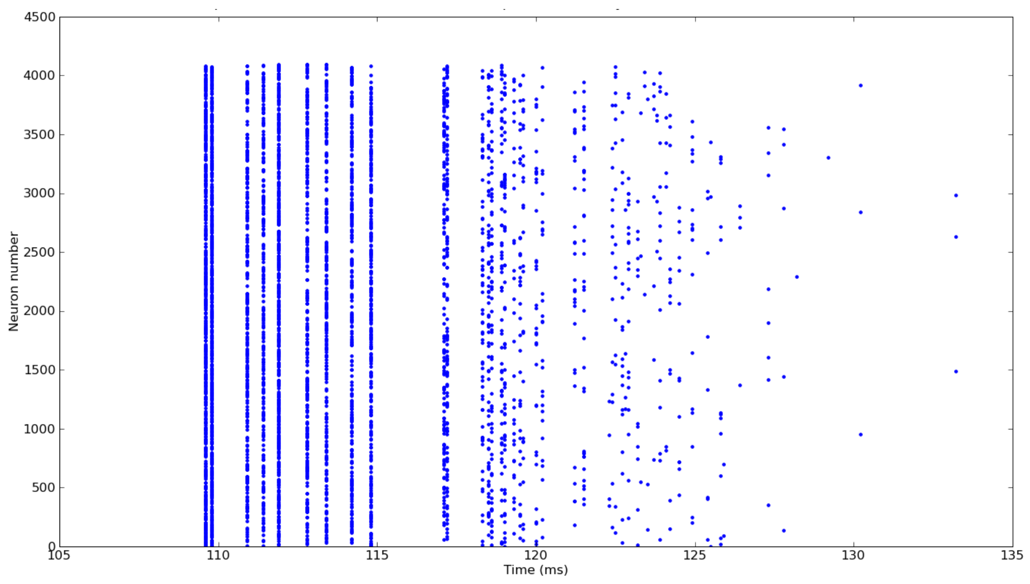

Figure 4.4: Decoder firings in response to an asynchronous address ($\theta = 0.1$mV).

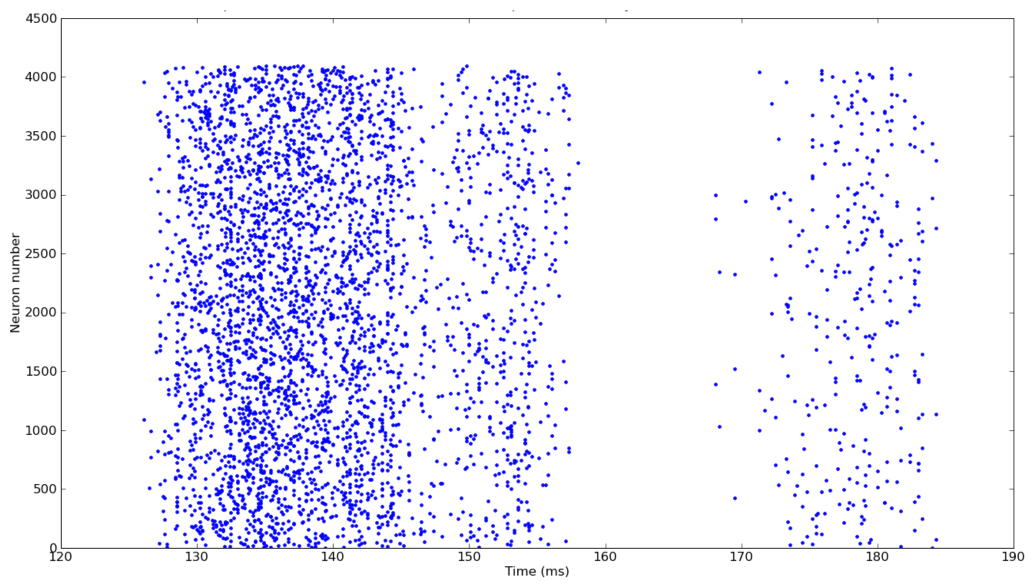

Figure 4.5: Decoder firings in response to an asynchronous address ($\theta = 1$mV).

seen in figure 4.5.[2] Raising the decoder neuron threshold to 2mV effectively achieved this by 'thinning the field' of firings as can be seen in figure 4.6.

A yet sparser field of firings was achieved by raising the decoder threshold to 3mV (figure 4.7) but at this threshold too few decoders fired at too greater intervals to be useful.

4.1.3 Inhibitory Feed-back

The above trials demonstrated that varying decoder neuron thresholds allows only very coarse control over the number that fire in response to an address. Experiments were conducted into the use of inhibitory feed-back to better control address decoder behaviour. An address decoder was constructed with the parameters derived above: 4096 decoders on 256 lines sampling a random 128 of the inputs each, 2mV thresholds, 100ms leaky time constants and 11-of-256 addresses presented asynchronously with 2ms between input spikes. The activation level reset of the decoder neurons was adjusted to a more realistic value of -1mV.

To establish the number of decoders which fired in response to an address, 10 randomly generated addresses were presented (neuron activation and rate levels were reset between each presentation) and the number of firings was recorded.

[2]At least the first 100 decoders fire within 5ms here, which may appear to be practically synchronous input to data neurons with slow time constants.

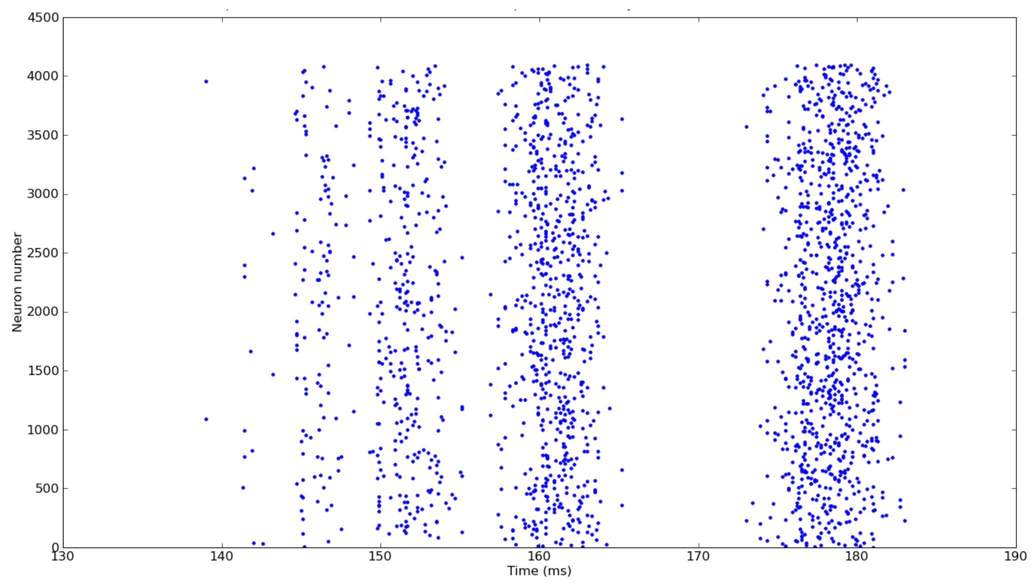

Figure 4.6: Decoder firings in response to an asynchronous address ($\theta = 2\text{mV}$).

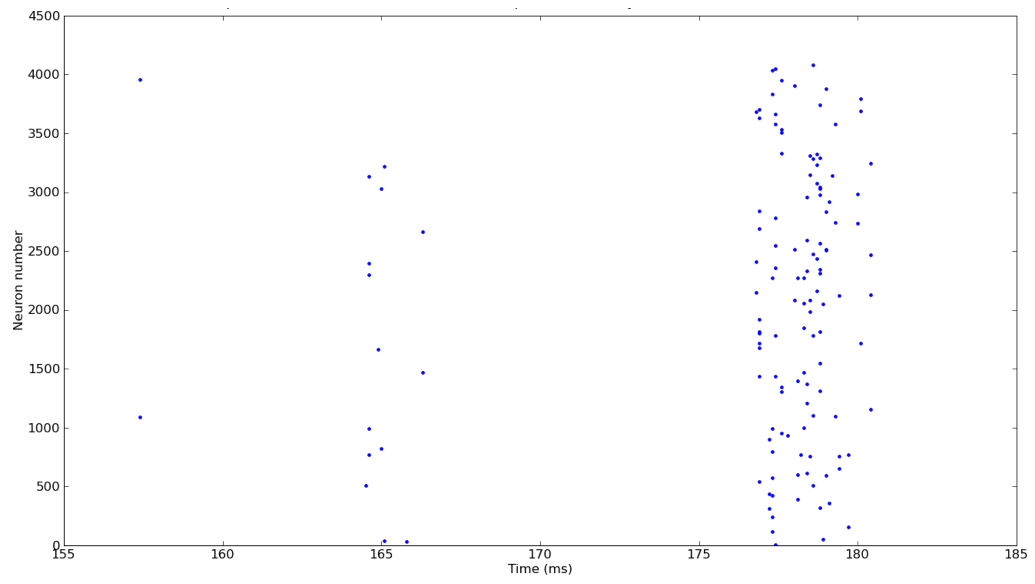

Figure 4.7: Decoder firings in response to an asynchronous address ($\theta = 3\text{mV}$).

A mean of 2044.3 decoders fired with a standard deviation of 33. Next, a single inhibition neuron with a time constant of infinity (so that activation level did not decay) was created and wired to receive inputs with a synaptic weight of 1mV from every decoder neuron. The threshold was set to 64mV and the axon of the inhibition neuron was connected back to the rate variable of every decoder by synapses with weights of -10mV, that is, one less than the maximum possible number of address inputs. It was expected that exactly 64 decoders should fire before the inhibition neuron fired and suppressed further activity. In fact, an average of 72.7 decoders fired (standard deviation of 6.3) which demonstrates a limit on the degree of control that an inhibition neuron can exert over a population. For the same reason that it is impossible to enforce any X-of-Y firing pattern if all neurons fire simultaneously, it is also impossible to enforce the pattern if any subset of size z fires simultaneously after at least $X - z$ neurons have already fired in the same cycle.

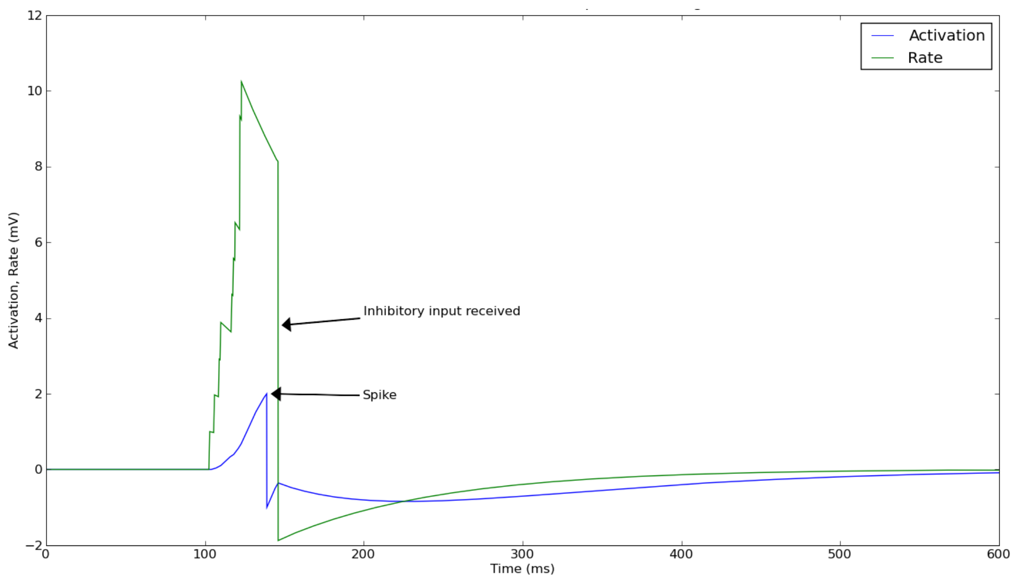

Figure 4.8: Activation and rate levels of a decoder neuron in response to an address.

Adjusting the leaky time constant of the inhibition neuron to 100ms and repeating the trials a further 10 times resulted in a mean firing count of 71 with a standard deviation of 3.2. It was expected that introducing leak to the inhibition neuron would permit more decoder neurons to fire than one which did not leak at all; the lower mean and tighter standard deviation might in this case be attributed to the small sample size. Nevertheless, it appeares that the slow time constant and tight grouping of decoder neurons (resulting in little activation decay between inputs) made the difference in behaviour between a leaky and non-leaky inhibition neuron negligible.

The effect of the inhibitory feed-back on decoder activation levels can be seen in figure 4.8. The neuron shown is one of a number that fired in response to an address presented at 100ms (the jagged growth of the rate variable reflects reception of 9 asynchronous address spikes) after which the activation level was reset and then began again to climb since the rate variable remained high. Inhibitory input was received some milliseconds later and the rate variable was forced down, taking the activation level with it. Note that some 500ms after presentation of the first address input both the activation and rate variables have largely recovered from the inhibitory feed-back spikes.

4.2 Building a Data Matrix

Having established appropriate neuron parameters during the construction of the address decoder, building the data matrix proved to be relatively simple. Again referring to figure 3.5, the bit storage locations of the data matrix were formed from synapses between the axons of the decoder neurons and 256 RDLIF data neurons. Construction of the data matrix involved the specification of these neurons and synapses in Brian, the reuse of the inhibition circuit developed for the address decoder and the adjustment of some of the neuron parameters. An approximate 11-of-256 (d-of-D) rank-order code was aimed for in the output of the data neurons but some variation was permitted in this figure given the demonstration in the previous section of the difficulty of enforcing exact firing patterns.

4.2.1 Data Neuron Parameters

Data neurons inherited most of the parameters derived for the decoder neurons: the leaky time constants were set to 100ms and binary weights were employed on all synapses. However, due to the greater number of inputs received by the data neurons from the decoder neurons (an average of the number of decoder neuron firings multiplied by the memory occupancy if the data matrix holds random synaptic weights) a different threshold was required. The threshold of $\frac{1}{2}w$ specified in §3.3.1 is appropriate for a perceptron model data neuron but the nature of the RDLIF neuron means that such a threshold might not be reached given $\frac{1}{2}w$ inputs. The data neuron threshold was set somewhat arbitrarily to $\frac{1}{5}w$ which permitted sufficient spread in the firing times when delivered address matrix output in response to random addresses to enforce an approximate d-of-D firing pattern with the inhibition circuit.

4.2.2 Inhibitory Feed-back

Experiments with the inhibitory feed-back of the data neurons showed that both the inhibition neuron threshold and its synapses with the data neurons required alteration from the parameters established for the address decoder inhibition loop.

Noting that an inhibition neuron threshold of w in the address decoder permitted some $w + n$ decoder neurons to fire, the data matrix inhibition neuron threshold was set to $d - 2$. When trials were conducted with this threshold and a synaptic weight between inhibition and data neurons of -10mV, some 80 data neurons fired in response to address matrix output, far exceeding the expected number. It became apparent that the feed-back provided by the inhibition neuron was not strong enough to pull the rate level of most of the data neurons to a resting state. This effect can be seen in figure 4.9, as a neuron spikes *after* the first of 4 distinct inhibitory inputs (an unwanted consequence of the large number of data neuron firings causing repeated firings of the inhibition neuron) were received.[3] This problem was solved by setting the inhibitory weights to the expected number of data neuron inputs $occupancy * w * 1.5$ (the last term to account for any data neurons receiving unusually large inputs) as can be seen in the firing counts listed in the following section.

4.3 Testing and Analysis

Testing of the completed SDM was carried out to ensure that it expressed the associative abilities expected of it and that its activity was consistent and balanced under continuous operation. In all trials each decoder neuron was initialised to be sensitive to half of the 256 address-lines on which 11-of-256 rank-order addresses were presented, the decoder inhibition neuron threshold was set to 64mV, the

[3]This is a pronounced example, in which each of the data neurons received approximately 60mV of input as a consequence of 70 decoder neurons firing into a memory of 0.8 occupancy but it is certainly valid in expressing the problem involved.

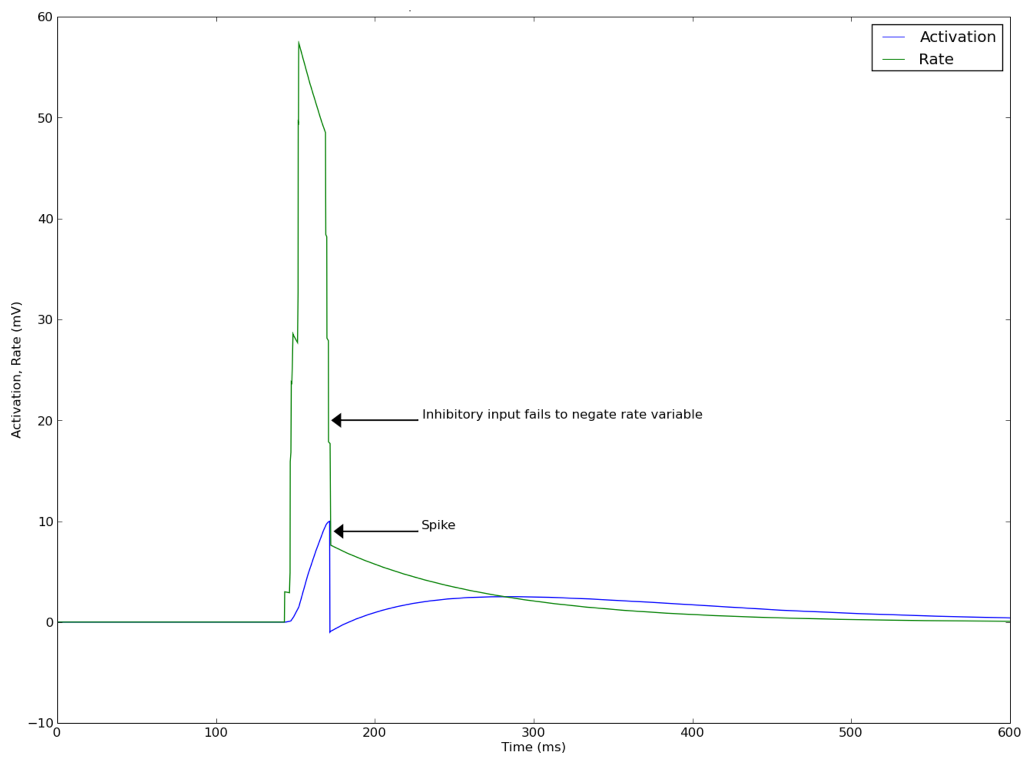

Figure 4.9: Failure of inhibitory feed-back to control the rate level.

data matrix was set with randomly distributed binary synaptic weights to the occupancy specified for the trial and the cycle period was varied between 500ms and 1000ms. All other parameters were set to the values established above.

4.3.1 Associative Capabilities

To test the associative capabilities of the SDM a number of trials were conducted in which ‘clean’ addresses and corrupt counterparts were presented at 500ms intervals to the SDM at occupancy 0.5 (chosen in relation to the learning method described in the following chapter). In each of the 5 trials 100 random addresses were generated and then corrupt copies were produced by randomly changing the position of between 0 and 3 of the spikes in the address. For example, taking a 3-of-7 address in significance vector representation

$$\{0, 0, 0, \sigma_1, 0, \sigma_3, \sigma_2\}$$

and randomly changing one spike, say the second, might give

$$\{0, \sigma_2, 0, \sigma_1, 0, \sigma_3, 0\}$$

Note, however, that the order of the other spikes in the address is not altered. Rank-order N-of-M codes may be corrupted either by altering the order of the spikes in them or by changing the locations of the spikes in the code; only the latter corruption was used here, partly for simplicity and partly due to the likely patterns of address inputs discussed in the following chapter. Each of the clean addresses was presented in turn and the approximately N-of-M firing of the data matrix was recorded, then all of the neurons in the network were returned to a resting state and the process was repeated for the corrupt addresses. Finally, the quality of the corrupt addresses and data with respect to their clean counterparts was calculated using the metric described in §3.4.1.

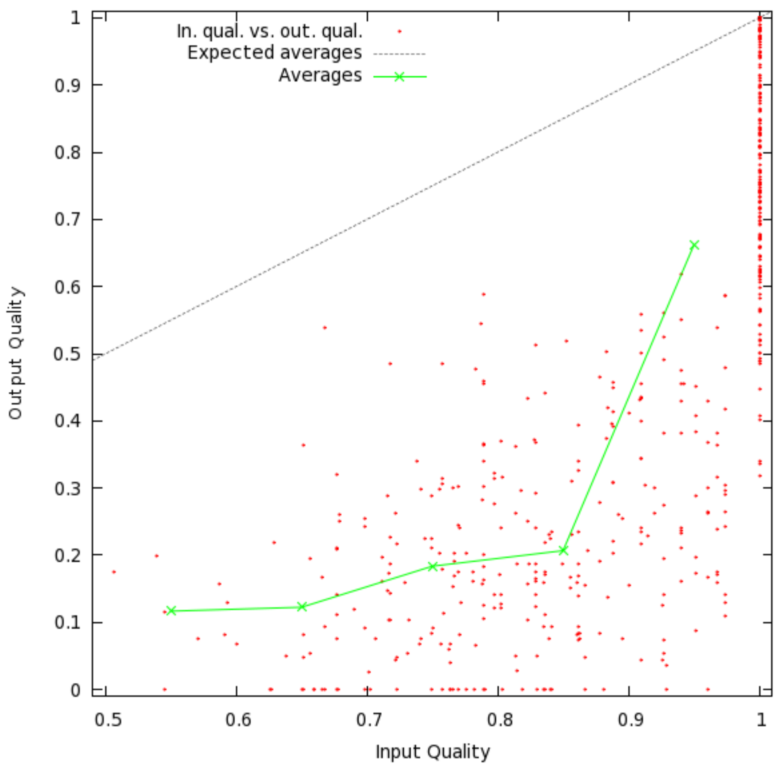

Figure 4.10: Input vs. output quality for corrupt addresses at 500ms intervals.

Figure 4.10 shows the quality of input addresses against the quality of data matrix output across all 5 trials. Some associative ability is demonstrated in that the average data qualities for each address quality bracket 0.1 wide sit clearly above what would be expected if the output data was purely random.[4] However, data quality was largely poor, peaking only in the range of 0.9–1.0 of address qualities. Even here perfect quality addresses returned data of highly variable quality. This is a strong indicator of inter-cycle interference since SDM behaviour, while complex, is entirely deterministic so that any difference in data output resulting from the presentation of two identical addresses at different times must be a consequence of 'remembered' prior network activity. Some addresses of high quality (greater that 0.9) returned data of quality no greater than the average random code.

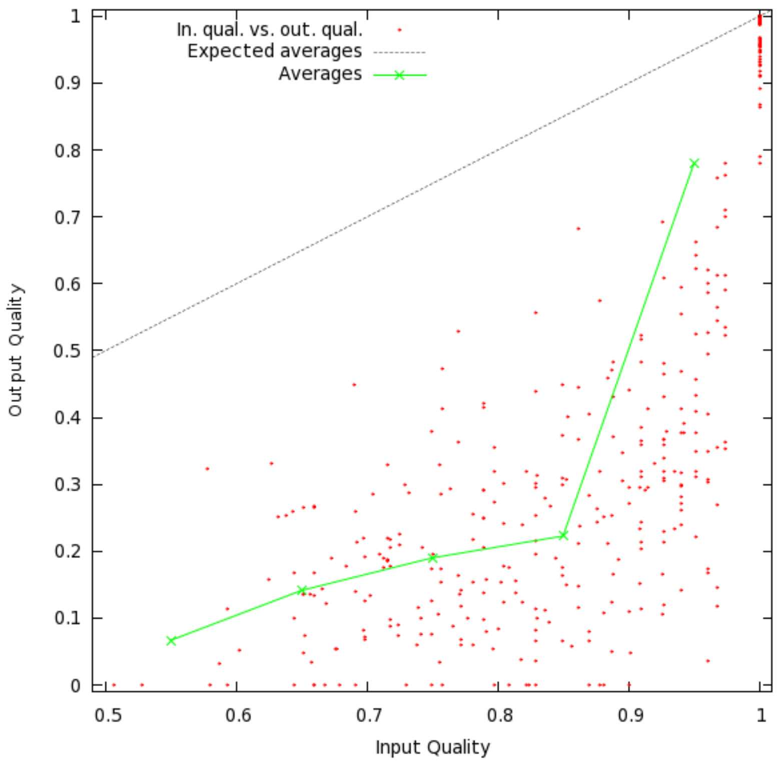

Figure 4.11: Input vs. output quality for corrupt addresses at 1000ms intervals.

To confirm the hypothesis of inter-cycle interference and attempt to address the resulting problems, the experiment was repeated with a clock period of 1000ms to allow the neurons in the network more time to return to a resting state and thereby 'forget' the past network activity. The results are shown in figure 4.11. For address qualities of less than 0.9 data quality was much the same as in the previous trial but for addresses of quality greater than this a significant

[4]A brief trial showed that randomly generated counterparts to a sequence of 500 set codes had an average quality of a little less than 0.04.

improvement can be seen: most perfect quality addresses returned data qualities of at greater than 0.85 and the average quality was 0.1 better than the previous trial.

So, a degree of associative ability has been demonstrated by the SDM. However, the high mean data quality of 0.78 for address quality of > 0.9 can largely be attributed to the data quality returned by perfect addresses: if the perfect address inputs are disregarded the mean data quality in this quality bracket is 0.38. The following chapters determine whether or not this is sufficient to address the inverted pendulum problem.

4.3.2 Spatial and Temporal Activity Distribution

It was expected of the SDM that the level of network activity from one cycle to the next should remain constant under continuous operation and that this activity should be distributed evenly among the neurons in each layer. To test this expectation, ten repetitions were performed of a trial in which 500 randomly generated addresses were presented at 500ms intervals to the SDM set to occupancy 0.8. The short cycle period and high occupancy were selected to strain the inhibitory circuits and see if they could continue to function correctly under relatively heavy inputs. During the trials the number of times each neuron fired and the total number of firings per cycle were recorded.

Figures 4.12 and 4.13 show the number of firings per cycle in the address decoder and data matrices respectively for one of the trials. Each trial produced almost identical data showing consistent numbers of neurons firing per cycle, as evident from the almost constant running[5] means and standard deviations: any gross increase or decrease in the activity of the network over time would have shown in the plot of the former of these metrics climbing or diving.

The even spread of network activity over time was largely reflected in the even spatial distribution of firings. Table 4.2 shows that each decoder neuron fired a similar number of times in all 5 trials and examination of the raw data showed that different neurons in each trial accounted for the excessive or moderate firing counts that caused the standard deviation to be non-zero. A slightly higher standard deviation was observed in the firing counts of the data neurons which can be explained by a difference in configuration: the decoder neurons sampled exactly 128 of the 256 address-lines but the data neurons sampled each word-line

[5]Calculated at x of the firing counts from cycle 0 to cycle x

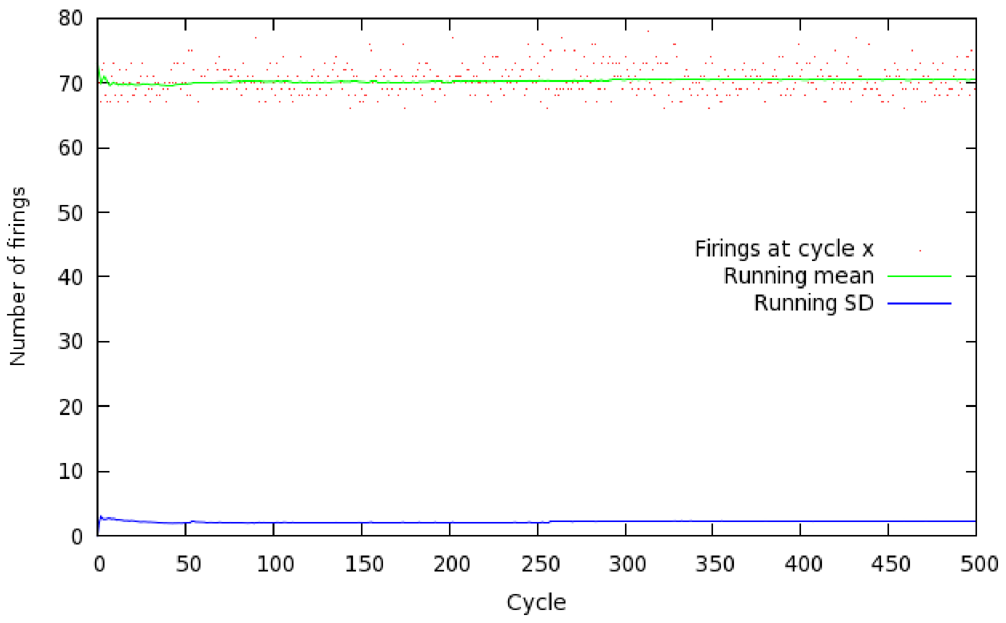

Figure 4.12: Decoder neuron firing counts over 500 500ms cycles.

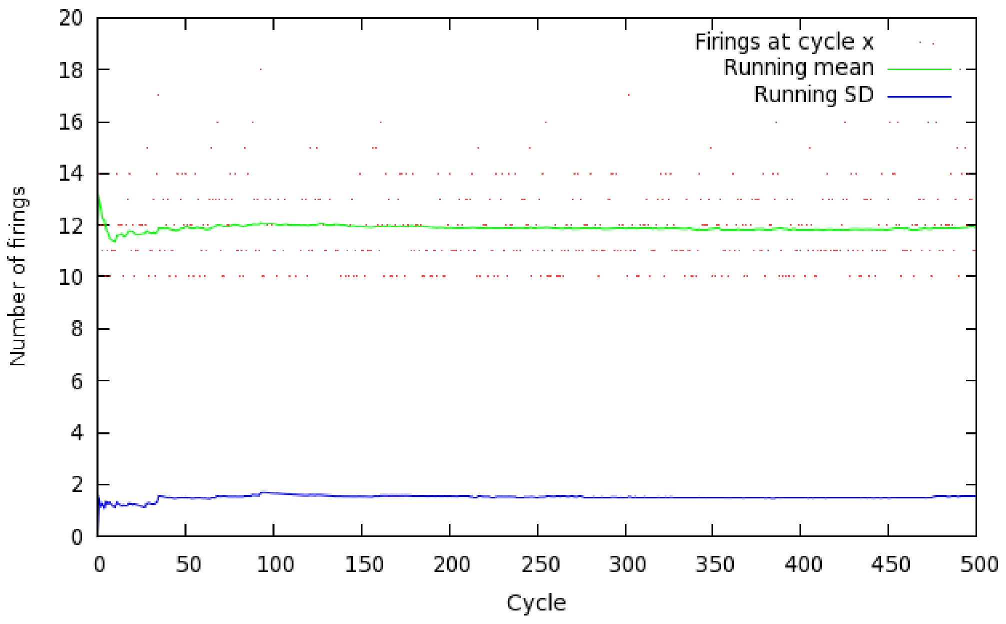

Figure 4.13: Data neuron firing counts over 500 500ms cycles.

with a probability of 0.5 so some sampled more than others. The synaptic weights in the data matrix would be adjusted in the course of the learning algorithm so this bias in the data neuron firings was not worrying.

	Per decoder		Per data	
Trial	Mean	Std dev	Mean	Std dev
0	8.593	2.978	23.046	8.751
1	8.587	2.931	23.253	8.104
2	8.596	2.958	23.160	8.559
3	8.594	2.967	23.355	7.946
4	8.602	2.974	23.277	8.380

Table 4.2: Total firings per decoder and data neuron over 500 cycles.

4.4 Summary

This chapter has reviewed a somewhat haphazard effort at constructing an SDM variant from spiking neuron models. The use of N-of-M codes in asynchronous networks has been explored—including the requirements of asynchronicity in the firings of a population, the resulting rank-order codes and the cycle separation required to give those codes meaning—and the difficulty of producing exact firing patterns in populations of neurons has been explained.

Much of the research presented in this chapter did not follow a highly structured methodology. Indeed, what is discussed here was preceded by a significant amount of time spent in undocumented and unstructured experimentation with neuron models and parameters to gain an insight into their behaviour and give direction to the presented work. As work progressed network behaviour was better understood and experimentation became more structured, concluding with trials that show successful performance of an associative memory constructed from spiking neural networks. The following chapter describes development of an inverted pendulum simulation with which to test the capabilities of this memory.

Chapter 5

Addressing the Inverted Pendulum Problem

The previous chapter documented the development of a variant of Sparse Distributed Memory which was supposed to be capable of controlling an inverted pendulum. This chapter documents the development of both the inverted pendulum simulation with which this supposition was tested and the mechanisms required to enable the SDM to interact with and learn from the simulation. As before, the chapter begins with a brief overview of the work involved and then proceeds to more detailed discussion.

Emulating Animal Learning

This dissertation is concerned primarily with examining a potentially animal approach to learning using an apparently animal model of memory. The complexity of biological neural networks and their sensors and actuators necessitates a number of abstractions in their emulation but the fundamental logic of any emulator must heed the realistic capabilities of biological processes. It is for this reason that N-of-M firing patterns were achieved in the output of the SDM in the previous chapter by means of inhibitory feed-back rather than a global variable that each neuron could read to check its eligibility to fire. For the same reason, the mechanism by which the SDM interprets, controls and learns from the state of the inverted pendulum simulation was required to respect biological capabilities.

In this research an SDM variant was tasked with learning to control the inverted pendulum in analogy to a developing animal learning to walk, so it was decided that the neural network should have no *a priori* knowledge of the physics involved in the problem. An implication of this ignorance is that the neural network also had no conception of the value of any of the pendulum states that it was exposed to, being unaware that gravity threatens a bob arcing away from its pivot more than one balancing directly above it. The network was, however, informed of its failure to control the simulation when the pendulum fell in the same way that a toddler feels a sharp bump when a misstep causes her to fall over. The learning process for the SDM began by completely filling the memory with random guesses about the correct mapping from state to action—effectively initialising the data matrix with random bits to an evenly spread occupancy of 0.5, which is the level of maximum information content—and proceeded as such: at the start of each 1000ms memory cycle the state of the simulation was encoded as an address and delivered to the SDM; the spikes propagated through the network according to the synaptic weights in the address decoder and data matrix (the guesses); the output of the data neurons was decoded as force on the cart which then moved to balance or further unbalance the inverted pendulum; if the inverted pendulum fell over the SDM was forced to reassess the relevant guess by delivery of a shock that permuted the weights on the last-active word-lines and the simulation was reset to begin again; otherwise, no shock was delivered and the simulation proceeded with delivery of the next (state encoding) address. By this *brutal* learning approach it was expected that the SDM would discard those guesses shown to be incorrect and, by generation of new guesses through permutation, eventually learn the appropriate mappings from state to action to balance the inverted pendulum indefinitely.

The Action Attribution Problem

Controlling the inverted pendulum, like walking, usually requires sequences of correct actions to be executed to maintain a stable state. A single incorrect action, such as stumbling in the walking analogy, may create a state that is impossible to recover from regardless of how well chosen the subsequent actions are. This gives rise, when applying a learning agent to a control task, to the problem of attributing a loss of control to one or more actions in a sequence in order to ensure that those actions are not repeated. In terms of the work described here, this is the task of deciding which of the recently accessed word-lines of the SDM should be permuted when a shock is received. For an agent that has no understanding of the value of a state and memory only for mappings between state and action this problem appears to be very difficult. Anderson [2] describes a solution in which an auxiliary neural network learns the value of inverted pendulum states through exposure to the dynamics of the control problem, establishing a relationship between states and the likelihood of an imminent loss of control. This network can then be used to evaluate actions based upon whether the resulting states are safer or more precarious than the preceding ones.

The use of an auxiliary network to infer action correctness is congruent with the desired ‘pure’ neural approach to the inverted pendulum problem but the idea was discarded in this research due to the complexity of adapting its structure and learning mechanisms for use with Sparse Distributed Memory. Instead, the action attribution problem was sidestepped by manipulation of the simulation parameters: gravity was reduced to a fraction of that experienced at the Earth’s surface; loss of control was considered to have occurred if the pendulum swing exceeded ± 0.2 radians (as suggested by Anderson); the track was made wide enough to minimise the possibility of the cart falling off the end; and the network output was afforded sufficient strength to move the cart and rebalance the pendulum even from the extreme range of its movement.

5.1 Simulating the Inverted Pendulum

A simple pendulum [21, 24] employs a frictionless pivot from which a rigid, massless rod is afforded a single degree of freedom to swing according to the effect of gravity on a point mass (the ‘bob’) hung on its end. Figure 5.1 shows the forces acting on a bob of mass m attached to a rod of length l hanging at an angle of θ from vertical. Gravity exerts a force of mg on the bob which is resolved into $mg \cos\theta$ (counteracted entirely by the tensile strength of the rod) and a restoring force of $-mg \sin\theta$ acting tangentially to the bob’s arc.[1] Given knowledge of this force, Newton’s second law gives the bob’s acceleration.

$$\begin{aligned} F = ma &= -mg\sin\theta \\ a &= -g\sin\theta \end{aligned} \tag{5.1}$$

The swing of a simple pendulum can computed by numerical approximations and making the further assumption that the bob acceleration is constant during

[1]The negative sign accounts for the perpetual opposition of the sign of the force and the sign of θ which explains the pendulum’s oscillatory motion.

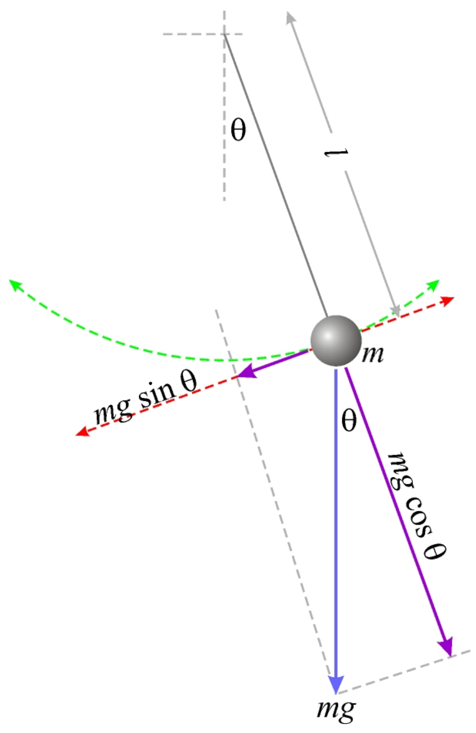

Figure 5.1: The dimensions of and forces on a simple pendulum [24].

each computed 1ms time step, velocity v is obtained from acceleration [3] as

$$v(t) = v(t-1) + a(t) \tag{5.2}$$

and the bob's position x in its arc is in turn given by

$$x(t) = x(t-1) + v(t-1) + \frac{1}{2}a(t) \tag{5.3}$$

If a simple pendulum is stable due to the restoring force applied by gravity, the inverted pendulum [23] in figure 5.2 is unstable due to the same force[2] and must be balanced by applying force to the cart on which the pivot is mounted. The same assumptions are made for the inverted pendulum as for the simple pendulum with the additional arguments that the cart's wheels are frictionless and the cart is sufficiently heavier than the bob that the latter exerts no significant force upon the former.

[2]In the inverted pendulum problem θ is typically measured from the skyward axis and so the tangential force that causes the pendulum to diverge from this axis is given by $mg \sin\theta$.

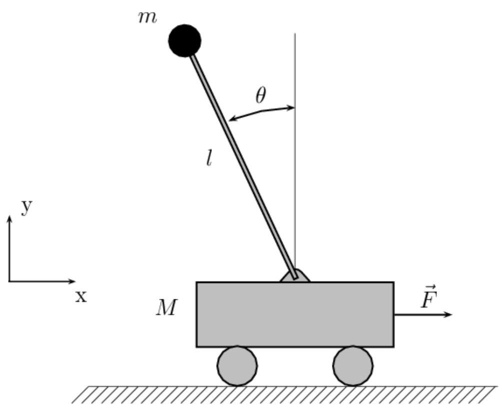

Figure 5.2: The inverted pendulum [23].

The precise swing of an inverted pendulum depends simultaneously upon accelerations due to gravity and force applied to the cart but an approximate representation may be arrived at by calculating these two effects consecutively. Such an approximation was considered adequate to create a learning environment for the SDM, given that consistent behaviour and predictable responses to inputs were more important than precise physical realism. The Python code listed below was called to calculate the approximate motions of the cart[3] and bob by this method for every 1ms of simulation time, returning `False` to the caller if the cart ran off the end of the track or the pendulum fell over.

```
# ACQUIRE CURRENT X POSITION OF BOB BEFORE MOVING CART
bob_x = cart_position + math.sin(theta) * rod_length

# CALCULATE MOTION OF CART
cart_acceleration = sum(SDM_impulses) / cart_mass
cart_position = cart_velocity + cart_acceleraton / 2
cart_velocity += cart_acceleration

# CALCULATE EFFECT OF CART MOTION ON THETA
bob_x += cart_velocity * math.sin(theta)
opp = bob_x - cart_position # OPP = X DIST. FROM BOB TO VERTICAL
theta = math.asin(opp / rod_length)
```

[3]The cart was driven by output from the SDM accumulated in the `SDM_impulses` variable between calculations.

```
# CALCULATE EFFECT OF GRAVITY ON THETA
bob_acceleration = gravity * math.sin(theta)
arc = rod_length * theta + bob_velocity + bob_acceleration / 2
bob_velocity += bob_acceleration
theta = arc / rod_length # Store effect of gravity on bob

# RETURN FALSE IF CART OR BOB ARE NOW OUT OF BOUNDS
if abs(cart_position) > track_bound or abs(theta) > theta_bound:
    return False
else:
    return True
```

The constants and variables that persisted from one invocation of this code to the next (effectively the simulation state) are listed with their initial values in table 5.1. The total track length was measured between the negative and positive bounds, coming to 1024 metres centred upon metre 0 where the cart was positioned at initialisation. Likewise the range of θ was measured negative to positive, centred at 0 with the rod exactly vertical. The speed of the simulation dynamics was controlled in part by greatly reducing the strength of gravity to be appropriate for the cycle period of the SDM, that is, to allow the SDM sufficient time to read and react to the pendulum before it fell.

Name	Value	Invariant?
Track bound	512.0 m	Yes
θ bound	0.2 rad	Yes
Gravity	0.5 m/s^2	Yes
Cart mass	1024.0 g	Yes
Cart position	0.0 m	No
Cart velocity	0.0 m/s	No
Rod length	100.0 m	Yes
Theta	0.01 rad	No
Bob mass	1.0 g	Yes
Bob velocity	0.0 m/s	No
SDM impulses	[ ]	No

Table 5.1: Initial parameters for simulations of the inverted pendulum problem.

5.2 Passing Control to the SDM

An interface was required between the network and dynamical simulations to allow the SDM to sense and control the state of the inverted pendulum. The network was delivered the simulation state by means of an encoding of the variables listed in table 5.1 in a rank-order address and cart control was in turn provided by a decoding scheme which interpreted data neuron output as lateral force. The shock mechanism for permuting recently accessed locations was implemented by attributing decoder and data neurons with a leaky memory variable that was set high on firing and thus recorded activity for some time afterwards.

5.2.1 Reading the Inverted Pendulum State

The constraint of biological plausibility applied throughout this dissertation was carried into the method of encoding the inverted pendulum state for delivery to the SDM. Expressivity and associativity of encoding schemes was also a concern so that any state might be adequately represented in a rank-order N-of-M (11-of-256) address and also that two similar states might produce two similar addresses and thus two similar data matrix outputs.

An Optical Approach

An optical approach emulating motion-blur experienced in human vision was taken to represent the cart and bob dynamics. For example, a candidate encoding for the one-dimensional dynamics of any object might be to divide a fixed 1D field of vision spanning the potential range of movement into M chunks corresponding to the SDM address-lines. At *read time* t, the first address spike might be delivered on the line corresponding to the position of the object in the visual field at t; the second spike might be delivered on the address-line corresponding to the object's location at $t-1$; the third spike at the object's position at $t-2$ and so on until N spikes have been delivered. This approach requires, of course, a history of visual observations to be maintained but has the benefit of representing up to the $N-1^{\text{th}}$ derivative of the object's position, so it is highly expressive.

Figure 5.3 plots the significance vector representation of an 11-of-256 address produced by this method from an object observed at read time at metre 112 of a 1024 metre track. This meter falls into the $112/(1024/256) = 28^{\text{th}}$ chunk of the track, so the position of the object at t is represented by the first address spike on

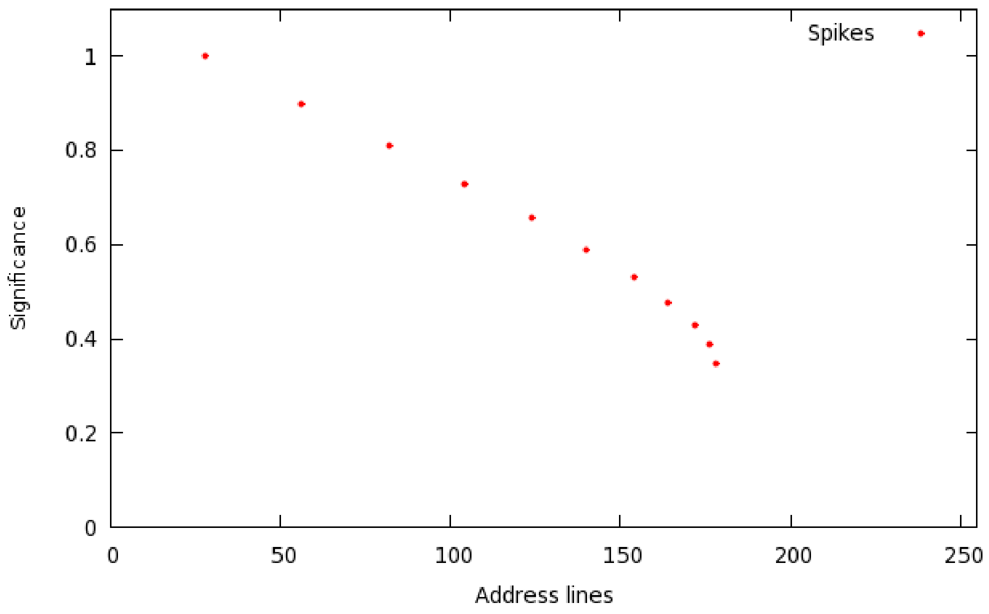

Figure 5.3: Significance vector representation of a fixed-field motion-blur address.

line 28. Examination of the history shows that the object's position at $t-1$ was metre 226 so the second spike is delivered on line 56 and so on. The final spike on line 178 represents the object's position 11 seconds prior to the read at metre 712. It may be established from the information presented in the address that the object accelerated leftwards from a stop at metre 712 at a constant 12m/s^2 for the duration of the history.

An improvement may be made to this scheme by freeing the visual field, and thus the address, to centre upon the object at each read time—with the width of the field specified by the greatest likely range of movement between two read times—so that historical positions of the object are represented only relatively to its position at t. If the object's absolute position at t is of little concern then the first address spike might be delivered on the line corresponding to its position at $t-1$ relative to the position at t, the second spike on the line corresponding to the relative position at $t-2$ and so on. This approach has the benefit of producing two identical addresses for two objects moving with the same velocity through different areas of space, thereby associating the two behaviours.

Figure 5.4 shows the object dynamics demonstrated above, encoded using the relative-field scheme. In this example, the position of the object at t is implicit in the centre of the address and the position relative to this at $t-1$ is delivered

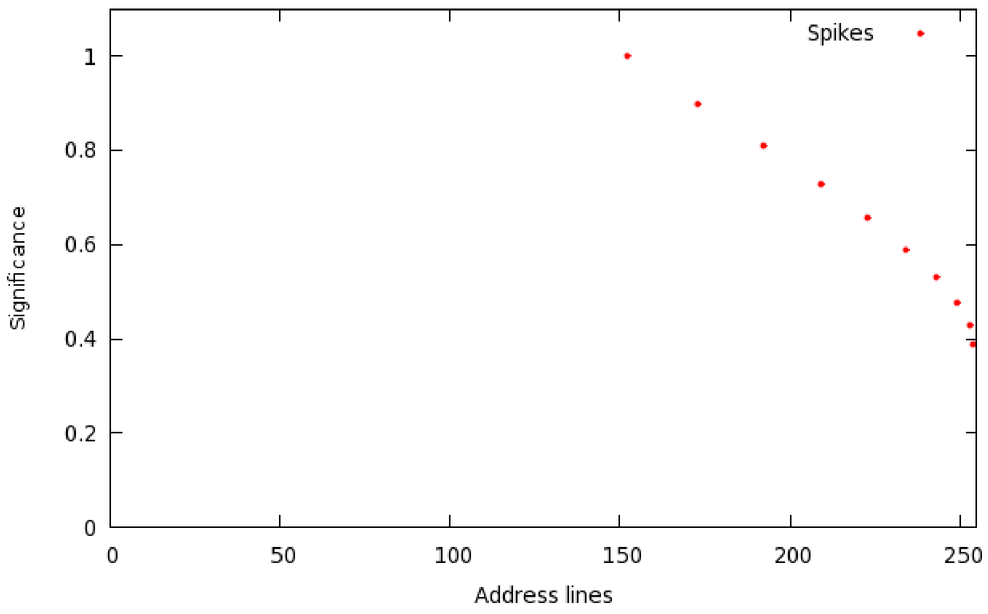

Figure 5.4: Significance vector of a relative-field motion-blur address.

on an address-line calculated by

$$\frac{M}{2} + (pos_t - pos_{t-1}) * (\frac{M}{2}/likely\_range) \tag{5.4}$$

The first term, $\frac{M}{2}$, gives the centre of the address and the rest of the equation establishes the relative position of the object at $t-1$ and maps it onto the available address-lines by scaling up or down depending on the likely range of movement.[4]

If an object's velocity does not change direction during the history retrieved at read time, as is the case in the example given here, the relative-field approach fails to exploit the full address space to encode the object dynamics. Figure 5.4 demonstrates this in the 'motion blur' originating to the right of the address centre and trailing away in that direction, leaving the left half of the address unused. To overcome this waste of expressivity, the 'blur' may be scaled to enploy the full address space and be wrapped around using modulo operator:

$$(\frac{M}{2} + (pos_t - pos_{t-1}) * (M/likely\_range))modM \tag{5.5}$$

[4]Established in this artificial case as 600m by the first example.

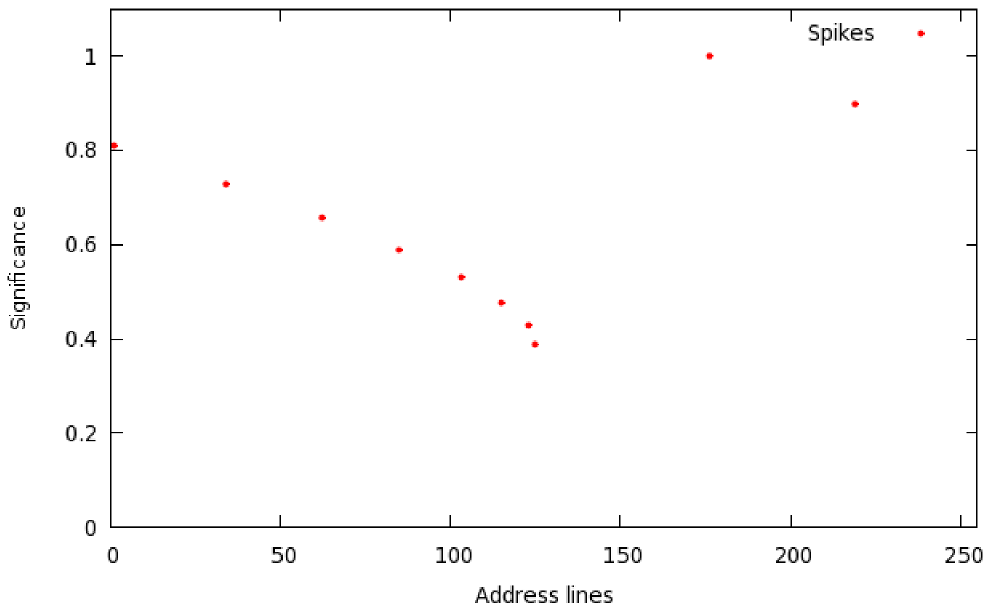

Figure 5.5: Significance vector of a wrapped-relative-field motion-blur address.

Figure 5.5 shows the effect of this final improvement, with the motion blur originating as before the the right of the address center (around address-line 175) and then leaving the right hand end of the address and continuing at the left.

The Implementation

To encode the cart and bob dynamics the address inputs were divided into two sections corresponding to two hypothetical, parallel, one-dimensional visual fields: one at a height to read the cart's pivot and one to read the rod just below the bob.[5] At read time t the read strips at their respective heights were centred on the cart's pivot (figure 5.6) and every 25th history state was extracted from a cyclic buffer holding records of the previous 500 states.

It was expected that the cart dynamics would be simpler than those of the bob and so the former were represented on 64 address-lines (one quarter of the address space) exactly according to the scheme described above. The bob dynamics were encoded on the remaining 192 address-lines also using the pivot's position at t as the address centre since it was necessary to know the bob's position relative to

[5]Reading the position of the bob directly was not possible as it moved in two dimensions. Instead, the top read strip was positioned just low enough that it could always expect to be intersected by the rod given the bound on the range of θ imposed in the simulation. The bob position was then be effectively inferred from the reading.

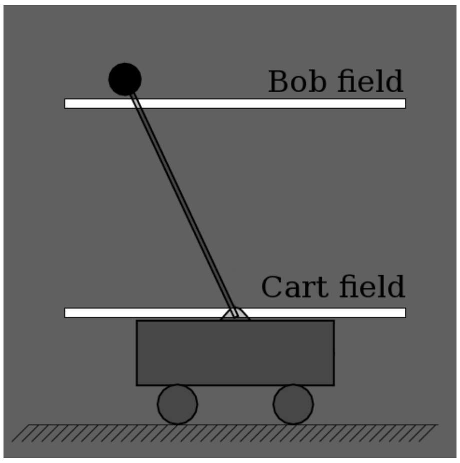

Figure 5.6: Visual fields centred upon the pivot giving positions of cart and bob.

the cart to infer θ and so acceleration due to gravity. For this reason the report omitted for the pivot's position at t was included for the bob's position because it could not necessarily be represented implicitly in the centre of the visual field.

Figure 5.7 shows an example of an address obtained from a cart moving to the right at 20m/s attempting to rebalance a bob falling at 25m/s relative to it. The leftmost 64 address-lines encode the cart dynamics relative to its position at read time and the rightmost 192 lines encode the bob dynamics relative to the exact same point. Since these encodings occur simultaneously and independently more than N spikes are present in the address so before delivery to the SDM the most significant N are 'skimmed off' and the rest discarded.

In the case that multiple spikes were reported on the same address-line, as a consequence of either the cart of bob not changing position significantly during the history, their average significance value would be taken before 'skimming'. This had the effect of minimising the presence in the address of the static object as figure 5.8 representing a static cart and a bob accelerating from rest at $\theta = 0.01$ under gravity of 1m/s^2 shows.

The likely range of movement required to map object locations onto address-lines was established by positioning the bob at the extreme right of its range, driving the cart at a speed such that the bob arrived at the extreme left of its range 1000ms (the duration of the SDM clock period) later and measuring the distance the cart travelled.

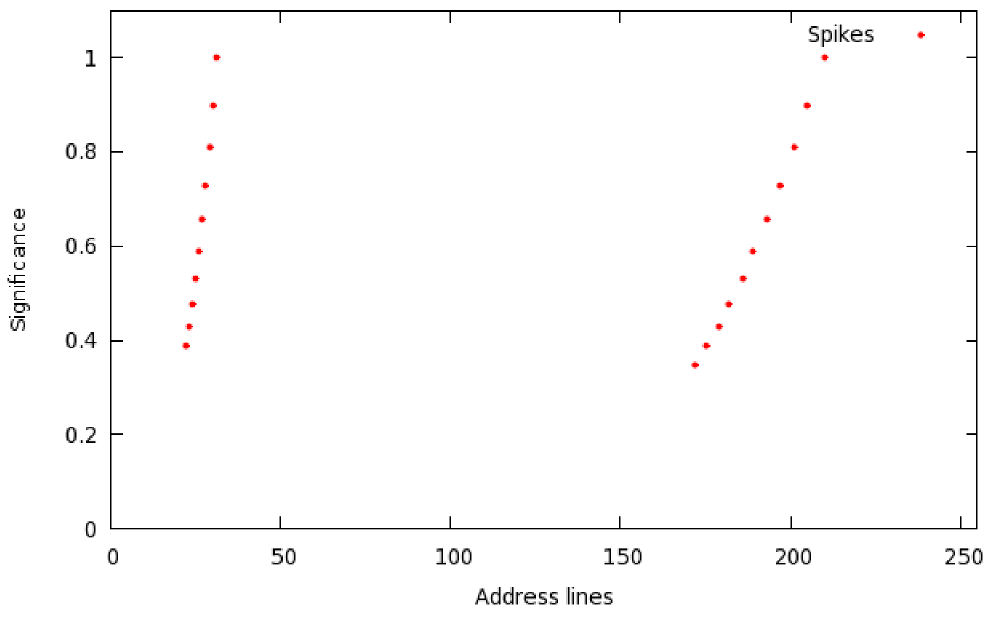

Figure 5.7: Significance vector representation of a mobile cart and bob.

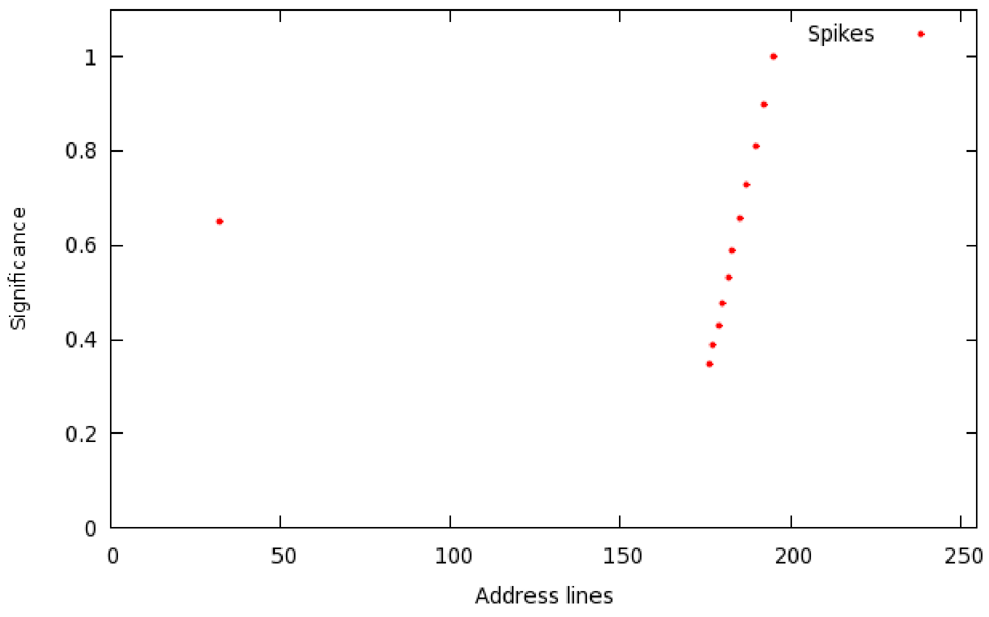

Figure 5.8: Significance vector representation of a static cart and mobile bob.

5.2.2 Driving the Cart

To drive the cart an interpretation of the SDM output as force was required that was both tolerant of the variations in the d-of-D firing patterns of the data matrix and observant of the rank-order of those patterns. Such an interpretation was developed by attributing to the output of each data neuron an impulse of variable size and one of two directions. The data neurons were indexed from $-\frac{M}{2}$ to $\frac{M}{2}$ (-128 to 128): those with negative indeces drove the cart to the left and those with positive indeces to the right; the larger the absolute value of the index the greater the impulse the neuron would apply in its respective direction.

Observation of the rank-order of data neuron spikes was made with a leaky *drive rank* variable with a resting level of 1. When a data neuron spiked, the size of the impulse attributed to it was divided by twice the drive rank squared before application to the cart and the drive rank was incremented by 1. In the time steps between spikes the drive rank value would decay with a time constant of $\tau = 10$ms.[6] In addition to respecting the rank-order of data neuron spikes this approach minimised the effect of variations in the d-of-D code produced by the data matrix by according less significance to later spikes.

Practically, data neuron spikes tended to be grouped in time such that their effect appeared to be nearly simultaneous relative to the dynamical simulation clock, although later experiments will show that their timings were significant. Spikes could also be evenly distributed between each half of the data neurons resulting in no net force upon the cart.

5.2.3 Punishing the Network

In the event of loss of inverted pendulum control (either the cart or bob running out-of-bounds) the immediately preceding action was invariably held accountable for the reasons described under *The Action Attribution Problem*. The actions produced by the SDM in response to input, according to the interpretation of the output described in the preceding section, were dependent upon the synaptic weights between the active word-lines and the data neurons so shocking the SDM to prevent repetition of an inappropriate action involved the permutation of these weights.

[6] A plot of the drive rank variable would look a great deal like the activation level plot of the leaky integrate-and-fire neuron in chapter 2 with the exception of the spike-and-reset mechanism.

Each decoder neuron was accorded a leaky *memory* variable which was set to 1000mV upon firing and then decayed with a time constant $\tau = 400$ms. When the SDM was shocked, the decoder neurons were examined to find those which had recently fired (that is, with a memory variable at > 80mV) and the synapses on the corresponding word-lines were flipped with a specified probability. In this way locations involved in generating incorrect mappings from state to action were repeatedly permuted until they ceased to do so.

5.3 Summary

This chapter has described how the inverted pendulum problem may be used to emulate an animal approach to learning. The dynamics of a simple inverted pendulum have been given along with code to simulate them in Python. Finally, the novel mechanisms with which to interface the dynamical and neural simulations have been described and an implementation of a brutal learning approach proposed.

The following chapter is the penultimate of this dissertation and demonstrates the limited success of Sparse Distributed Memory applied to the inverted pendulum problem using the brutal learning approach.

Chapter 6

Experimentation and Analysis

Sparse Distributed Memory was employed as a learning agent in the inverted pendulum problem for its ability to recognise associated inputs and respond with similarly associated outputs, thereby solving the intractable problem of storing in RAM a mapping from every inverted pendulum state to a controlling response. Learning proceeded with a punishment scheme (the *brutal* approach) designed to enable the SDM to acquire control over the inverted pendulum with no *a priori* knowledge of the physics involved.

If fixed initial conditions and parameters that circumvent the action attribution problem are specified then a sequence of actions to balance an inverted pendulum may be found by depth first search. Figure 6.1 shows how an SDM subject to a brutal learning approach might explore the search space:

The initial state is encoded as an address and presented to the SDM which returns a guess at the action to take based upon the (randomly initialised) synaptic

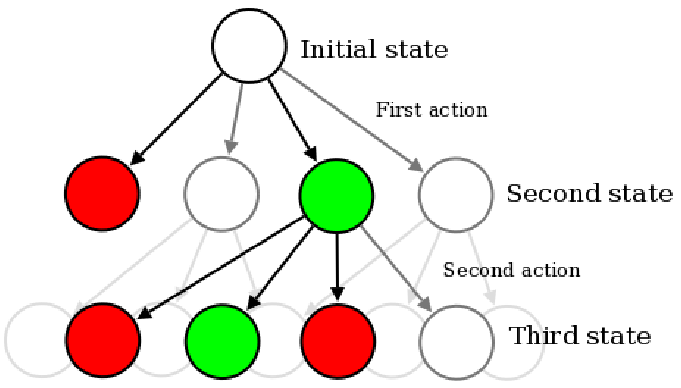

Figure 6.1: An action chain in the inverted pendulum problem.

weights on the word-lines accessed by the address. The guess proves to be incorrect so a failure state is reached (state 1 in red) and the synaptic weights involved in the guess are permuted. The simulation is reset and on the second attempt a success state (state 2 in green) is reached, indicating that the SDM balanced the pendulum for the duration of the memory cycle. The simulation then continues: the SDM is delivered an address encoding state 2 and makes second guess which proves to be incorrect resulting in a further failure state (state 3 in red). Again the simulation is reset and the synaptic weights involved in the incorrect guess are permuted. However, the SDM has now *banked* its first guess, knowing implicitly that the chosen action was correct by the absence of a shock in the cycle in which it was executed. Consequently, on every subsequent presentation of the address encoding the initial state the SDM will return the same action and reach the following success state. The same can be said for successive banked action guesses and after sufficient time exploring the search space the SDM should form an *action chain* to balance the inverted pendulum from the initial state to the depth explored.

The maximum length of an action chain that can be constructed by the SDM is both an indicator of its ability to learn in a free environment in which the initial conditions are not fixed and a basis for performance comparison with Random Access Memory. RAM may equally well store action chains in addresses pointed to by encodings of successive inverted pendulum states since the execution of one banked action deterministically adjusts the simulation state to point to the next. So although RAM is an inappropriate data store for learning in a free environment it may construct action chains (using the same brutal learning approach) of a length influenced by the size of the memory and the resulting probability of contention for locations.

Initial trials planned to show that Sparse Distributed Memory and the brutal learning approach could construct action chains of significant length and subsequent trials intended to test the SDM's learning and association performance in a free environment in analogy to real-world learning.

Initial Parameters

Experiments were initially conducted using an SDM of 4096 word-lines accessed by an address decoder configured to fire approximately 64 decoder neurons in response to each address. Addresses encoding inverted pendulum states were presented at the beginning of each 1000ms cycle (starting at $t = 1$ second) in rank-order 11-of-256 form, with 2ms intervals between each spike, and the resulting data neuron firings were controlled into 11-of-256 patterns and used to drive the cart. On shocking the SDM each of the synaptic weights on the last-active word-lines were flipped with a probability of 0.05. The initial binary weights on the decoder and data neurons were set pseudo-randomly and regenerated between each trial. All other SDM parameters (leaky time constants, thresholds and so on) were given by the conclusions of the experiments in chapter 4.

Gravity was set to 0.5m/s^2, the bound on inverted pendulum angle was set to ± 0.2 radians and the track bounds were placed at ± 512 metres. In the experiments on building action chains every run commenced with the cart at metre 0 and the pendulum at 0.01 radians from vertical. In the free environment the starting positions and velocities of objects was generated randomly for each run.

6.1 Control Experiment

A control was established against which to test the performance of the SDM in the following trials. The control was chosen on the assumption that the most probable worst-case performance of the SDM in attempting to balance the inverted pendulum would be if its data outputs were random and bore no relation to its address inputs. As such, a series of five trials were conducted as described above with the exception that each data neuron firing was observed only for its timing: its index, which typically would have dictated the force applied to the cart, was substituted with one popped from a stack containing a random shuffling of all of the data neuron indeces. The stack was repopulated and reshuffled on presentation of each new address.

Figure 6.2 shows the time for which the inverted pendulum was balanced in 250 runs across 5 trials using this random output scheme. The inverted pendulum was released at $t = 0$s and the first state-encoding address was presented to the SDM at $t = 1$s. Failing any movement of the cart the bob would fall under gravity and reach its boundary at $t = 1.651$s causing the simulation to be reset. Should the cart movement balance the inverted pendulum until $t = 2$ the next state-encoding address was presented and the simulation continued.

Performance in the control was poor with most runs lasting no longer than 2 seconds, indicating unsurprisingly that random SDM output does not control the

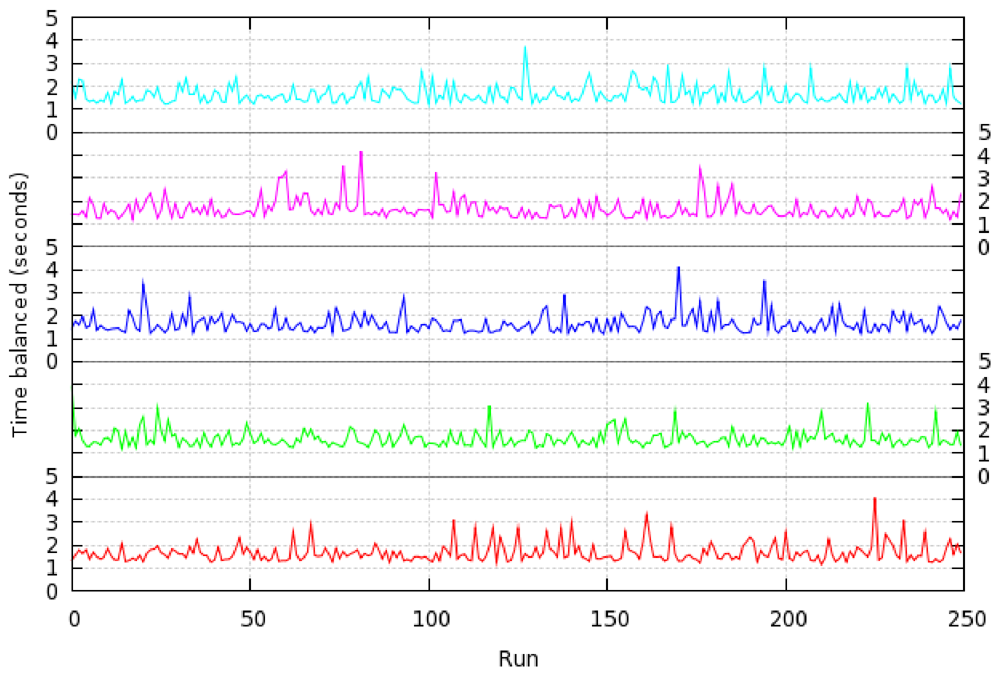

Figure 6.2: Inverted pendulum balancing performance for random SDM output.

inverted pendulum well nor results in an increase in performance in successive runs. Of those runs lasting longer than 2 seconds, the majority were ended by the second random output causing loss of control. A small minority of the runs lasted longer than 3 seconds and the most successful run lasted approximately 4. If a very approximate estimate is made from this data that each random output will rebalance the inverted pendulum with a probability of $\frac{1}{5}$ it is easy to see that the random output scheme must result in poor average performance and necessarily short best-runs.

6.2 Building Action Chains

Having established the control, the SDM outputs were reassociated with their corresponding impulses and the memory was applied the task of constructing action chains using the brutal learning scheme described above.

6.2.1 Initial Experiments

Figure 6.3 shows the performance of the initial attempts to construct action chains as an improvement upon the control. Most most runs lasted longer than

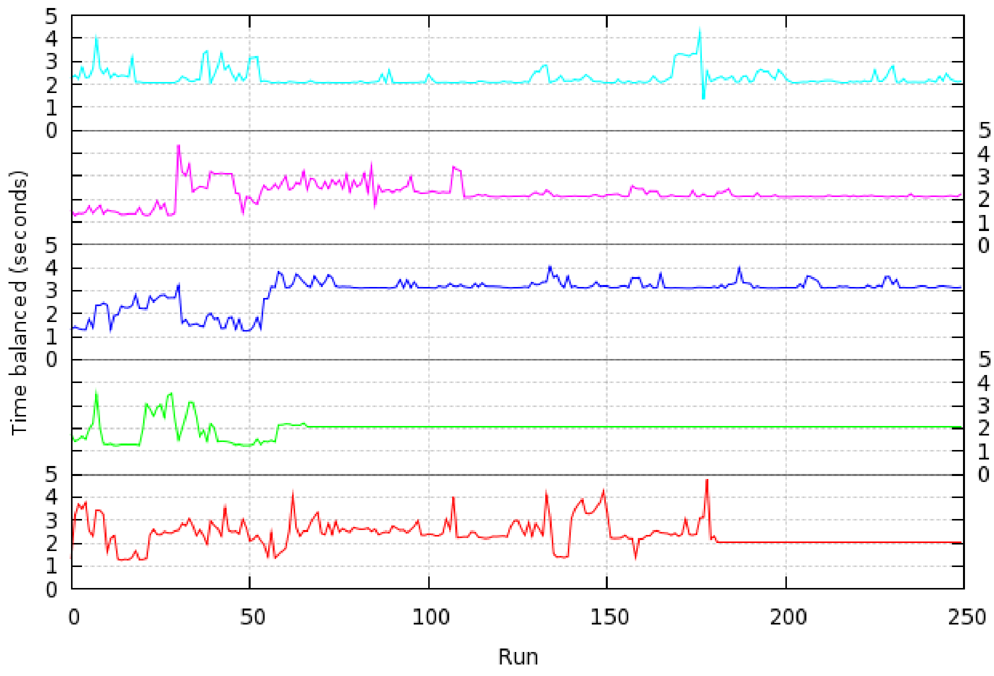

Figure 6.3: Inverted pendulum balancing performance for $w = 64$.

2 seconds[1] indicating that at least one correct action was learnt but three issues became immediately apparent:

The most obvious difference between these trials and the control is the occurrence of hard plateaus in the time for which the pendulum was balanced. These plateaus imply that the SDM was being shocked immediately after the memories of its decoder neurons had faded and just before the next set of decoders fired. If this was the case then the resulting failure to permute any of the synaptic weights in the SDM caused identical behaviour to be exhibited in the following run and so repetition of the same situation until the end of the trial.

The data also exhibits soft plateaus in which the run length varies in a range of only 1 second in as many as 150 consecutive runs. This may be attributed in part to the small proportion of synaptic weights that were flipped in response to each shock, causing little change in the SDM output from one run to the next. However, soft plateaus also reflect on the learning approach used: the SDM learns only by permutation of the last-active word-lines

[1]It should be noted that every run concluded with the bob falling beyond its bounds; the cart moved moved within only a very small span of the track and never reached either end.

when a loss of inverted pendulum control occurs, so an action that results in a run length of 2.9 seconds is considered to be no better than one resulting in a run length of 2.1 seconds. Only when a new cycle begins is the preceding action implicitly recognised as appropriate and banked safely away from further alteration.

The expected monotonic growth in the length of the action chains, and so the length of each run, was not evident and in fact the run time frequently decreased dramatically following slight increases in the length of action chains. This implies that the word-lines involved in the generation of each action in the chain were not mutually exclusive and that the permutation of the synaptic weights on one of the lines affected multiple banked actions, thus breaking the action chains.

These initial trials show that brutal learning can produce in the SDM the correct mappings from stimulus to response as hypothesised. However, the memory demonstrated a limited ability to store mappings in that upon learning one it 'forgot' another. A greater storage capacity would be required for the SDM to be capable of controlling the inverted pendulum indefinitely.

6.2.2 Adjusting w-of-W

The second set of trials sought to address the issues demonstrated by the first: the memory of the decoder neurons was lengthened slightly to avoid hard plateaus in the run lengths and the number of word-lines activated by each address (w) was reduced from 64 to 32 to limit the overlap in their usage. Since soft plateaus are an issue of learning rate rather than overall learning capacity, no changes were made to the SDM parameters to address them.

Figure 6.4 shows the effects of these changes: hard plateaus are completely absent from the data, the soft plateaus appear to sit higher and the best run is almost 2 seconds longer. Table 6.1 lists the average run lengths for the random output scheme and the trials using 64 and 32 word-lines, confirming the better performance of the latter. However, even with these improvements the average action chain runs for just two cycles and the longest chains continue to be broken apparently by permutation.

To support the hypothesis that action chains were broken by permutation of word-lines the issue was reproduced under close observation of network activity.

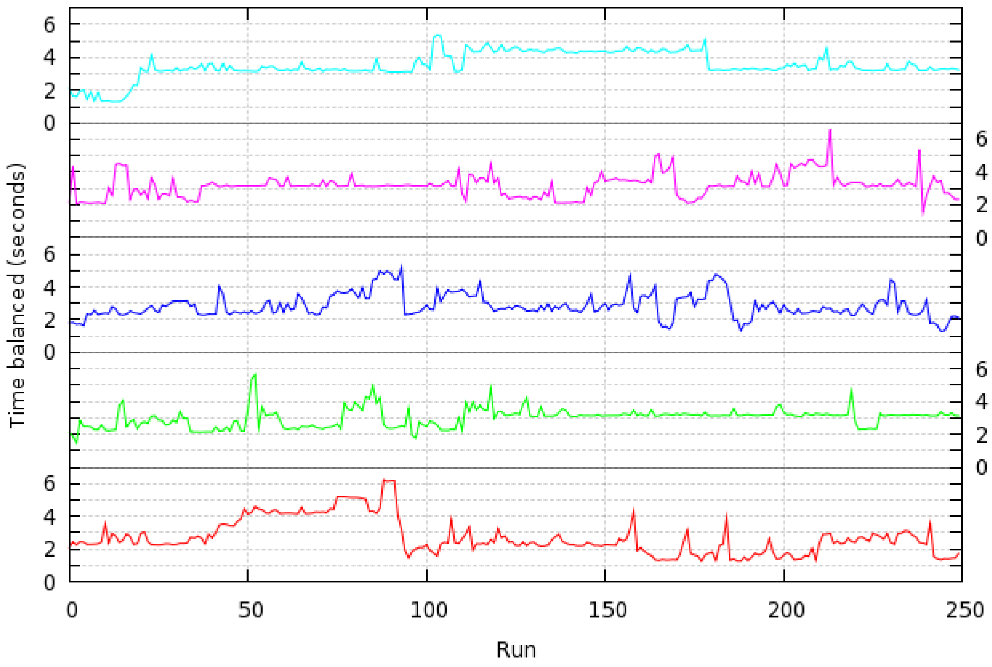

Figure 6.4: Inverted pendulum balancing performance for $w = 32$.

Table 6.2 lists the state of the inverted pendulum in two successive runs, of which the latter is shorter than the former, suggesting that permutation of the last-accessed word-lines in the first run adversely affected performance in the second. Figure 6.5 supports this argument by showing that a single location accessed and shocked in the final cycle of the first run was also accessed in the first cycle of the second run. So, in the absence of any other difference between the two trials, permutation of the synaptic weights on a single word-line appears to be responsible for the slightly different data neuron output (not shown but evident in the slightly different cart position and θ reading at $t = 2.0$s) in the first cycle

Trial	Random	64 lines	32 lines
0	1.664	2.382	2.708
1	1.636	2.021	3.000
2	1.654	2.953	2.851
3	1.669	2.217	3.181
4	1.656	2.284	3.527
Mean	1.656	2.372	3.055

Table 6.1: Average run lengths for random and learnt SDM output.

	Time (s)	Cart pos.	Cart vel.	θ	Bob vel.
Run 1	1.0	0.0	0.0	0.047	0.010
	2.0	22.68	0.024	0.023	0.026
	3.0	58.49	0.036	-0.075	0.018
	4.0	63.07	0.002	-0.130	-0.022
	4.1	64.13	0.007	**-0.200**	-0.038
Run 2	1.0	0.0	0.0	0.047	0.010
	2.0	23.84	0.025	0.000	0.022
	2.5	50.52	0.050	**-0.200**	0.000

Table 6.2: Inverted pendulum states per cycle in successive runs.

of the two runs and the resulting break of the action chain.

The significance of the adverse effect of contention for word-lines is initially surprising given that Sparse Distributed Memory is typically considered robust in this regard [17, 9, 8]. However, data is usually stored in Sparse Distributed Memory as a number of identical copies on different word-lines thereby mitigating the effect of shared use of certain storage locations. The brutal learning approach does not follow this method, instead the synaptic weights on the word-lines activated by an address are permuted until an apparently sensitive arrangement is established that produces appropriate firing patterns in the data neurons. Subsequent experiments focused on increasing the stability of the lessons acquired by the SDM by increasing the size of the memory to reduce contention and altering the learning scheme to negate its effects.

6.2.3 Stabilising the Memory

An attempt was made to enable the SDM to build significant action chains by expanding the memory from 4096 to 16384 locations to mitigate the adverse effect of contention for word-lines. w was held, as in all of the remaining trials, at 32. Such an adjustment does not address the volatility of the lessons learnt by the SDM but it was hoped that it might elicit more impressive performance from the memory. Significant computational expense was incurred by the four-fold increase in the size of the SDM and it is for this reason that larger memories were not built.

Figure 6.6 shows results much as expected: the soft plateaus again appear to sit higher than in the preceding trial and the best run is improved by some 3

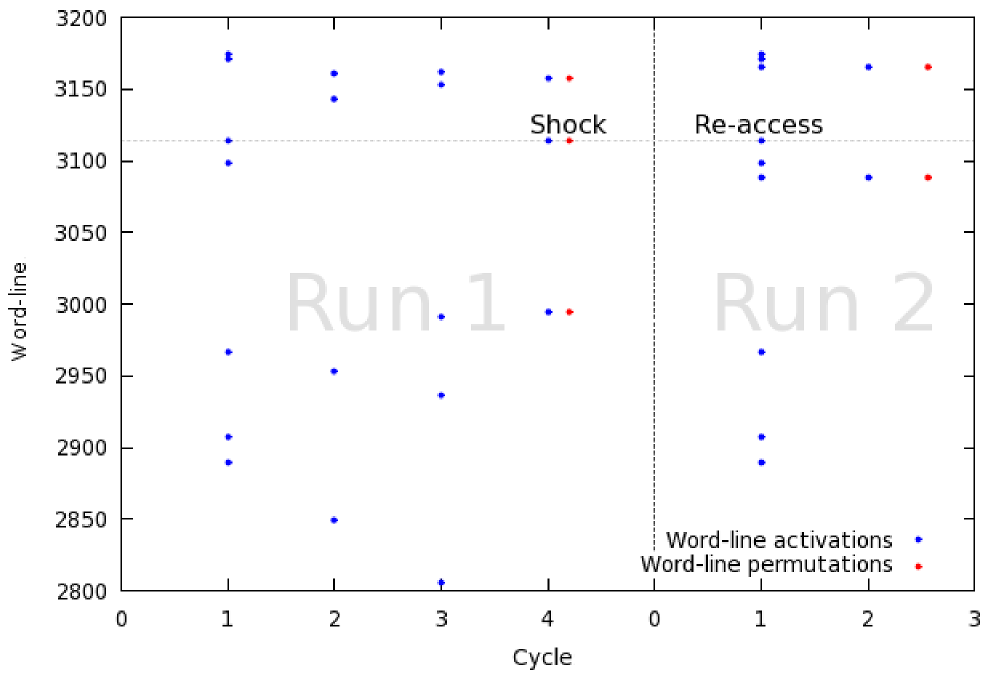

Figure 6.5: Word-line activations and permutations in two successive runs.

seconds but action chains continue to be short. The mean run lengths listed in table 6.3 confirm the improvement in average performance over the 4096 word-line SDM but in small proportion to the large increase in memory size required to achieve the results.

Aiming to reduce contention for word-lines by increasing memory size is computationally expensive and more suited to avoiding collisions in a hashmap, RAM or cache than in Sparse Distributed Memory; the latter is typically shown to be robust to contention and any use of the memory should actively exploit this. A variation on the learning scheme was developed in which shocks to the memory set the synaptic weights on all recently-active word-lines to an identical randomly generated 11-of-256 pattern (that is, the same 11 synaptic weights on each last-active word-line were set to 1 and the remainder to 0) in the hope that the distributed storage of the pattern would reinforce it against contention. The memory was initialised at occupancy 0 to simplify behaviour and the data neuron thresholds were increased to $\frac{w}{3}$mV to account for the greater input that they would receive. The size inhibitory feed-back to the data neurons was increased to $-w$mV for the same reason. Finally, the drive rank mechanism was replaced because the shock mechanism set all data neurons to receive the same input in

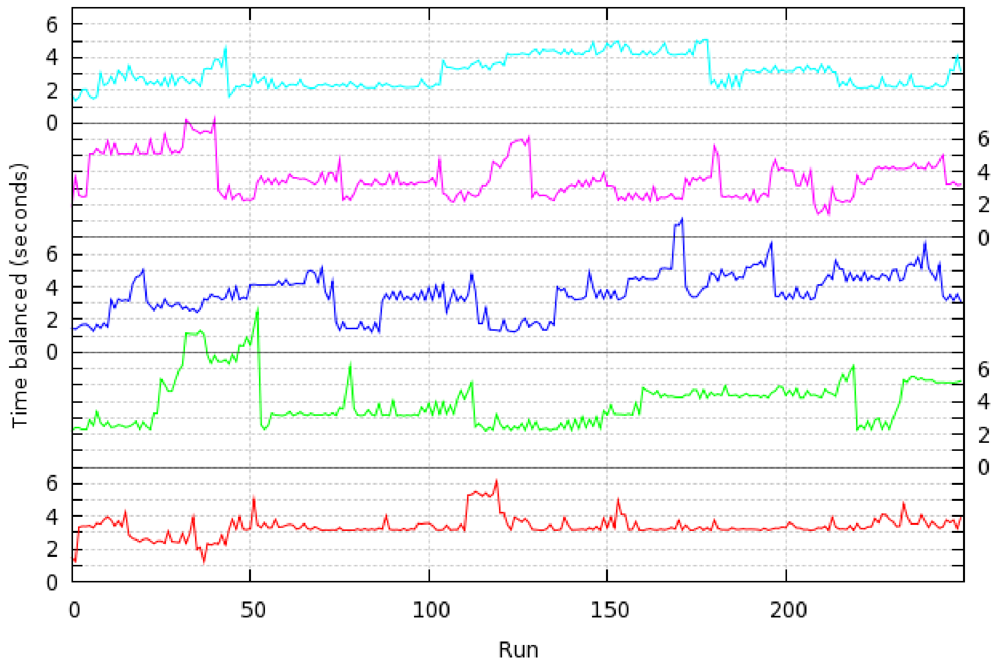

Figure 6.6: Inverted pendulum balancing performance for $W = 16384$.

response to a particular address, causing them to fire simultaneously. Instead, the size of all the impulses produced by data neurons was simply divided by 4.

The new *reinforced* variation was employed in the usual series of trials in a memory of 4096 locations. Table 6.3 shows performance close to that of the original learning scheme in the same size memory but more importantly figure 6.7 shows that the reinforced variation suffers the same issue of breaking action chains. Contention in this case had the effect not of changing *which* data neurons fired but the *timing* in which they fired: data neurons that fired simultaneously in response to a certain address would begin to do so asynchronously after varying numbers of the synaptic weights on their dendrites had been altered. The change in data neuron firing times would slightly change the dynamics of the inverted pendulum so that the following address would not point exactly to the locations of the next action and the action chain would be broken. It is worth noting that the order of firings did not need to change to break an action chain, only the intervals between them, indicating that the significance vector representations do not record all of the information transmitted in rank-order codes.

Neither variation on the brutal learning approach has been shown to be capable of constructing action chains of significant length. However, the original

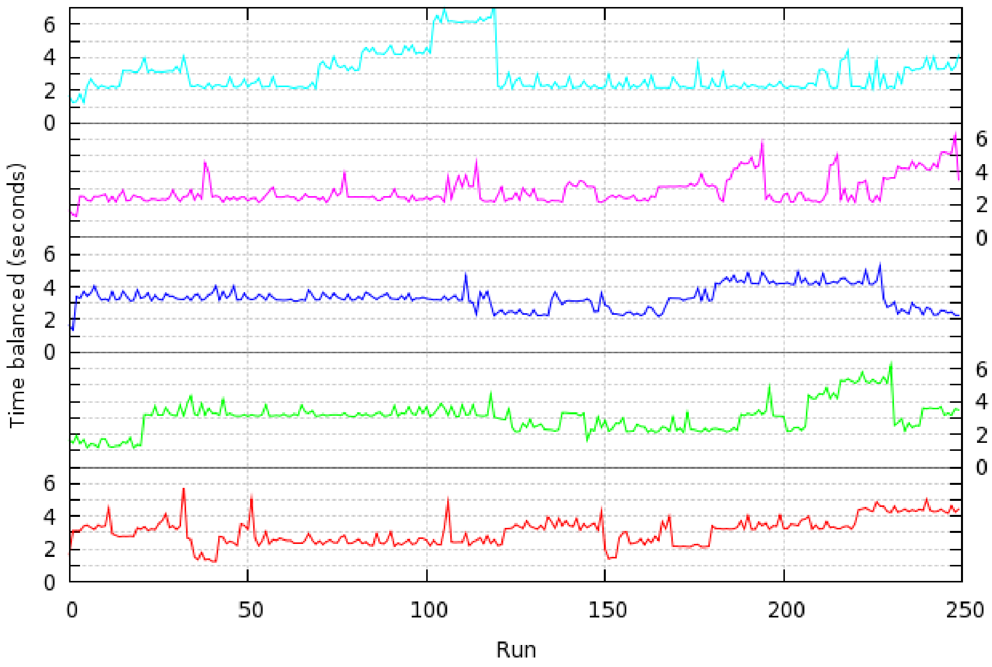

Figure 6.7: Performance with the reinforced learning variation.

variation greatly limited the memory's learning capacity by producing lessons that were unstable and easily damaged by contention; the reinforced variation, in contrast, stored the N-of-M codings of lessons in a robust manner but was unable to fully preserve the exact temporal details of the code. This is a problem when precise cart and bob motions are required to chain actions together but perhaps not when an SDM is sufficiently well-trained in a free environment to provide an appropriate action for every state, as the final experiment investigates.

Trial	32-of-4096	32-of-16384	Reinf. scheme
0	2.708	3.362	3.071
1	3.000	3.969	3.063
2	2.851	3.601	3.328
3	3.181	3.630	2.859
4	3.527	3.050	2.993
Mean	3.055	3.522	3.063

Table 6.3: Average run lengths in the initial and reinforced approaches.

6.3 Learning in a Free Environment

Real-world learning does not typically offer fixed initial conditions and the ability to reset the environment upon making a mistake. The experiments into the ability of the SDM to build action chains have been interesting, and the results enable informed analysis of the following trials, but they did not realistically emulate process of acquiring motor skills through experience.

To better reflect real-world learning conditions, experiments were conducted in which the initial state of the inverted pendulum in each run was randomly generated. The cart was positioned within the centre quarter of the track travelling left or right at up to 5m/s and the rod was positioned at an angle θ within the range ± 0.01 radians. The possible range of each initial parameter was chosen to ensure that the inverted pendulum would not fall over before the first opportunity was given to the SDM to control it at $t = 1$s.

Both variations of the learning approach were employed in turn in a memory of 16384 locations, with trials extending to 2500 runs to allow time for exposure to a range of states. The corresponding increase in computational cost limited each experiment to 2 repetitions.

The original learning variation was not expected to show any improvement in performance over time due to the limited learning capacity it demonstrated in the action chain experiments. Figure 6.8 confirms this result, showing no improvement in the running mean of the run lengths over 2500 trials.

It was hoped that the reinforced variation on the learning scheme would fare better because of the distributed storage of codes and the minimal effect (on only timing rather than ordering and indeces of data neuron firings) of contention. Figure 6.7 demonstrates that this is not the case. Again, no improvement in performance over the duration of the trials is evident and average performance is the same as that found in the control.

The reinforced variation was developed in contrast to the original and has not been subject to the same degree of analysis, so its poor performance is not so easily explained. Speculation on the results obtained here is presented alongside the other conclusions in the final chapter.

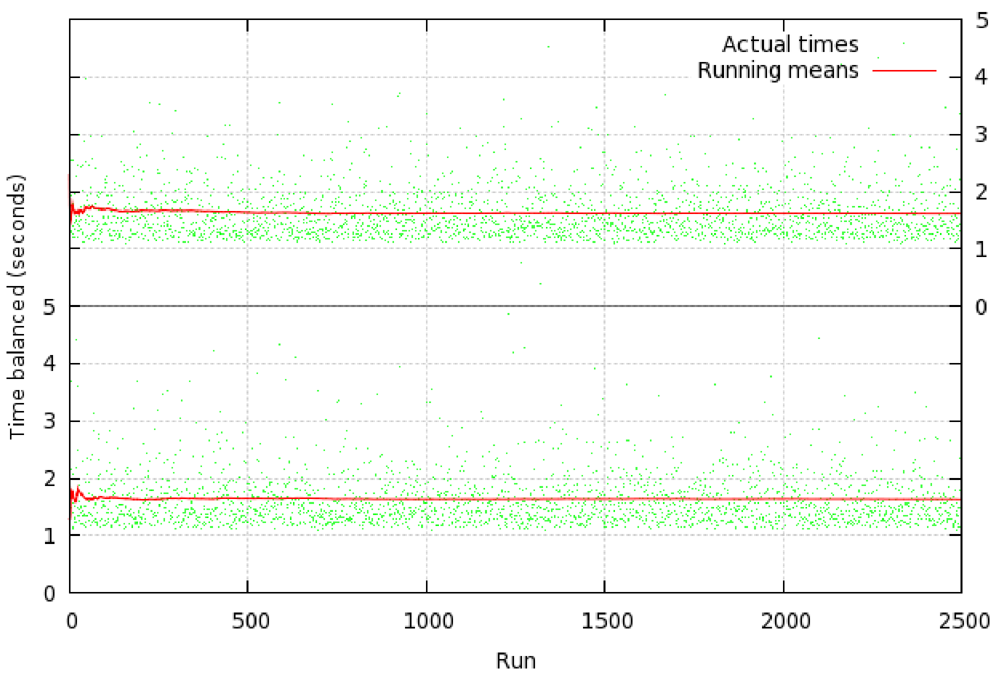

Figure 6.8: Performance of the original learning variation in a free environment.

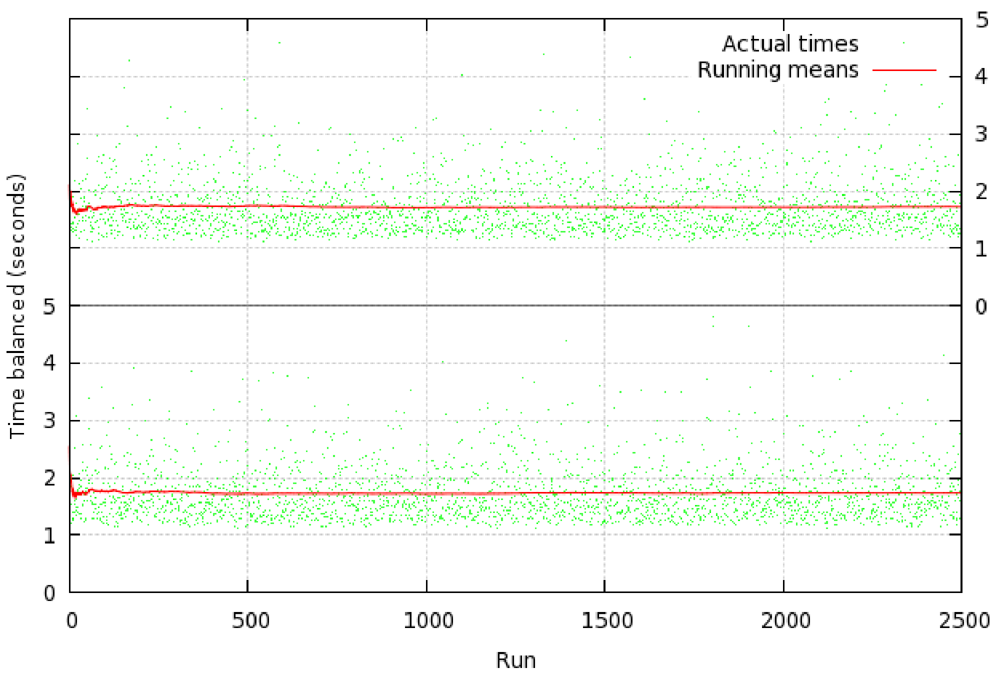

Figure 6.9: Performance of the reinforced learning variation in a free environment.

6.4 Summary

This chapter has shown that a variant of Sparse Distributed Memory may be used to briefly control the inverted pendulum following repeated exposure to identical initial conditions and punishment to correct mistakes. The permutation-based learning scheme has been seen to limit the ability of Sparse Distributed Memory to robustly store lessons and a variation on the scheme has been proposed and partially examined. Neither variation on the learning scheme showed any success in controlling the inverted pendulum when initial conditions for each run were not fixed. The following and final chapter summarises the dissertation, draws conclusions from the performance demonstrated here and presents suggestions for further work.

Chapter 7

Conclusions

This dissertation has presented a variant of Sparse Distributed Memory built from spiking neuron models and has demonstrated the limited associative ability it posses. The memory was applied to the task of learning to control an inverted pendulum with limited success conditional upon an artificially controlled learning environment. This success, however, did not exploit the associative behaviour for which the memory was employed and the attempts to do so in a more realistic learning environment failed completely.

The hypothesis upon which this research was based—that an SDM could acquire control of an inverted pendulum through punishment learning alone—has not been disproved but the particular variations on the learning approach developed were shown to be inadequate. Specifically, learning through random permutation of the synaptic weights on recently active-axons (word-lines) produces highly unstable lessons which may easily be destroyed by subsequent permutations. The result is a learning agent with memory for far fewer lessons in dynamics than appear to be required to control the inverted pendulum indefinitely.

This last statement highlights the omission from this research of investigation into *exactly how many lessons are required to control the inverted pendulum indefinitely.* The argument was initially made that if a memory could recognise an inverted pendulum state as similar to one that it had already learnt to control then it could apply the same lesson to the new state with similar effects. It was hoped that by using Sparse Distributed Memory to associate similar states in this way it would be feasible to store those lessons necessary to control any state. But the definition of 'similar' is conspicuously absent from this dissertation and without a specification for the states that can be considered alike it is impossible

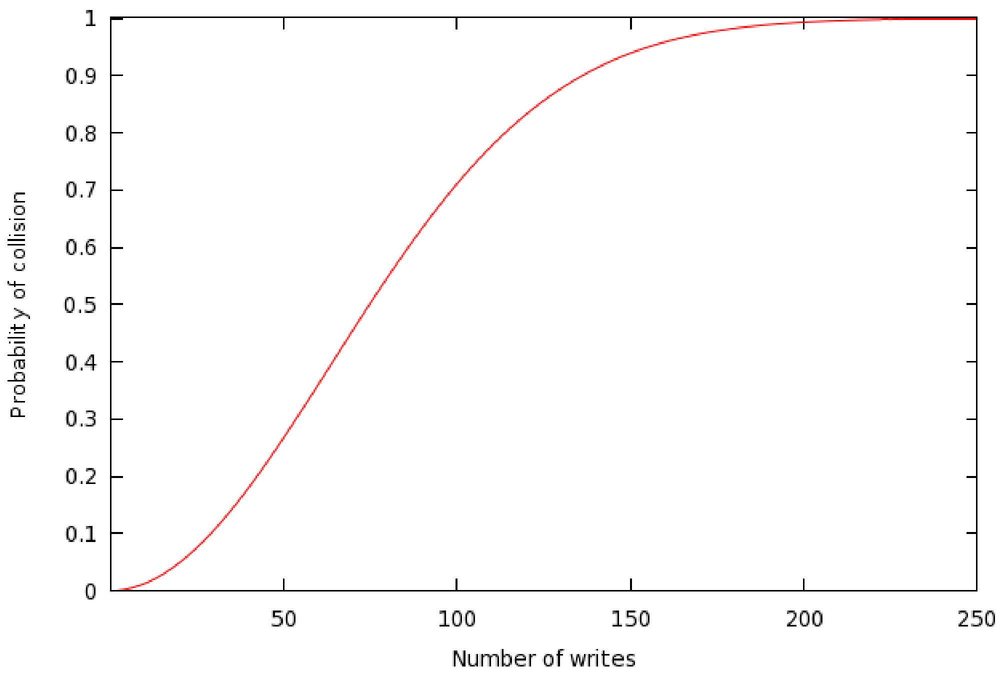

Figure 7.1: Probability of contention for storage breaking action chains in RAM.

to estimate the storage requirements of the memory. It may even be argued that the storage requirements are so great that the brutal approach using associativity is no more feasible than simply generating and storing a unique controlling action for every inverted pendulum state. Investigation of this argument may be required for research in this area to progress further.

*

This work was motivated by an interest in the disjunction between the abilities of the digital computer and the brain. Ultimately, the product of the research failed to emulate animal capabilities but the little success that was had in the action chain experiments provides grounds for further work on comparing biological and electronic computers.

As described in the previous chapter, the construction and execution of action chains does not require any associative abilities so conventional Random Access Memory is as suited to the task as Sparse Distributed Memory. Figure 7.1 suggests that a Random Access Memory of 4096 locations should greatly outperform a similar sized Sparse Distributed Memory in that the probability of contention damaging chains as long as 30 actions is below 0.1. However, an action chain executed from RAM would be broken immediately by any slight perturbation to the inverted pendulum motions such as a simulated wind. In contrast, an SDM programmed with the same action chain may be able to recover from the 'wind' by recognising the perturbed state as being similar to the unperturbed state that it knows how to control. Experiments along these lines, noting particularly the size of the perturbations the SDM can handle, may produce results properly demonstrating the advantages of Sparse Distributed Memory over Random Access Memory.

*

The Sparse Distributed Memory variant developed here is novel for its construction entirely from spiking neuron models. Pentti Kanerva's observation that the architecture of Sparse Distributed Memory is similar to structures found in the cerebral cortex was motivation for both the use of the memory in this research and the attempt to adhere to biological principles throughout the work. In practice, the difficulty of managing the complex behaviour of spiking neuron models influenced decisions to simplify network operation by, for example, using relatively simple spiking models and imposing a regular cycle on activity to avoid successive waves of spikes interfering with one another. Work into the use of highly realistic neuron models such as those developed by Izhikevich and efforts to actively exploit the interaction between successive network inputs would be interesting. Developments in these areas might also permit movement towards a model of network activity more realistic than the artificial cycles imposed here.

The suggestion that cyclic behaviour in neural networks is artificial raises questions about the model of activity in biological nets. Population coding in waves of spikes suggests a soft synchronicity to neural firings in that each wave must be somehow delimited, but rate coding, which conveys information in the frequency at which neurons fire, requires no such timing. The possibility that different coding schemes may be used in different parts of the brain was hidden in this work by a construction that spanned everything from optical input, signal processing and actuator output using just two layers of neurons firing in rank-order coded N-of-M patterns. This observation also makes the goal of emulating a complex animal learning system using such a simple network seem extremely ambitious.

Finally, learning by punishment seems to be a reasonable approach in a computational simulation, given that painful mistakes are an integral part of acquiring many gross motor skills. However, the immense complexity of the brain certainly leaves room for auxiliary networks capable of learning in parallel and contributing additional information to an agent's decisions. This suggests that, just as further research into low-level network behaviour would be interesting, examining the emergent properties of connected networks may be necessary to explain the high-level capabilities of the brain.

Bibliography

[1] Scientific American. *Scientific American Book of the Brain.* The Lyons Press, 1st edition, 1999.

[2] C.W. Anderson. Learning to control an inverted pendulum using neural networks. *Control Systems Magazine, IEEE*, 9(3):31–37, 1989.

[3] David Austin, Leah Keshet, and Denis Sjerve. Acceleration, velocity and position. `http://www.ugrad.math.ubc.ca/coursedoc/math101/notes/applications/velocity.html`, 1998. [Online; accessed 7-August-2009].

[4] G. Q. Bi and M. M. Poo. Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *The Journal of Neuroscience*, 18(24):10464–10472, 1998.

[5] Kenneth I. Blum and L. F. Abbott. A model of spatial map formation in the hippocampus of the rat. *Neural Computation*, 8(1):85–93, 1996.

[6] J. Bose, S.B. Furber, and J.L. Shapiro. A system for transmitting a coherent burst of activity through a network of spiking neurons. *Lecture Notes in Computer Science*, 3931:44–48, 2006.

[7] Peter Dayan and L. F. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems.* The MIT Press, 1st edition, 2001.

[8] S. Furber, John W. Bainbridge, Mike J. Cumpstey, and S. Temple. Sparse distributed memory using N-of-M codes. *Neural Networks*, 17(10):1437–1451, 2004.

[9] S. B. Furber, G. Brown, J. Bose, J. M. Cumpstey, P. Marshall, and J. L. Shapiro. Sparse distributed memory using rank-order neural codes. *Neural Networks, IEEE Transactions on*, 18(3):648–659, 2007.

[10] Steve Furber and Steve Temple. Neural systems engineering. *J. R. Soc. Interface*, 4(13):193–206, 2006.

[11] D. Goodman and R. Brette. Brian: a simulator for spiking neural networks in Python. *Frontiers in Neuroinformatics*, 2, 2008.

[12] Simon Haykin. *Neural Networks: A Comprehensive Foundation.* Prentice Hall, 2nd edition, 1998.

[13] D. O. Hebb. *The Organization of Behavior: A Neuropsychological Theory.* Willey, 1st edition, 1949.

[14] E.M. Izhikevich. Simple model of spiking neurons. *Neural Networks, IEEE Transactions on*, 14(6):1569–1572, 2003.

[15] Xin Jin, Steve B. Furber, and John V. Woods. Efficient modelling of spiking neural networks on a scalable chip multiprocessor. In *IEEE International Joint Conference on Neural Networks*, pages 2812–2819. IEEE, 2008.

[16] Eric R. Kandel, James H. Schwartz, and Thomas M. Jessell. *Principles of Neural Science.* McGraw-Hill Medical, 4th edition, 2000.

[17] Pentti Kanerva. *Sparse Distributed Memory.* The MIT Press, 1st edition, 1988.

[18] Pentti Kanerva. Sparse distributed memory and related models. In *Associative Neural Memories*, pages 50–76. Oxford University Press, 1993.

[19] M. M. Khan, D. R. Lester, Luis A. Plana, Alexander D. Rast, X. Jin, E. Painkras, and Stephen B. Furber. SpiNNaker: Mapping neural networks onto a massively-parallel chip multiprocessor. In *IJCNN*, pages 2849–2856. IEEE, 2008.

[20] F. Rosenblatt. The perceptron: A perceiving and recognizing automaton. Technical Report 85-460-1, Cornel Aeuronautical Laboratory, 1957.

[21] Keith R. Symon. *Mechanics.* Addison-Wesley, 2nd edition, 1961.

[22] Simon Thorpe and Jacques Gautrais. Rank order coding. In *Computational Neuroscience: Trends in research*, pages 113–118. Plenum Press, 1998.

[23] Wikipedia. Inverted pendulum — Wikipedia, The Free Encyclopedia. `http://en.wikipedia.org/w/index.php?title=Inverted_pendulum&oldid=291321108`, 2009. [Online; accessed 21-May-2009].

[24] Wikipedia. Pendulum (derivations) — Wikipedia, The Free Encyclopedia. `http://en.wikipedia.org/w/index.php?title=Pendulum_(derivations)&oldid=306566978`, 2009. [Online; accessed 7-August-2009].

[25] D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins. Non-holographic associative memory. *Nature*, 222(5197):960–962, 1969.