Refining the Neuro-Connector Model

Paul Wilson

MSc in Artificial Intelligence
Division of Informatics
University of Edinburgh
2000

Abstract

The neuro-connector model is an artificial neural network developed as a model of behavioural selection in animals, based on observations of Siamese fighting fish (*Betta splendens*). The possibility of using it as a robot controller, both for biological and engineering research, has also been suggested and investigated. This dissertation describes refining the model to incorporate artificial neurons with outputs that can take on any value in the range 0 to 1, rather than the binary outputs of the original neurons. First, the new neurons are described mathematically, then a new weight update algorithm capable of dealing with them is suggested. Finally the dynamics of the new network are investigated and a simulated classical conditioning experiment carried out.

Acknowledgements

Firstly, it has to be the Lord Jesus: "For by him all things were created.... He is before all things, and in him all things hold together." *Colossians 1:16,17 (NIV)*

I owe a great debt of gratitude to my supervisor, Bridget Hallam, for supplying advice, especially in keeping check on the biological plausibility of the work, providing guidance and inspiration during this project, yet also giving me the freedom to pursue my own ideas.

My thanks also go to Dr John Hallam, for useful advice on the mathematical implementations; Jan and Marietta for many fruitful discussions on the workings of the original hypothesis, and its problems; Nico, for proof reading chapters 3 and 4; and my parents for encouraging me to take this course. Finally, I thank the EPSRC who provided financial support during my year of study via advanced course studentship no: 99407173.

Contents

List of Figures

Chapter 1

Introduction

This short chapter summarises the background work in which the project is grounded (section 1.1), outlines the initial aims of the project and summarises how well those aims were achieved (section 1.2) and gives an overview of the contents of the remainder of the dissertation (section 1.3).

1.1 Motivation for project

This thesis is concerned with mathematically refining the neuro-connector model. The neuro-connector model was proposed as a model of animal behavioural selection in [Halperin 90], based on observation of Siamese Fighting Fish (*Betta splendens*), and the advantages of using it as a robot controller have been discussed [Hallam 00b, Halperin www, Hallam *et al.* 97, Hallam 00a] and attempted both in simulation [Hallam 99] and in physical robots [Weßnitzer 00, Scott 00] with varying degrees of success. Despite this, the mathematics underpinning the implementations is shaky and diverges from biological plausibility in several areas. The current software simulation [Hallam 99] is computationally expensive and faces both computational and theoretical problems running in real time.

1.2 Aims and achievements

The aims of the project were threefold:

1. To replace the existing binary output neurons[1] with neurons capable of a continuous output[2] .

2. To design an algorithm to implement Halperin's update rules compatible with the new analogue neurons.

3. To test the new system on a small set of animal conditioning paradigms, similar to the simulations in [Hallam 00b, chapter 5].

The reasons for the first aim are discussed in depth in section 4.1, but briefly it is to improve the biological plausibility of the model and thus hopefully provide a better control architecture. The second aim is a knock-on effect from the first—the current weight update algorithm is incompatible with continuous output neurons.

These two of these aims, the main bulk of the proposed project, have been carried out successfully and tested. The third aim, unfortunately, was only partially fulfilled as time did not permit a proper testing of the model—this could provide fuel for at least another MSc thesis in itself. A short test of the net was possible, showing it was capable of simulating one type of animal learning.

1.3 Dissertation overview

Before being able to adequately address the work carried out through the course of this project, it must be placed in the context of wider research

[1] Throughout this thesis, the term 'neuron' refers to an artificial neuron, rather than its biological inspiration. The same applies to 'synapse'. Also, the artificial neuron may actually more accurately model a neural *circuit* than an individual neuron.

[2] Hereafter referred to as *analogue neurons*, although strictly speaking they will most likely still be represented in a digital form in a computer.

in this field. Chapter 2 provides an introduction to both "engineering" and "biological" uses for robotics, and argues that the neuro-connector model can be considered as part of both fields, providing a useful engineering tool and a proposed method of animal learning.

Chapter 3 then reviews the current description of the model and weight update rules, concentrating on the topology of the network and the mathematical implementation of its component neurons and synapses. It concludes with a discussion of the pros and cons of the model, together with suggested applications.

A revised version of the artificial neurons is proposed in chapter 4. Initially there is a discussion of the advantages of changing to analogue neurons, both in terms of increasing the accuracy of biological modelling and in terms of possibly creating a more flexible robot controller. Following this, the design specifications for the new neuron is spelt out and a candidate neuron, the *phasic analog* [sic] *neuron* from [McMillen *et al.* 99], introduced and discussed in detail.

Virtually all of the work of the project was involved in implementing Halperin's synaptic update rules (set out in section 3.2.3); chapter 5 describes the implementation of synaptic weight updates, staying in the spirit of the current implementation where possible. The reasons for the failure of the current method of weight updates is discussed and a new method put forward in considerable detail, along with associated parameters required. Some testing on the new algorithm is also carried out.

The new components are then combined, within the current network architecture, to form a complete mathematical implementation. The intricacies of this are described in chapter 6, along with a brief description of the constraints on parameter values within the model, required to implement behaviour consistent both with Halperin's description and with biological plausibility. The second half of the chapter is devoted to testing the model in

simulation with an experiment based closely on that carried out by Halperin for classical conditioning, with a discussion of the results.

Finally, chapter 7 summarises and evaluates the work carried out, discusses the problems encountered and proposes areas for further research with the neuro-connector model.

Chapter 2

Background

This chapter aims to show the position of the neuro-connector model in both engineering and biological research. It starts with a whistle-stop tour of robotics from an engineering perspective; with its goals and methods (section 2.1), before moving on to the biological interest in robotics as a test bed for evaluating hypotheses (section 2.2). Next the link between 'science' and 'engineering' is examined, and the boundary separating them shown to be less than sharp, (section 2.3) before the neuro-connector model is shown to have the possibility of being beneficial to both biologists and engineers (section 2.4).

2.1 Engineering interest in robotics

The power of creating machines to carry out activities burdensome or impossible to people, and the possibility these machines may turn on their human masters or develop societies by themselves, has provided fuel for science fiction writers for generations, to the extent where it is virtually impossible to find a sci-fi book without a robot or android character. More recently, serious university research has focused on creating such devices, with greater or lesser degrees of success. One thing is clear now though—the robots of

science fiction are still a long way off.

2.1.1 Definition of *robot*

> One of the mechanical men and women in Čapek's play; hence, a machine (sometimes resembling a human being in appearance) designed to function in place of a living agent, esp. one which carries out a variety of tasks automatically or with a minimum of external impulse.
> —*Oxford English Dictionary, second edition,* 1989.

This description feeds on these science fiction ideas of robots and provides a rudimentary distinction between *robots*, which have a degree of autonomy and are capable of performing a variety of tasks, and other mechanical devices which may not be.

The traditional engineering approach to creating such devices is similar to that first described by Čapek in the opening scenes of his famous play [Čapek 20]:

> Well, anyone who's looked into anatomy will have seen at once that man is too complicated and that a good engineer could make him more simply.... A working machine must not want to play the fiddle, must not feel happy, must not do a whole lot of other things. A petrol motor must not have tassels or ornaments, Miss Glory. And to manufacture artificial workers is the same thing as to manufacture motors. The process must be the simplest, and the product must be the best from a practical point of view.
> —Karel Čapek, *Rossum's Universal Robots*, 1920.

That is to say, an engineer can solve the problem of creating a useful robot using a better solution than any found in nature, in a similar manner to the

way in which, after years of fruitless attempts to copy birds, flying machines were finally constructed which bore little resemblance to anything found in nature. However Čapek's remarks, and most robotics researchers, carry the implicit assumption that nature still provides the benchmark by which robots are evaluated.

2.1.2 Early attempts

With the advent of computer technology and early work in artificial intelligence (AI), the idea that "[An intelligent agent] would tend to build up within itself an abstract model of the environment in which it is placed. If it were given a problem it could first explore solutions within the internal abstract model of the environment and then attempt external experiments" (Marvin Minsky, quoted in [Arkin 98]) was dominant, now referred to as the *symbolic paradigm*. This engineering attempt to create robots provided some interesting artifacts, but ultimately proved computationally explosive[1] for all but trivial worlds. Certainly not yet coming close to being able to "function in place of a living agent."

An alternative approach was required if the dream of autonomous robots was to be realised.

2.1.3 Behaviour based paradigms

A quick glance at biology revealed that even "simple" animals were routinely able to solve the problem that ground computers to a halt under the symbolic paradigm. Minsky's vision of symbolic representation being the key to intelligent was beginning to be questioned, and robotics researchers admitted they needed a new way. Taking a bit of inspiration from biology, Rodney Brooks

[1] That is, computational power required increased exponentially for linear increases in problem complexity, so even faster computers would provide only limited improvements.

proposed starting with primitive robots, at the level of insects, and building up complexity incrementally [Brooks 86b]. Many architectures exist in this paradigm, such as Brooks' own subsumption architecture [Brooks 86a] or the Really Useful Robot (*RUR*) architecture ([Nehmzow *et al.* 89] and section 7.3.4).

Although these architectures were plagiarizing some ideas and methods from biology, they still hoped to create 'simpler' implementations at the next level up. The engineering methodology of robotics was still very much in line with that of the Turing test—provided the robot acts in the desired manner, what is going on inside it is of no real importance—it is effectively a black box.

2.2 Biological interest in robotics

Biologists create hypotheses concerning the nature and action of living things. These hypothesises can have their target anywhere from the biochemical level, concerning the actions of proteins and enzymes for example, right through to the ethological level, concerning the interactions of animals with each other and their environment.

In order for hypotheses to be scientific, they must be falsifiable. That is, they must be able to generate "outputs" (in a broad sense of the term) which can be compared to nature and enable the hypothesis to be shown inadequate, or else consistent with that particular case. Hypotheses consistent with large numbers of cases are deemed useful or accurate.

Most biological hypotheses never progress beyond the stage of qualitative comparisons with observations in natural media (e.g. animals, tissues, cells, etc.). Some, however, are tested in computer simulation, but this is often impossible to evaluate in an impartial manner, as the designer of the simulation makes humanly biased decisions on what is modelled in the simulator. This

frequently misses key features, but can stand as a good preliminary model test ground [Webb 00].

Testing hypotheses on robots, however, has the advantage that the world is used as its own model [Brooks 86b], and is thus much less susceptible to the biases and presuppositions present in simulators. It is also able to generate conclusions possibly unforeseen from the hypothesis and present nouvelle perspectives on the work (although both these may occur to a lesser extent in simulation).

In contrast with the 'engineering' methodology discussed in section 2.1, biology is interested in the inner workings of the robot—it must be consistent with the hypothesis being tested. As Webb highlights, however, these hypotheses are rarely stated accurately enough to permit exact implementation as a robot. Almost invariably, *biologically plausible* engineering solutions (i.e. "make it work") are padded around the key elements of the hypothesis. Nevertheless, the model must resemble biology at some level, as opposed to the 'black box' methodology described in section 2.1.3. An overview of current thinking and research in this area is presented in [Webb 00].

2.3 An unhelpful distinction?

> There are two types of people in this world: those who separate everything into two groups, and those who don't.

Scientists in particular are very keen to classify anything they can lay their hands (or minds) on, often splitting hairs over fine distinctions to include, or more normally omit, certain types of artifacts from whatever set they are seeking to define. Often this is useful to unpack subtle differences in methodologies, to clarify the meaning of terminology or to give research a firm philosophical grounding. In the 1960's, however, fuzzy logic was developed

as an acknowledgement by mathematicians that often classifications do not have rigid boundaries; a similar view will be beneficial to this area of robotics.

The engineering methodology described above is characterised primarily in terms of function—the system works if it does the right thing, regardless of how it does it. It is judged on factors such as production costs, production time, running cost, maintenance costs, mean down times, computational power required, etc..

Nature already has a working solution for many of the problems in engineering, particularly in the field of mobile robots. These solutions may be the best solution to a particular problem from an engineering perspective: rather than try to create a new solution from scratch, why not take one that already works? In turn, this feeds back into the biology by finding out what does and does not work in real life, and throwing up questions for biologists to address (with the aid of the engineers to test and use them ...).

As a final discussion on this point, [Pfeifer 96] poses two solutions to a hypothetical problem of creating a robot to collect ping-pong balls in a particular room as quickly as possible. His engineering solution is simply a high powered vacuum cleaner where, he argues, "the only considerations are performance and price. The performance criterion in this case is obvious, namely the number of balls collected per unit time." The cognitive scientist, he claims, would rather build a robot capable of learning to distinguishing the balls from other objects in the room and able to pick them up and carry them safely with a hand-like actuator to solve the problem. The task specification does not mention that no other objects should be picked up. It makes no requirements on limits of noise pollution and assumes a relatively flat environment. If power consumption, noise generated and a requirement not to disrupt other objects in the room are included in the performance criterion, the vacuum cleaner may no longer be the best solution, and the cognitive scientist's robot may be better. Since real environments cannot be

known fully in advance and are subject to change, robust systems, capable of adaption (or learning), may provide the best systems to implement to create robots capable of functioning in place of a living being. Thus, a realization that the task and environment *cannot* be specified to a high degree of accuracy for many applications in the real world leads to the possibility that biological solutions, of the type discussed above, may prove best at fulfilling engineering requirements. To end on a cautious note, [Webb 00] warns of the dangers of using "biological inspiration" as a buzz-word to avoid both the engineering necessity of comparing a chosen implementation against more traditional solutions and the need to contribute to biology by accurately modelling it.

2.4 Neuro-connector model

The neuro-connector model, the basis of this thesis, falls into both the biological and engineering camps. It was proposed by an ethologist and is claimed to be "a working hypothesis for the functional mechanism underlying much of vertebrae learning" (Janet Halperin, quoted in [Hallam 00a]). The initial version of the model is described in detail in chapter 3.

Various classical conditioning experiments have been carried out in simulation on the model [Hallam 00b]. Realising the shortcomings of simulation (section 2.2), work has recently been carried out on implementing the network on physical robots to further test it. This involves hard coupling of the net to actual sensors and motors. The decoupling of the net (control structure) from the robot's physical body can be justified in terms of the results from conditioning experiments (see [Schmajuk 97] for examples)—a generic set of results exists describing responses for many types of animal in conditioning experiments, despite there being no such thing as a generic animal! Thus it is hoped the neuro-connector model will provide a generic

control structure to be coupled to specific robot bodies. This sensor–net and net–actuator coupling, unspecified in the model (see section 3.2.1), has been investigated in a Lego robot [Scott 00] and a Khepera robot [Weßnitzer 00].

This work, however, is concerned with the mathematical implementation of the model, and how that can be improved with the hope of implementing a more biologically realistic model which will in turn prove to be a better robot controller than the current model. It is worth noting now, however, that a true evaluation would be best carried out on a robot, rather than in simulation, for the reasons discussed above. Unfortunately, time did not permit such an implementation in a this project—evaluation is only made with respect to how well the new mathematics and implementation follow the textual hypothesis.

2.5 Summary

This chapter has quickly reviewed the reasons for wanting to implement biological adaptive control mechanisms in robots, both from the "engineering" approach of creating useful artifacts capable of being able to carry out a variety of tasks reliably and in potentially unknown environments, and the biological point of view of trying to generate and test hypotheses of animal behaviour and learning in real world environments, showing how the neuro-connector model can be useful in both these disciplines.

Unfortunately there is not room for a more in depth discussion on this fascinating inter-disciplinary area. For a good introduction to the history of robotics, current thinking and achievements within the behaviour based paradigm, consult [Arkin 98]. More details on the philosophical issues and relationship between the engineering and biological methods can be found in [Webb 00].

Chapter 3

Original model

In this chapter the original version of the neuron-connector model will be described, as devised in [Halperin 90] and further developed in [Hallam 00b, Hallam *et al.* 97]. Section 3.1 is an overview of the model, describing its key features. Section 3.2 provides details on the components of the model, including the neurons, synapses and weight update rules. Finally, section 3.3 gives a discussion of the uses of the model, its strengths and weaknesses.

3.1 Model overview

The neuro-connector model is a self reinforcing artificial neural network (SR-ANN), developed by Janet Halperin in order to model aggressive behaviour in Siamese fighting fish [Halperin 90], then used to test classical conditioning experiments [Hallam 00b]. Being an SR-ANN, it 'learns' by assessing the effects of its outputs itself, without the need for a 'teacher' to show it the correct response [Gaudiano *et al.* 97]. It differs from many other SR-ANN's in that it determines the success or failure of an action by comparing the observed time difference between the sensory state finishing and the behaviour finishing (t_{obs}) with a predefined expected difference (t_{exp}). If the two times show good correlation, the behaviour has been successful. If not,

it is deemed inappropriate for the given sensory situation. This technique can be fooled by coincidences which leave t_{obs} and t_{exp} with similar values; this is not a problem, however, since these coincidences are assumed to be rare and, biologically, animals can be fooled by them too.

3.2 Original implementation

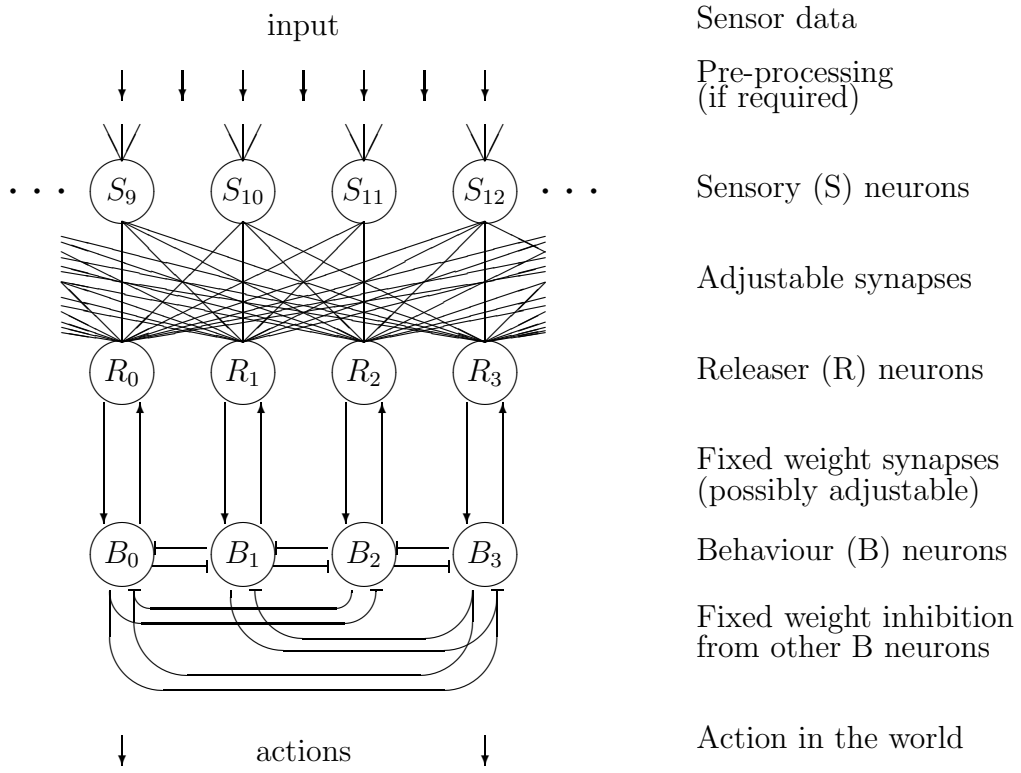


Figure 3.1: Structure of network in neuro-connector model (courtesy B Hallam)

The neuro-connector model consists of three types of units (artificial neurons), connected together by synapses arranged as in figure 3.1. This section describes the structure and operation of each component, together with an overview of the network.

3.2.1 Network architecture

The network looks, at first glance, like many neural network models, for example a multi-layer perceptron (MLP) network (for an introduction to MLPs, see [Luger & Stubblefield 98, chapter 14]). In fact it does share many of the characteristics of an MLP network, in that an input is entered into the network as a series of real valued numbers to input neurons (S neurons in the diagram); these neurons then output a value which forms the input to other neurons, via a weight factor.

Despite these similarities, there are several major differences between this model and a MLP network:

1. *Neurons have memory.* The output from each neuron in a MLP network is purely a function of their current input. In Halperin's net, however, the neurons have a memory of recent activity and their current output depends upon the state of their previous output. This is discussed further in section 3.2.2.

2. *Synapses have memory.* In addition to the neuron's memory, synapses also have a memory of the recent activity of their pre- and post-synaptic neurons, which is used in weight update rules, (section 3.2.3).

3. *Synapses cannot learn inhibitory weights.* Whilst MLP synapses can take any real valued weight, neuro-connector synapses can only have adjustable weights between zero and one. Weights can be hard coded outside this range, either to be inhibitory (e.g. B–B weights), or to provide default reflexes, but the weight update rule will only function correctly on weights in this range.

Data from the sensors is pre-processed, if required, to provide a sensory state as each of the sensory (S) neurons represents a *state*, such as 'standing

in front of a wall', rather than a raw sensor input value. The mechanism for this conversion is beyond the scope of the model (see section 3.3).

These processed states then propagate to the releaser (R) neurons, each of which is uniquely associated with a behaviour (B) neuron, another departure from the traditional MLP architecture. Each B neuron is associated with a behaviour in much the same manner as the S neurons are associated with sensory state—i.e. at a high level, unconcerned with details of implementation. In normal operation, multiple S and R neurons will be on, but only one B neuron will be active. Feedback between the active B neuron and its associated R neuron is provided, and the active B neuron inhibits other B neurons encouraging a behaviour to continue once started[1] , although this inhibition can also give certain behaviours priority to interrupt the active B.

3.2.2 Neurons

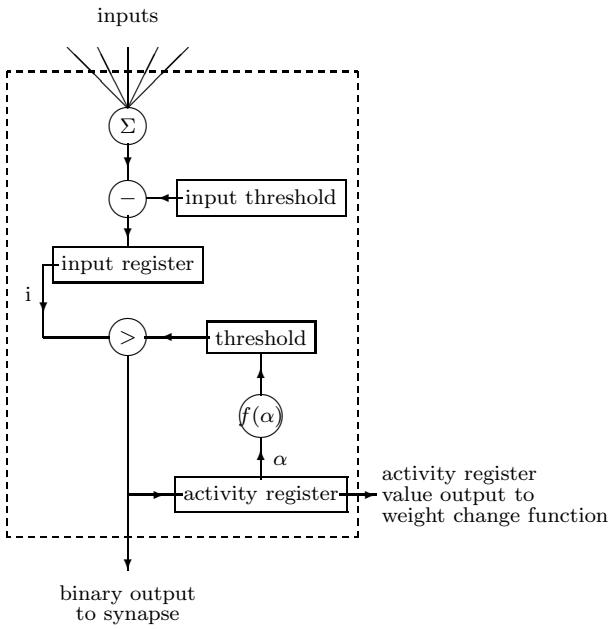


Figure 3.2: Schematic of original binary output neuron

[1]But see also section 6.1.1.

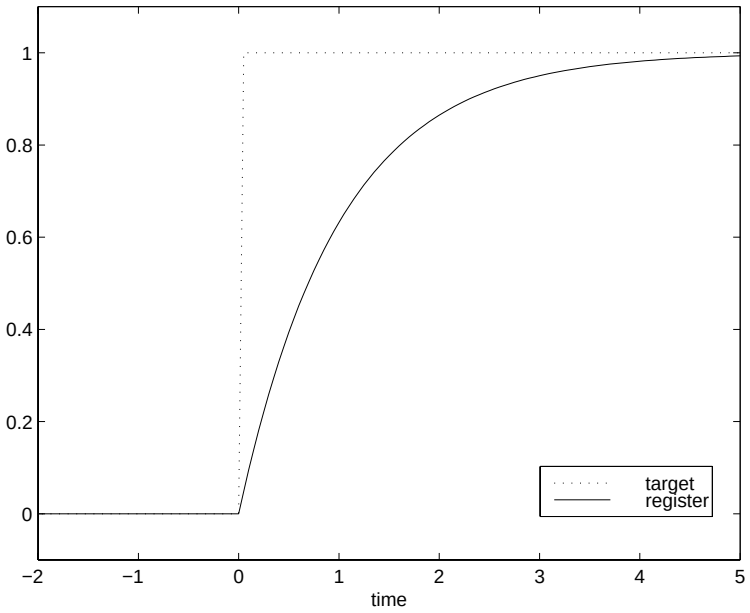


Figure 3.3: Register dynamics with a step input at $t = 0$

The network models *phasic*[2] neurons, shown diagrammatically in figure 3.2. The inputs to each neuron are summed and a threshold subtracted to avoid being triggered by a low level of input noise. This value then passes into the *input register*, a first order lag system governed by the equation:

$$\tau_x \dot{x} = x - \text{target} \tag{3.1}$$

where x is the register value, $\dot{x} \equiv \frac{dx}{dt}$, and τ_x is the time constant for the register. With a constant input, the input register output is an exponential decay toward this input value as shown in figure 3.3. That is, x maintains a *trace* of the input.

The input register is compared with the time varying threshold to determine the binary output: one if it is above the threshold, zero otherwise. The binary output value is then set as a target for another register, the activity

[2] *Phasic* (or adaptive) neurons initially respond to a stimulus, but cease to respond as the stimulus persists, as opposed to *tonic* neurons which give a constant response to a constant stimulus.

register (α), which operates according to equation 3.1, with a different time constant, τ_α, i.e.

$$\tau_\alpha \dot{\alpha} = \alpha - \text{target} \tag{3.1a}$$

Since the input register describes the *current* input to the neuron (i.e. averaged over a short time period), whilst the activity register monitors its *past activity* (i.e. over a much longer time period, a kind of short term memory or *STM*), τ_α is typically much larger than τ_x. As if enough free parameters did not already exist in the model it permits the biologically plausible effect of having different values of τ_x and τ_α depending upon whether the registers are increasing or decreasing (denoted by $\tau_{reg}^{(\uparrow)}$ and $\tau_{reg}^{(\downarrow)}$ respectively). This is a similar effect to being suddenly startled by hearing a noise in the night—in a fraction of a second, it is possible to move from being fast asleep to completely awake, yet it takes much longer to get back to sleep again.

Threshold function

The question of the relationship between the activity register and the threshold ($f(\alpha)$ in figure 3.2) is a thorny one which Halperin does not adequately address. Some sort of continuous function may be suggested, but this causes problems with oscillations. Consider the case where the input is switched to a constant value having been zero for some time (i.e. a step function). The input register will fairly quickly assume this value compared to the activity register. The activity register will be effectively zero, and the neuron will switch on. This will cause the activity register to rise towards one and, via $f(\alpha)$, the threshold to rise too. There will come a point, however, when the threshold will exceed the input register value. This then causes the neuron to switch off and α decreases. But this will cause the neuron to switch on again as α drops below the input register, x, and oscillations will ensue, at a frequency determined by the size of the integration step of the activity

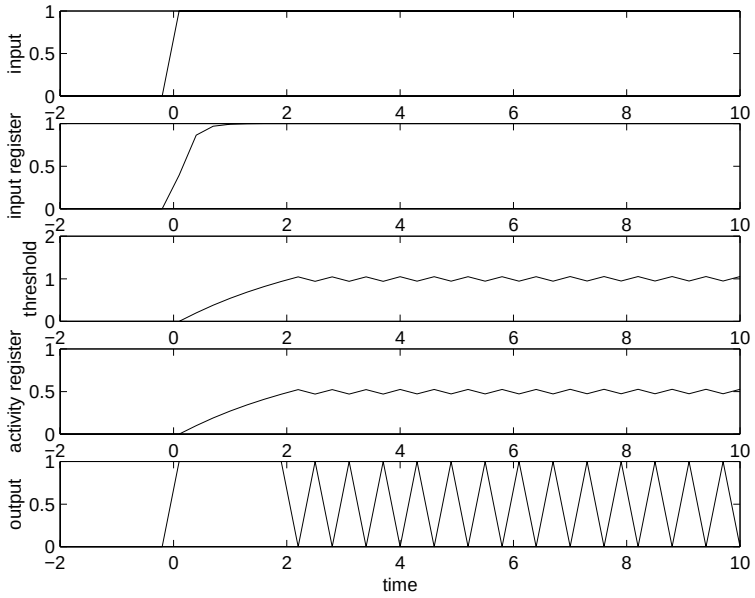


Figure 3.4: Oscillations with binary neuron

register. This phenomenon is illustrated in figure 3.4, where a large step size is used for the integration to show the oscillations. Clearly this is an undesirable feature—it brings instability depending upon the computational implementation rather than any properties of the model. Oscillations in output are not undesirable *per se*, in fact many activities appear to rely on them (see discussion at the end of section 6.1.3), however the oscillation should be a feature purely of the mathematics rather than the iterative methods used to solve differential equations.

The solution suggested in [Hallam *et al.* 97] is to incorporate hysteresis into the function so that, for any given value of α, the threshold, $f(\alpha)$, is smaller if α is increasing than if α is decreasing. The problem can also be solved by enforcing an adapt out time—once the neuron switches off, it is unable to fire again for some time regardless of its input—as implemented in [Hallam 99]. Alternatively, a momentum term (second order differential term) could be incorporated to the activity register to control the oscillations (rather than have their frequency determined by the integration step size),

but although this is mathematically elegant, there is no biological justification for having the activity register continue to rise when the neuron is off. Making the threshold a function of the input rather than the output could also solve this problem if the right function is selected, although it is less biologically plausible as the threshold register approximately represents the concentration gradient of ions across the cell membrane. This decreases with each voltage spike, therefore is approximately a function of the spiking rate. Switching to an analogue output removes the problem (see section 4.1).

3.2.3 Synapses

Two types of synapse exist in this model: plastic (or adjustable) and fixed. Fixed synapses simply take the output of one neuron, multiply it by a pre-defined factor (or weight), and use this as the input for another neuron. The more interesting synapses, however, are the plastic synapses, and the remainder of this section is dedicated to describing them.

In addition to the properties listed above, each plastic synapse also has a predetermined notion of the expected time delay between the pre-synaptic neuron switching off and the post-synaptic neuron switching off, called the *expectation time* (t_{exp}, see section 3.1), and a *command register*, a record of the correlation between the pre- and post-synaptic neuron activity with respect to t_{exp}.

The weight update algorithm used, crucial to the success of any ANN, consists of an attempt to mathematically implement several rules devised by Halperin. These are listed below, where, for the sake of conciseness, the synapses between an S and R neuron is considered.

1. *If good correlation exists between the end points of the activities of S and R neurons (i.e. $t_{exp} \approx t_{obs}$), the synapse should be strengthened.* This means that either the correct response was selected for a given input

sensory state (most likely), or else the correlation occurred by accident (see section 3.1). This should occur regardless of which neuron started firing first.

2. *If the S neuron has not been active, no significant weight change should occur.* This allows the same response (e.g. flee) to be triggered by multiple independent sensory states.

3. *If the S neuron is active without the R, the synapse should be weakened.* This means the inactive behaviour is unrelated to the current sensory state.

4. *If the correlation between S ending and R ending is poor (i.e. $t_{exp} \gg t_{obs}$), the synapse should be weakened.* This corresponds to the R being unrelated to the S.

Note that these rules describe a continuum in terms of the relationship between the correlations of the S and R neurons switching off: consider varying the time between the S and R switching off, starting with the S switching off a very long time before the R. This corresponds to rule 2 and no synaptic change. As this time difference decreases, rule 4 invokes a weight decrease before rule 1 implements an increase [Hallam 00b, chapter 4]. The current implementation then diverges from biology, under certain circumstances, by producing a weakening of synaptic weights as the time difference gets close to zero, whereas biologically a strengthening is observed [Hallam 00b, section 4.3]. If the time gap becomes negative (i.e. the S goes off after the R), rule 3 becomes active and weakening occurs.

The current method described for attempting to convert these discrete rules apply to a continuum of synapse weight change values dependent on t_{obs} is by using a command register, c, in each synapse coupled with the co-activation function, $A(\vec{\alpha})$, shown in figure 3.5.

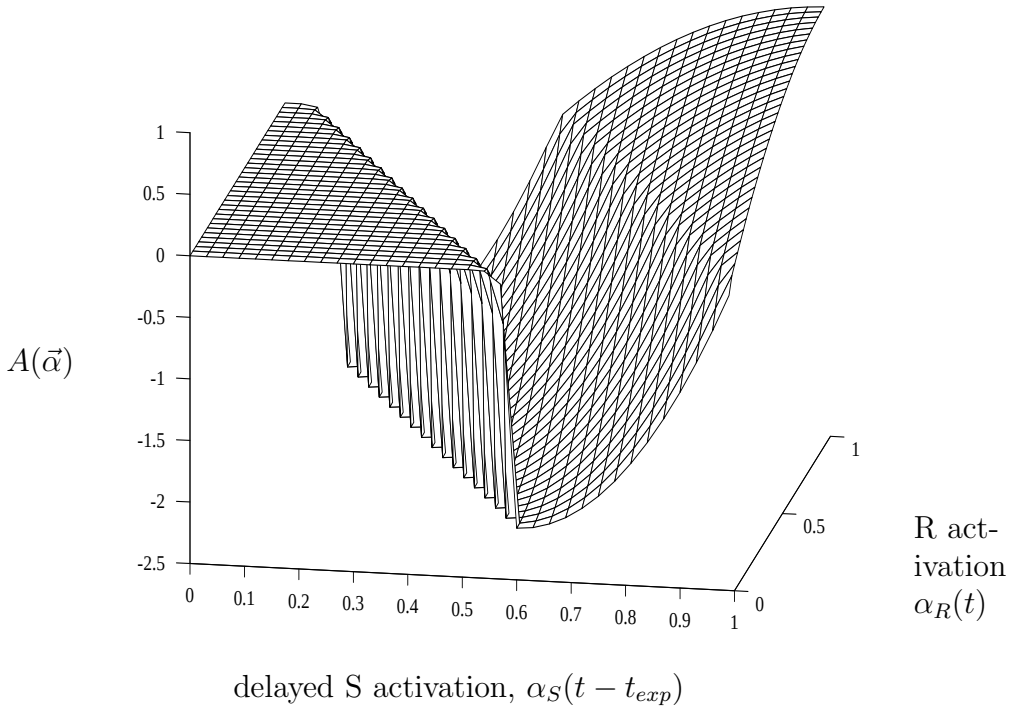


Figure 3.5: The control surface used for synapse weight adjustment $A(\vec{\alpha})$ (courtesy B Hallam)

$$A(\vec{\alpha}) \;=\; g\left(\frac{1}{2}(\vec{\alpha}\cdot\vec{u}-1)\right) \tag{3.2}$$

where $\vec{u}$ is $(2,1)$ and $g(\cdot)$ is the function

$$g(u) \;=\; \begin{cases} -2.08e^{100(u-0.1)} & \text{if } \; u < 0.1 \\ 13(u+0.3)(u-0.5) & \text{if } \; 0.1 <= u <= 0.5 \\ -4(u-0.5)(u-1.5) & \text{if } u > 0.5 \end{cases} \tag{3.3}$$

The purpose of the co-activation function is best understood by considering the trajectory of a vector, $\vec{\alpha}(t) = (\alpha_R(t), \alpha_S(t - t_{exp}))$, moving over the co-activation surface as S and R neurons fire. The α_S value is delayed by t_{exp} so that, for a perfect correlation, it returns to zero at the same time as α_R.

If we consider how $\vec{\alpha}$ moves over this surface, it will start at the origin when both neurons are off, and move towards one of the corners on the unit square as either the S, R or both neurons switch on.

The command register (c), is a register (see section 3.2.2) which tends towards the value given by the co-activation function, to implement the rules given above. So for good correlation, c tends to 1, for poor correlation, it tends to a negative value. Of course, upon returning to the state where both neurons are inactive, $\vec{\alpha}$ will pass through a negative region in the graph. Time constants must be set such that the decrease this causes in the command register is not enough to make it negative (corresponding to a weakening of the synapse).

Finally, the weight update rule incorporates Sinclair's rest principle by only updating the weights when both neurons have been inactive for a time. This is achieved mathematically with the following formula:

$$\tau_{wt}\frac{dw}{dt} \;=\; \chi(\vec{\alpha})|c|(w_{tgt} - w) \tag{3.4}$$

where w_{tgt} is 0 for $|c| < 0$ or 1 for $|c| > 1$, and $\chi(\vec{\alpha})$ is a gating function which is effectively zero when either α_R or the delayed α_S is greater than 0.1, as shown in figure 3.6. It is given by the formula:

$$\chi(\alpha'_S, \alpha_R) = (1 - \alpha'^{30}_S)(1 - \alpha^{30}_R) \tag{3.5}$$

where $\alpha_S(t)' \;=\; \alpha_S(t - t_{exp})$.

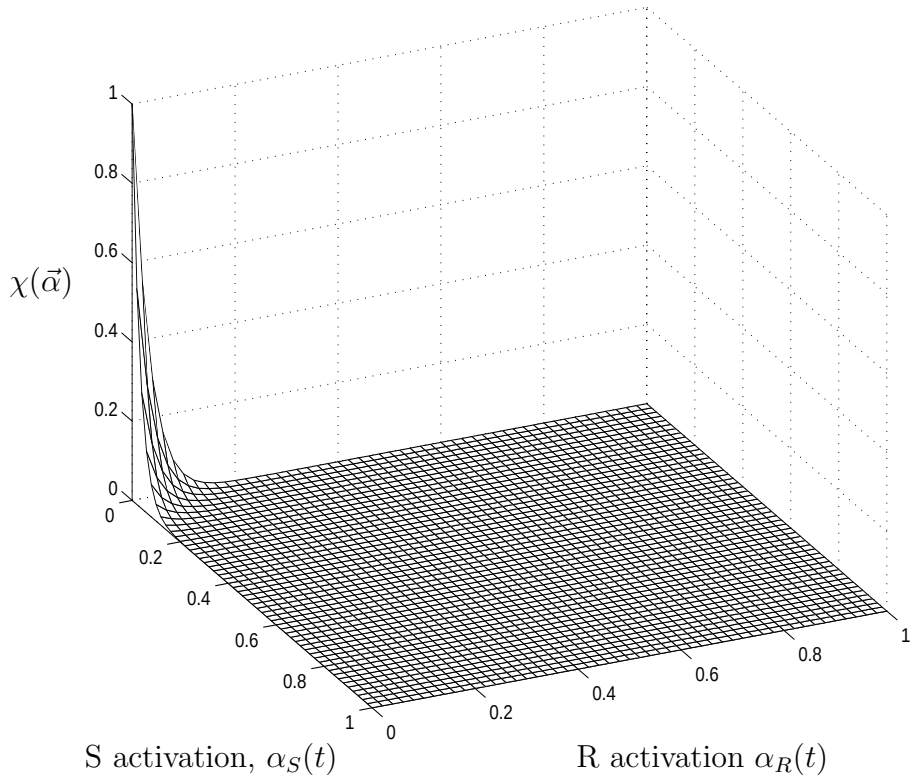


Figure 3.6: Gating function used for synapse weight adjustment $\chi(\vec{\alpha})$

3.3 Model uses

The model has certain strengths and weakness which will best determine its robotics applications.

Strengths

- *Ability to handle complex tasks*: The model is able to deal with making fine distinctions between similar sensory states in a sensor rich agent.

- *Flexibility*: The net can adapt to varying environmental conditions and still produce the "correct" response (for example, detecting predators in good or poor lighting).

- *Memory*: Once the net has learnt the correct behaviour for a sensory state, it will not forget it (although it can 'unlearn' it if the behaviour fails to produce the desired effect in the future).

- *Rapid learning*: The exponential terms in the registers used in the weight update procedure result in an algorithm that can rapidly learn the correct behaviour for a given sensory state, typically after only four or five presentations [Hallam *et al.* 97, sections 4, 5]. So the agent (either biological are robotic) can respond appropriately after only a little experience (which may be hand coded top start with).

Weaknesses

- *Computationally complex*: The network requires a fast computer and mathematical simplifications to run in real time. This will be less of a problem later with the increase in processing power, and would be a good candidate for implementation in a parallel architecture.

- *Mathematically inelegant*: Halperin proposed the model primarily in a textual form, describing its behaviour rather than its mathematical implementation. Her suggested mathematization ([Halperin 90, appendix I]) lacks the elegance of many other ANN architectures.

- *Fixed sensor and behaviour neurons*: The description of the model is only concerned with its inner workings—sensory neurons must be hard coded initially. This adds an extra burden to the programmer: how to choose sensory states which will be useful. Each S neuron represents a sensory state (rather than the input from a single sensor), so processing the input from the sensors to determine each state is also most likely a non-trivial task. Also, the problem of where to draw the boundaries between sensory states exists, as at one extreme each S neuron could refer to an individual sensor's output, whilst at the other end an infinite number of S neurons could be used, corresponding to all possible states of the world! This is also a potential problem with current psychological understanding, as it is unknown exactly how animals make these distinctions.

  A similar problem occurs with the behaviour neurons—each determines a particular behaviour, but it is up to the designer to pick these behaviours in advance, and decide what post-processing needs to be carried out to turn an active B neuron into possibly a set of motor commands. Although these behaviours can be viewed as fixed action patterns (FAPs, see [Plymouth www, Cardoso www]), it seems highly constrained to have an agent with a completely fixed tool box of behaviours, unable to update or add to them.

  It may be possible to have layers of automatic preprocessing structures (e.g. self organising maps) to pre-process some of the S neuron inputs and post-process the B outputs. This problem remains unresolved in

the new implementation and is discussed further in sections 6.1 and 7.2.

- *Hand coding*: As is probably already evident, the network requires a large amount of hand coding of values. The expectation times for each synapse must be hard coded (although if there is uncertainty, multiple synapses, with different t_{exp}'s, can be used at the expense of computational power, as must the default synapse weights, the B–B inhibitory weights and the pre- and post-processing. Again, this problem still exists in the new implementation. Its implications are investigated further in sections 6.2.2 and 7.3.3.

Chapter 4

Analogue neurons

The modifications to the binary neurons (described in chapter 3) are discussed in this chapter. Firstly, the reasons for wanting to change from neurons with a binary output to ones with an analogue output are reviewed (section 4.1). The new properties required are then discussed (section 4.2) and design choices explored (section 4.3). The architecture of the new neuron is developed (section 4.4) and some mathematical tools used in the simulation mentioned (section 4.5).

4.1 Why use analogue neurons?

Scientific models are a mathematical representation of physical reality. By the very nature of being a model, they lose some of the detail of the reality in order to attempt to discern the key features of a system. The model can then be judged on how well its predicted output measures up to the real output from the system, and the complexity of model required to approximate the system. Model refinement therefore, can be aimed at either reducing the complexity of the model to discern its key operations and components, or, as in this case, making the model components more closely resemble the elements of the physical system they represent to attempt to create more ac-

curate predictions. In the case of sensory–motor interface in animals, which the neuro-connector model is attempting to represent, the reality is an extremely complex system, acting through both chemical and electrical media, with organisation over several orders of magnitude, from the ordering of atoms and ions within fatty acids, sugars, amino acids and nucleotides, up through synapses, micro-circuits, neurons, local circuits to systems and pathways within the central nervous system [Shepherd 94], which is intrinsically linked to the rest of the animals body, ranging from tenths of nanometres to tens of centimetres. Representing all this information in a digital computer is inconceivable anytime in the foreseeable future.

The output from biological neurons is a series of voltage spikes[1] (see [Schmajuk 97, chapter 1] for a brief introduction) which are transported to other neurons through dendrites. The original neuro-connector model [Halperin 90, Hallam 00b] operates at a level above this, giving a binary output if the neuron spiking *rate* is above a predefined threshold (concealed in the threshold of figure 3.2 and other operating parameters). This loses a vast amount of information but, it is hoped, still carries through enough to make the model behaviour reflect observed behaviour. Instead of sending an output if the firing rate is above a threshold and not if it is below, more information can be retained if the output is given a value reflecting the frequency of the spiking. This still loses information, as it is impossible to define a frequency at a point in time, but rather it must be defined over a period of time[2] . Recent advances in neuroscience have shown that animals can distinguish and exploit even small differences in the time between individual spikes, and the premise in virtually all ANNs that all inputs to a neuron are passively

[1] Actually, neurons have other methods of inter-cellular communication [Shepherd 94, page 99] which are being neglected (or combined with the electrical, depending on your point of view) in this model.

[2] This is much the same as the way it is impossible to measure the speed of an object from a single photograph (snap-shot of time). Rather, its position at two points in time is needed, and the speed can be calculated *between* the two points.

added is a poor reflection of the biological complexity of actual neurons (see [Koch 97] for a discussion on some of these differences), however analogue neurons are a step closer to the natural reality.

4.2 Desired properties

This thesis consists of an effort to (mathematically) improve the neuro-connector model. It is therefore important to get a feel for what the key elements of the model are, and what is purely implementation detail, although these boundaries are somewhat plastic. From the literature on the model [Halperin 90, Hallam 00b, Hallam *et al.* 97], the main properties are:

1. The synaptic weight update ideology (that weight changes depend upon the correlation between the ending of the firing of the pre- and post-synaptic neurons).

2. The architecture of three (or four[3]) levels representing sensory, releaser and behavioral processing.

3. Neural units with an idea of time and recent activity (i.e. a short term memory).

4. Synaptic weights only change after both the pre- and post-synaptic neuron have been inactive for a period (Sinclair's rest principle).

In addition to the third property, the artificial neurons should:

1. Be phasic, or able to 'adapt out' after a period of activity, as described in [Hallam *et al.* 97, section 2.2][4] .

[3] [Halperin 90, p56] defines four, but the I and E levels can be condensed into one B level.

[4] Tonic, or non-adapting, neurons in addition would be biologically plausible [McMillen *et al.* 99], and, as it turns out, are simply a special case of phasic neurons with infinite τ_α's.

2. Accept an unbounded input (i.e. in the range $\{-\infty \ldots \infty\}$).

3. Produce an output in the range $\{0 \ldots 1\}$, representing a fraction of the neuron's maximum firing rate.

4.3 Design choices

The artificial neuron structure will be selected according to a three way trade off between competing forces. These are:

- It should bear a strong resemblance to Halperin's original model.
- It should be biologically plausible.
- It should be computationally tractable.

Where possible, mathematically elegant solutions will be chosen in preference to 'hacks'.

The reason for the first factor has already been touched upon (sections 4.1 and 4.2); in summary it is because the work is concerned with *refining* rather than *replacing* Halperin's model. The second factor means that, although modelling every aspect of the system is computationally impossible, the behaviour of the neuron must, to some extent, model that of an actual biological neuron. The third factor is purely practical—although this work is not concerned with producing an optimised system, or even one that can run in real time, the work must not hit the combinatorial explosion problem. Additionally, it is desirable to select faster algorithms where a choice exists and it may be necessary to sacrifice some model details which incur a large computational penalty.

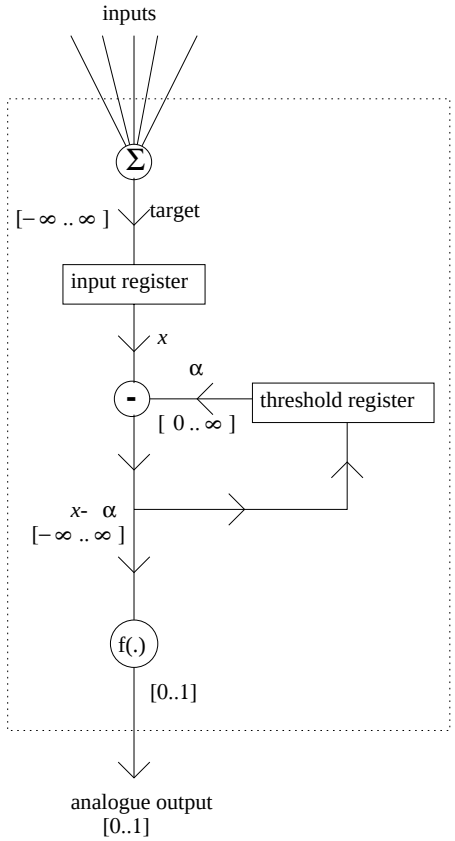


Figure 4.1: Schematic of analogue neuron

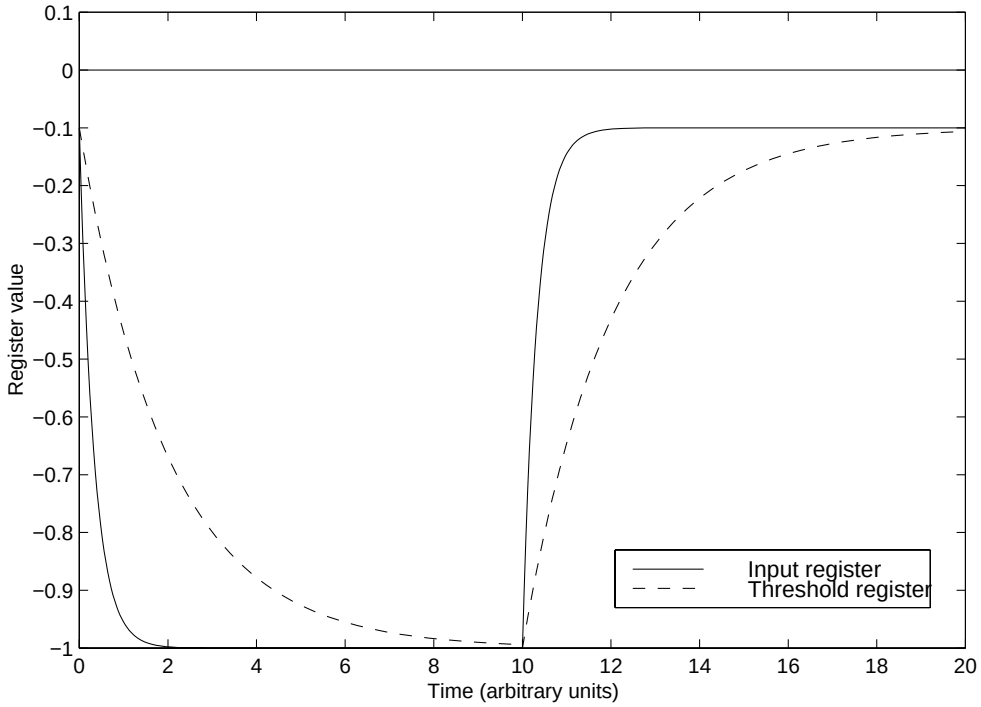


Figure 4.2: Problems with unbounded activity register

4.4 Neuron architecture

The neuron architecture used is that described in [McMillen *et al.* 99], and shown in figure 4.1. The range of variables is indicated where appropriate. This neuron bears a strong resemblance to the original neuron (figure 3.2) as desired, with the notable changes being discussed below.

4.4.1 Subtracter unit

The comparator in the original has been replaced by a subtracter unit. This preserves much more information than a straight forward comparison by giving an output in the range $\{0 \ldots 1\}$, rather than a binary value.

4.4.2 Threshold register

The time varying threshold is now determined by the threshold register, α, which tracks the input register, x, rather than being set as an (unspecified) function of the activity register, which tracked the output (section 3.2.2). Note that the threshold register is constrained to remain above zero to prevent the undesirable situation of a neuron giving a strong output when its input changes from being strongly inhibitory to slightly inhibitory, as illustrated in figure 4.2, where at time 10, the strength of the inhibitory input is reduced and $x - \alpha$, a measure of the neurons output, is strongly positive, despite the neuron's input register, x, being negative (inhibitory). Restricting the range of the input register would also solve this problem, but is not as biologically plausible.

The input register–threshold register coupling produces an effect equivalent to an electronic band pass filter: the input register smoothes very high frequencies (to their mean value), and the threshold register removes low frequencies, only permitting those in a band in between to be propagated. The mathematics for the frequency response are developed in [McMillen *et al.* 99].

4.4.3 Input threshold

The input threshold in the original is no longer required in the analogue version, again as a beneficial side effect of having the subtracter unit instead of the comparator. This threshold was originally required to filter out low amplitude noise from the input signals. With an analogue neuron, however, low amplitude noise should only cause small changes in the neuron output (depending upon the function $f(.)$—see section 4.4.4). Also, now that the α threshold tends towards the input register, subtracting a constant value from the input register (as in the original neuron) would simply lower the operational point of the threshold register, subject to two conditions:

1. The threshold register does not try to fall below its zero cut off point.

2. The transient effects of the neuron are neglected.

The first of these points may seem too strict, but it can be eliminated by changing the minimum permitted value of the threshold register from zero to the desired input threshold. For example, to set an input threshold of y, prevent the threshold register from moving below y—i.e. bound it in the range $\{y \ldots \infty\}$.

The second means allowing the system to 'settle' before using it, i.e. allow the activity registers to reach constant values in the absence of any network stimulus. It must be noted that this is only possible in phasic neurons—for tonic neurons the threshold registers do not change, and cannot act to eliminate the input threshold value. In this limited case, however, the threshold register can simply be initialised to the value of the desired input threshold.

The modifications above may seem excessive for the puny computational benefit of removing a single subtraction, however they are included simply to show that the effects of adding an input register can be achieved by other means with this neuron and no functionality is lost in removing it. In reality, its original purpose was to stop a binary output due to small amplitude noise in the original, a problem which no longer exists—in analogue neurons, a small input variation results in a small output variation[5] . None of the modifications in the preceding paragraph were required in the course of this work.

4.4.4 Squashing function

The additional function, $f(.)$, is required to convert the difference between the input register and threshold register, $(x - \alpha)$, which is on an unbounded

[5]This is only valid for squashing function with continuous gradients that are never too steep.

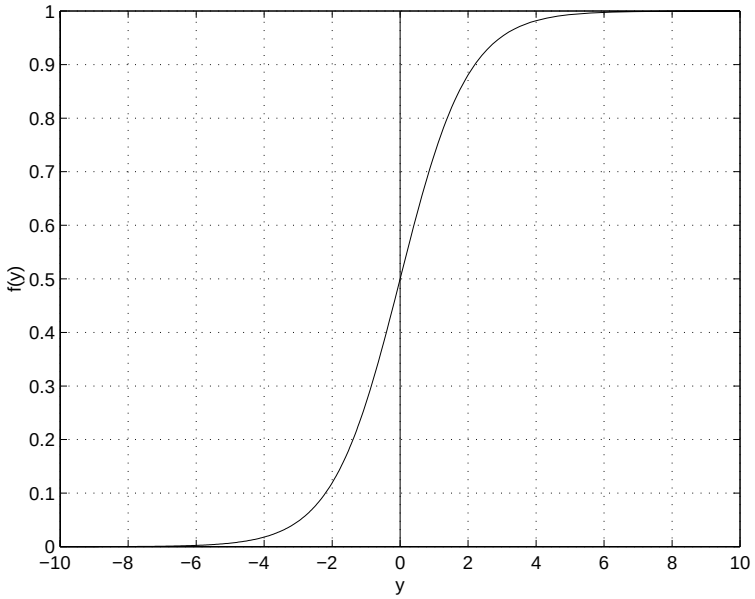


Figure 4.3: Sigmoid squashing function

range, to a firing rate bounded between zero (no activity) and unity (maximum spiking rate). Biologically, there is not much which can be said about this function, except that it should not have a negative gradient anywhere (i.e. increasing the input should never decrease the output), therefore computational intuition and mathematical elegance guide the decision. Three such functions are considered for this role, each of which can be scaled by letting

$$y = \gamma(x - \alpha) + \theta \tag{4.1}$$

where γ scales or 'tightens' the squashing function, meaning a smaller change in the input produces a larger change in the output as γ increases, and θ shifts the axis origin, or, in effect, changes the stable value (neuron output when the input and threshold registers have the same value).

Sigmoid function

Traditionally, sigmoid functions such as

$$f_1(y) = \frac{1}{1 + e^y} \tag{4.2}$$

have been used to both map an infinite range onto a finite range, and provide non-linearity to a system. This is also one of the functions suggested in [McMillen *et al.* 99]. Despite its historic significance, the sigmoid has a conceptual flaw in its inherent symmetry (figure 4.3) which renders it inappropriate for the current task. Firstly, θ must be chosen such that the steady state value is close to zero (by shifting the vertical axis to the left) so that neurons with $x = \alpha$ produce little output. Then, γ must be selected so that when one active input is received the neuron output (at the time after the input register has almost fully reacted, but before the threshold register has changed significantly), is significant (say about 0.5). The two free parameters, θ and γ, have now been fixed, so if a second significant input is added, the output will be as far from saturated as it was from zero initially. Thus further inputs at this level will only have negligible effect, which is undesirable.

Log-type function

The log type function shown in figure 4.4 was suggested [McMillen *et al.* 99, Coombes & Lord 97] with the formula

$$f_2(y) = \frac{\Theta(y)}{1 + \ln(1 + \frac{1}{y})} \tag{4.3}$$

where $\Theta(y) = 1$ for $y > 0$ and zero elsewhere. This function deals with many of the symmetry problems discussed in relation to equation 4.2. The function is many things, but it is certainly not symmetrical! Unfortunately, $f_2(y)$ moves too far in the right direction and suffers from another problem. The function is at its most sensitive in the region around the origin, where

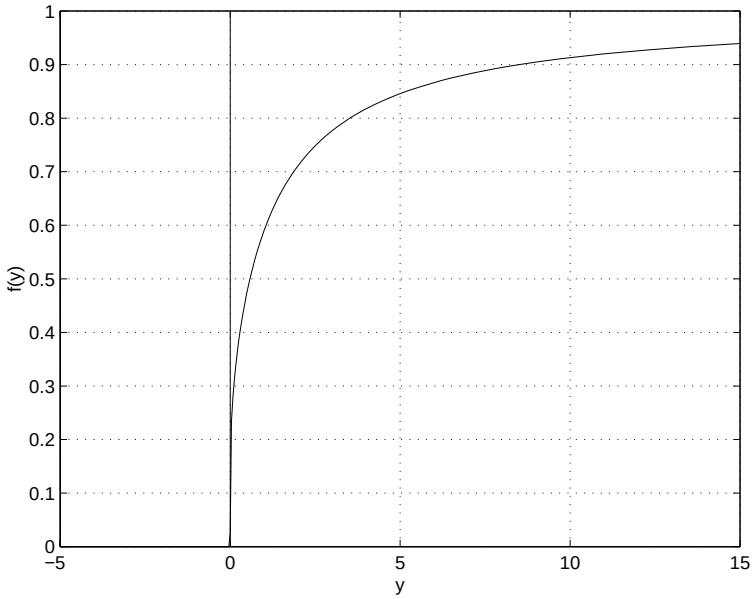


Figure 4.4: Log-like squashing function

its gradient jumps discontinuously from zero at $y = 0^-$ to being infinite at $y = 0^+$. Not only is this aesthetically displeasing, it also means that the maximum sensitivity of the neuron occurs when it has a very small output, thereby amplifying the small oscillations due to random noise. It also proves unstable even in simulation, where small oscillations due to rounding errors and finite step sizes in the equations cause loud noise, as can be clearly seen, from time 40 onwards in figure 4.5. Moving the activation point by varying the γ parameter in equation 4.1 does not remove the problem, although it may help lessen its effect.

Skew-sigmoid function

Considering the shortcomings of the previous two squashing functions led to a more rigid specification of the desired properties of a squashing function. It should (in addition to the properties discussed above):

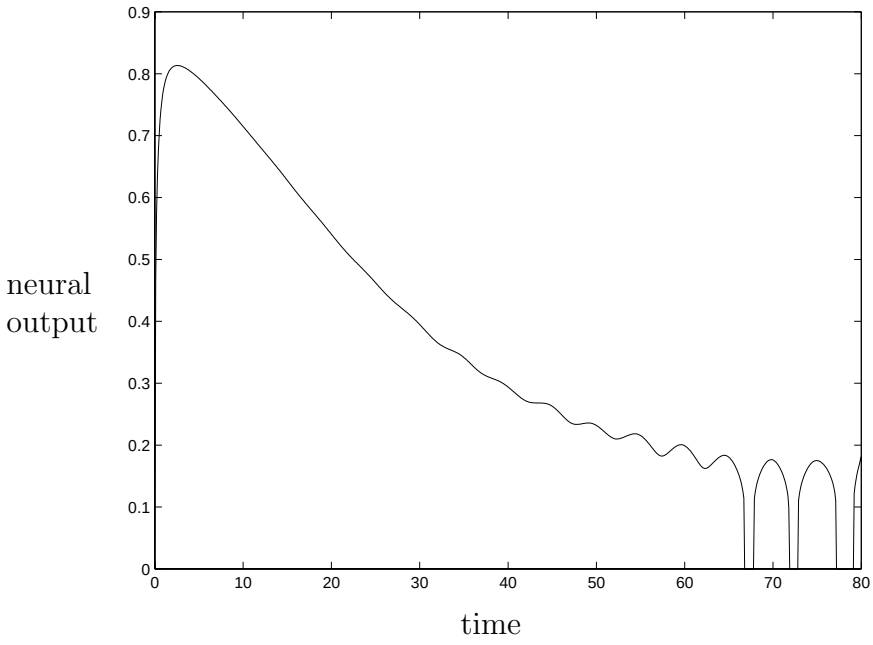


Figure 4.5: Response of neuron to a step input with a log type squashing function

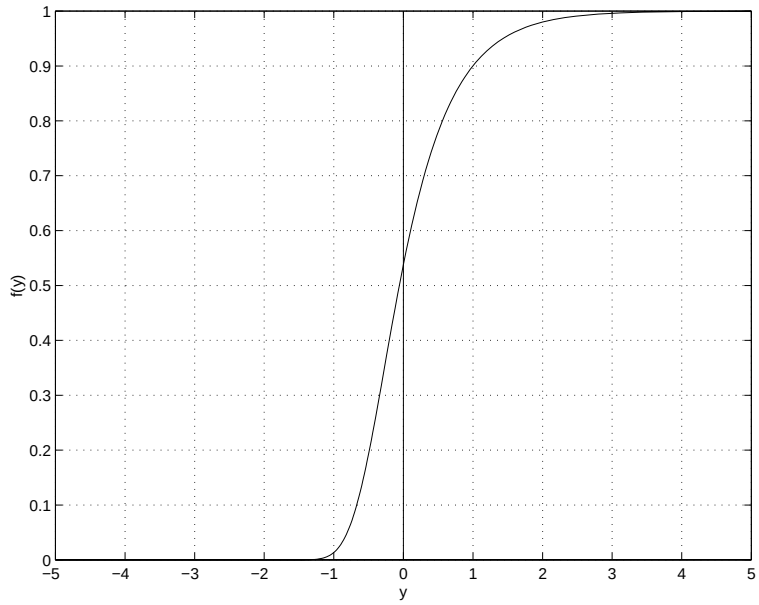


Figure 4.6: Skew-sigmoid squashing function

- Not amplify small amplitude noise.
- Provide a significant change in output for each additional input received.
- Be continuous, and have a continuous derivative if possible.

The function shown in figure 4.6 fulfils all these requirements. It is defined by:

$$f_3(y) = \frac{2}{1 + \exp(\alpha^{-y})} \tag{4.4}$$

with α being 5 in this diagram. Increasing α 'tightens' the attack of the function. Its response starts off slow to small values, rises for values in the region of zero, then decreases again at higher inputs, though more slowly than for the negative inputs. Once again, the value from the neuron can be shifted and scaled according to equation 4.1 before being squashed.

4.5 Implementation detail

A choice arose as to the best language to implement the simulation in. Hallam had a binary neuron implementation in C. This works, but translating it to analogue neurons would be a non-trivial task. Also, the implementation of the 4th order Runge-Kutta analytical differential equation solver, which carried out most of the work of the program, does not have a predictive step size algorithm which greatly slows the solution of problems. In its favour, it contains the ability to output a large quantity of diagnostic data for post simulation review.

With all software tasks, the choice of the correct language for implementation is vital for obtaining good results. Choosing a medium level language, such as C, enables very fast code to be created. Also, large numbers of standard libraries exist, and compilers can produce highly optimised code. An object orientated language, such as C++, would also make a good choice,

as the components of the system (neurons and synapses) lend themselves naturally to being described as objects.

Moving up to a higher level of language, especially one of the many optimised for scientific applications, such as Matlab, Maple or Mathematica, enables faster code generation, as support for many common operations and algorithms are included as standard. Graphical routines are also included, a major plus point compared with C. There is no such thing as a free lunch, however, and such languages pay the price in terms of efficiency (code will not run as rapidly) and portability (difficult to integrate with other applications).

Given that this work is concerned with improving the mathematical details of the neuro-connector model, not improving the efficiency of the implementation, working in a high level language that allows rapid prototyping, clear debugging and concise code to be written is more important than one which is more geared to producing fast, optimised software. Since Matlab was readily available in the department, it was selected as the development language. Although it contains some object orientated (OO) capabilities, they are not as powerful as those in C++, and, after a brief attempt at creating an OO implementation, it became clear that strict adherence to OO methodology would be impractical[6] . Despite this, the spirit of OO methodology was adopted where possible, mainly through the use of *persistent* (*static* in C) variables, and functions which could be passed flags to access or alter these variables, thereby some functions resembled objects, with various flags selecting their methods.

[6]Mainly because the standard differential equation solver used (*ODE45*) required that its parameters be given to it as a vector. This meant that all the values being changed simultaneously in the model, i.e. the input and activity register values for all neurons had to be converted to a single vector rather than accessed and changed by method functions.

4.5.1 Solving differential equations

As mentioned above, the solutions to differential equations is key to an accurate simulation of the model. The current implementation [Hallam 99], solves equations using a forth order Runge-Kutta method with fifth order error estimation (RK45), with a sub-optimal time step method. Details on the RK45 method can be found in [Garcia 94, pp 64–76]. A better solution is to use an adaptive time step, capable of estimating the error and changing its time step accordingly. The Matlab *ODE* solvers do just that, estimating a time step within the allowed error, and maintain the ability to change it if it is found to be too large.

The conceptual problem with this method is that it is unable to run in real time. In the real world, you cannot take two samples of data at different times, decide it varies too much between readings, move back in time and sample the data at an intermediary point! The alternative is to move to the electronic engineering domain of digital signal processing, where algorithms are designed to run in real time applications. The downside, however, is that these algorithms require more computational power to produce less accurate results (see also section 5.8.2). Again, since this work is concerned with mathematical improvements rather than real time implementation, the more accurate, faster RK45 algorithm has been used.

Chapter 5

Analogue synapses

This chapter contains the bulk of the new ideas developed through the course of the project and, as such, examines them in some depth. It starts by considering the properties of the synapse (section 5.1) and the essential properties of the weight update algorithm (section 5.2). Some ancillary terms are then introduced: the trace function (section 5.3) and the gating function (section 5.4), before attention can turn back to the terms of the weight update formula. These consist of a large amount on the new definition of the command register (section 5.5) and a short section on Sinclair's rest principle (section 5.6). An example of the dynamics of weight changes under several different conditions is demonstrated (section 5.7), before a final discussion of the viability of the methods used is presented (section 5.8).

5.1 Properties of synapses

As discussed in section 4.1, this project was aimed at *improving* Halperin's neuro-connector model, not creating a completely new model. In keeping with this, synapses will continue to be represented as a component which takes the output of one neuron, multiplies it by a value, and presents it as an input to another neuron, together with a mechanism for updating this multi-

plicative factor. This updated mechanism should still implement Halperin's original weight change rules (section 3.2.3) and also stick to the methodology laid out in section 4.3. In creating a mathematical implementation that fits these rules, however, it will be necessary to make the mathematics complex in some areas.

5.2 Weight update formula

The formula for the weight updates is the same as that proposed by Halperin, as given in equation 3.4,

$$\tau_{wt}\frac{dw}{dt} = \chi(\vec{\alpha})|c|(w_{tgt} - w) \tag{3.4}$$

however virtually all of the terms have been re-implemented. The left hand side of the equation still represents the change of a synapse weight with respect to time, as expected. The first component on the right hand side, $\chi(\vec{\alpha})$, still implements Sinclair's rest principle by being effectively zero when either the S or R neuron[1] has been firing recently, although both $\chi(.,.)$ and $\vec{\alpha}$ need to be redefined. The parameter c is the value of the command register, which calculates the magnitude and direction of any desired weight changes. This is the powerhouse of the method, whose new implementation (section 5.5) has been radically altered from that described in section 3.2.3. Finally, the target weight, w_{tgt}, has not been changed and still is given by the formula below.

$$w_{tgt} = \begin{cases} 1 & \text{if } c > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

Most of the remainder of this chapter will be devoted to describing the re-implementation of these terms. En route to these new definitions, a quick stop must be made to an old friend, the trace function.

[1] Here, as in section 3.2.3, S and R refer to the pre- or post-synaptic neurons respectively.

5.3 Trace functions

Trace functions are first order lag equations (as discussed in section 3.2.2), governed by equation 3.1. This implementation, like the original, will make extensive use of these trace equations, designated by the symbols $Tr_F(N)$ and $Tr_S(N)$, where the former represents a fast trace (i.e. small value of τ), and the latter a slow trace (i.e. large value of τ), of the output of neuron N. The trace functions are defined to react quickly to a rising neuron value and more slowly to a falling value, i.e.

$$\tau_{Tr}^{(\uparrow)} \; < \; \tau_{Tr}^{(\downarrow)} \tag{5.2}$$

in a similar manner to that discussed in section 3.2.2 for τ_α and τ_{inp}.

5.4 Gating function χ

The gating function, $\chi(.,.)$, gives a negligible output when either of its inputs is active. Initially this was used solely to prevent weight change when either neuron had been recently active (Sinclair's rest principle), but in the new algorithm, it will be given additional uses.

5.4.1 Problems using the old definition

Halperin's definition of the gating function (equation 3.5) works for binary neurons but has a problem with analogue neurons where an inactive neuron does not produce zero output. With the neuron parameters used in the Matlab implementation, an inactive neuron (i.e. one where the threshold equals the input) produces an output of 0.0279 (from equation 4.4). This gives a gating value of (from equation 3.5 for both S and R inactive):

$$\chi_{old}(0.0279, 0.0279) \; = \; 0.1826 \tag{5.3}$$

which is quite small. Now, consider one of the neurons being inhibited (i.e. in a refractory state), as happens when its input switches off causing the input register (x) to fall below the threshold register (α), and give a negative input to $f_3(.)$ (equation 4.4). This will cause one of the inputs to $\chi(.,.)$ to be closer to zero, giving an output of:

$$\chi_{old}(0.0279, 0) \;=\; 0.4279 \tag{5.4}$$

which is more than double the previous output. If both neurons are in their refractory period, this rises further to approximately one. Thus, when both the S and R neurons are 'inactive', the value of the gating function can vary by a factor of almost six. This is too large to be acceptable.

5.4.2 New definition

Ideally, a new gating function would not have as large a variation in output when both its inputs are neurons deemed to be inactive. A step function with a suitable cut off, say when either input is above 0.03, would fulfil this requirement but is mathematically inelegant and carries with it all of the binary baggage this project is trying to remove.

One of the closest smooth continuous function to a step function is the sigmoid function, introduced in section 4.4.4:

$$f_1(y) = \frac{1}{1 + e^y} \tag{4.2}$$

which can be scaled by

$$y = \gamma\, x + \theta \tag{4.1a}$$

to produce a function arbitrarily steep (by changing γ) and 'active' at an arbitrary value (by varying θ). Thus, equation 4.2 can approximate a step function with a discontinuity at θ by setting γ to a large value. Reducing the value of γ recovers a smooth variation between input and output. The sigmoid was selected for use as the new gating function, $\chi_{new}(.)$ (hereafter

simply referred to as $\chi(N)$, where N is a neuron), with $\gamma = -100$ and $\theta = 5$ being suitable values for the parameters for a relatively sharp cut off with a high (approximately one) output for $x < -5$ and a low (around zero) otherwise. These parameter values were selected from observations of test runs to permit the propagation of signals only when a neuron was significantly decreasing its output, not when it was undergoing natural 'wobbles', although they are not highly sensitive to small changes. A two input version can also be defined, $\chi(N, M) \equiv \chi(N)\chi(M)$, to give χ the same form as χ_{old}.

This gating function is used both in the rules to change the command register (section 5.5), and also to implement Sinclair's rest principle (section 5.6).

5.5 Command register

As in the binary implementation, the command register plays a pivotal role in the network's ability to learn.[2] In this section, some time is spent developing a new model for the command register from the bottom up. First, the inadequacy of the old command register for the new implementation is discussed (section 5.5.1), then methods for detecting changes and timings are introduced in sections 5.5.2, 5.5.3 and 5.5.4 with illustrations, before being drawn together in section 5.5.5.

5.5.1 Problems using the old definition

Halperin's command register worked well for binary neurons, although some discrepancies existed between it and her textual description, highlighted in [Hallam 00b, chapter 4, especially section 4.7]. A more pressing problem than

[2] 'Learning' can be defined as improving performance (by some measure of performance) over a set of tasks with experience [Mitchell 97, p2]. In this model, the performance measure loosely corresponds to the ability of the network to select behaviours for an agent to change it's current sensory state.

these occurs, however, in that the original implementation will only have a hope of working if a neuron switches off by changing it's output from unity to zero (consider the co-activation surface in figure 3.5; it is only positive for a small range of outputs around one). The analogue neurons described need not be giving an output of one to be 'on' and are unlikely to give an output of zero when 'off'. A new mechanism is needed.

5.5.2 Detecting neuron off times

The neuro–connector model requires that synaptic weight changes be made depending upon the temporal correlation between S and R neurons switching off. The first challenge therefore is to detect when a neuron switches off.

It makes sense to use the derivative of the neuron output as the basis for detecting when a neuron switches off, as it will be negative then. Preliminary investigations showed that, as a neuron switches off, its derivative could vary by a factor of perhaps 6 or 7, rising to very large values, again bringing up similar criticisms to those of the original gating function (section 5.4) if used directly. To avoid re-inventing the wheel, the same solution can be adapted, again using the sigmoid function with parameters γ and θ set to give a high (close to one) output for moderate to strongly negative gradients, and a low output (around zero) for small or positive gradients. Let the function $N_{off}(N)$ be defined as the sigmoid function with suitable parameters (for example, $\gamma = -2.5$ and $\theta = -5$).

(A possible alternative method is to use the difference between the current value of the output and its value a short time ago, which has the advantage that the output is bounded between plus and minus one, but the disadvantage that it reacts more slowly to neuronal changes and would register its maximum value after the neuron had switched off, rather than as it switches off. It would still require further processing to highlight only negative changes, rendering it no more advantageous.)

5.5.3 Implementing update rules 1, 2 and 4

In this section the method for determining the change in the command register due to weight update rules 1, 2 and 4, as described in section 3.2.3, is developed. As already discussed there, these rules form a continuum in the relationship between S and R off times, and their correlation can be determined as R switches off. A correlation function, $c1(t)$, will be presented which gives the change in c for these rules such that, at the time R switches off, the value of command register target is determined by the value of $c1(t)$, where $t \simeq {t_S}_{off} - {t_R}_{off}$. Here, ${t_S}_{off}$ and ${t_R}_{off}$ represent the times the S and R neurons switch off, relative to some starting time. Also, note that as the neurons switch off over a finite period of time, ${t_S}_{off}$ and ${t_R}_{off}$ are somewhat fuzzy. The dependent variable t in $c1(t)$ will be developed further when the command register target, $c^{(1)}_{tgt}$, is considered. As the correlation is being considered at time ${t_R}_{off}$, ${t_S}_{off} \leq {t_R}_{off}$, the variable t is therefore always negative. The case where S finishes after R, i.e. ${t_S}_{off} > {t_R}_{off}$, is considered in section 5.5.4.

The properties of $c1(t)$ required to implement these rules are:

1. As $t \to -\infty$, $c1(t) \to 0$ (rule 2).

2. For $t \ll -t_{exp}$ ($\equiv -t \gg t_{exp}$—remember t is always negative), $c1(t) < 0$ (rule 4).

3. When $t \geq -t_{exp}$ ($\equiv -t \leq t_{exp}$), $c1(t) > 0$ (rule 1).

In addition to these, the following properties are beneficial, although not strictly essential.

4. The value of $c1(-t_{exp})$ should be the global maximum (this can be interpreted as a mathematical definition of t_{exp} for this algorithm and, as such, is not really a new rule, provided that $c1(t)$ is a well behaved function).

5. The correlation function $c1(t)$ should be smooth and continuous (mathematically elegant). Since $c1(t)$ is only called with $t < 0$, part of it fulfilling this property is that $c1(0) = 0$.

A form of $c1(t)$ which fulfils these properties is given by:

$$t_{sc} = k_{tscale} t \tag{5.5}$$

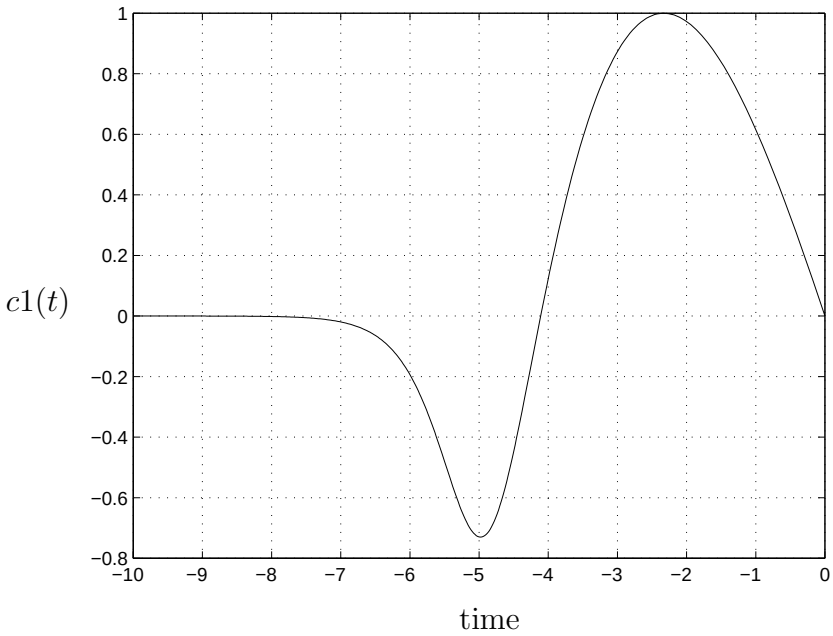
$$\begin{aligned} c1(t) = & \left(t_{sc}^3 + k_1 t_{sc}^2 + k_2 t_{sc} - k_3\right) \\ & \times \text{sig}(k_4(t_{sc} + k_5)) \\ & \times (1 - \text{sig}(k_6(t_{sc} + k_7))) \\ & \times k_{scale} \end{aligned} \tag{5.6}$$

where k_n ($n = 1, \ldots, 7$) is a parameter of the equation and $\text{sig}(y)$ is the sigmoid function given in equation 4.2. The first line of equation 5.6 is a cubic polynomial, which provides the oscillation needed for the function. Line two is the onset term which provides the ability to vary the attack of the function (i.e. response for $t \simeq 0^-$), and the next line represents the offset which brings the function smoothly to zero as $t \to -\infty$. The number of parameters may seem excessive, however it provides the ability to accurately control the shape of the curve, and not all the parameters are free: k_{tscale} is set to implement property 4 in the list above, k_3 is set so that $c1(0) = 0$ and k_{scale} is set such that $-1 \leq c1(t) \leq 1$ for all t.

Command register target $c_{tgt}^{(1)}$

The description of $c1(t)$, and the method for determining when neurons switch off developed thus far do not yet have the ability to update the command register, c. It is now time to thrash out the mathematical orchestration of these two factors to that end.

The first step is to define a storage function, $St(N, t)$ which returns the value of neuron N at time t. Obviously this function's output can only be defined for $t \leq$ the current time.



parameter	k_1	k_2	k_4	k_5	k_6	k_7	t_{exp}
value	7.8	7.8	5.0	3.0	5.0	0.25	2.0

Figure 5.1: Correlation function $c1(t)$ with value of free parameters

Thus, these rules can be implemented by setting a target for the command register, $c^{(1)}_{tgt}$, at time t to be

$$c^{(1)}_{tgt}(S,R,t) \;=\; \chi(S,R)\,Tr_F(R)\int_0^t c1(\tau,t_{exp})\,N_{\not off}(St(S,\tau))\;d\tau \qquad (5.7)$$

for a synapse connecting neuron S to R with an expectation time t_{exp}. The dependency of terms on key parameters is shown explicitly in this equation. The χ gate ensures that $c^{(1)}_{tgt}$ only has a significant value when both the S and R neurons are inactive. The trace function $Tr_F(R)$ (see section 5.3) acts as a second gate which only has a significant output if R was on a short time ago, thus $\chi(S,R)\,Tr_F(R)$ produces an update *window* for a short period when R turns off, provided S is already off. (The case where S switches off after R is discussed in section 5.5.4.)

Illustrated examples

Figures 5.2 and 5.3 show the value of the neuron outputs, the squashed derivative of the S neuron, $N_{\not off}(S)$ (see section 5.5.2), and finally the product of $N_{\not off}(S)$ with $c1(t)$—when integrated, this forms the basis of $c^{(1)}_{tgt}$ at time $t = 9.8$ (figure 5.2) or $t = 10.2$ (figure 5.3). As can clearly be seen, figure 5.2 will give a positive integrand (leading to increased synaptic weight) and figure 5.3 results in a negative integrand (wanting to give a decrease in synaptic weight) as expected.

5.5.4 Implementing update rule 3

This rule requires an update if S switches off whilst R is inactive. Again, the concept of a window defining when this occurs is useful; this time the expression defining this window is

$$Tr_F(S) - S \qquad (5.8)$$

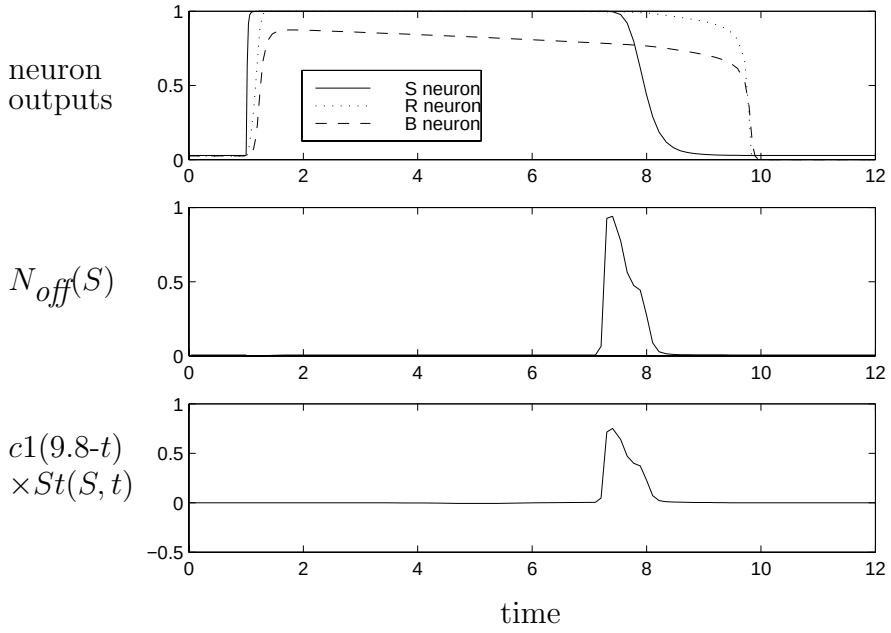


Figure 5.2: Value of command register variables for a good S–R correlation ($t_{exp} = 2$)

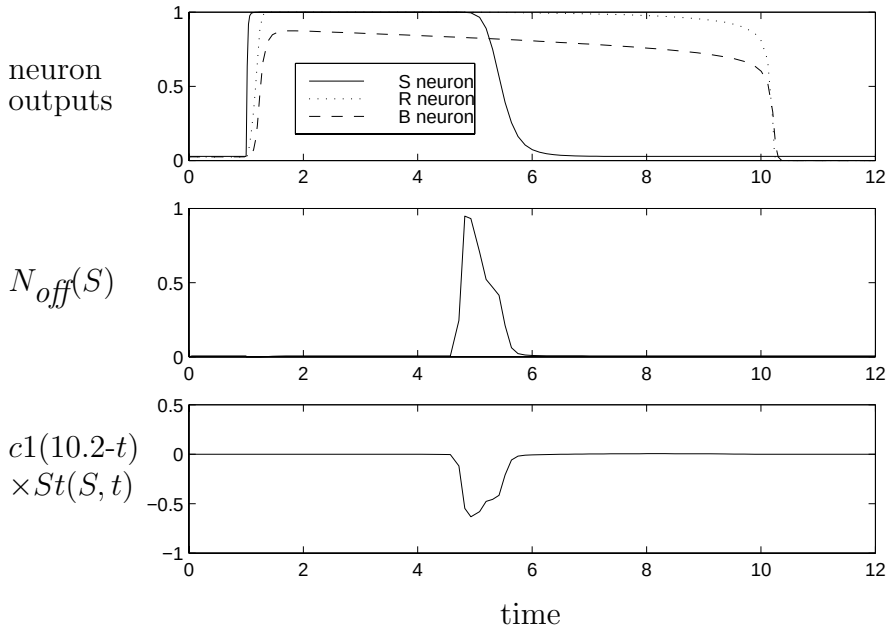


Figure 5.3: Value of command register variables for a poor S–R correlation ($t_{exp} = 2$)

which 'opens' for a short time when S changes value, giving a positive value when S switches off (or decreases) and a negative value when it switches on (or increases).

The command register target for this rule is given by:

$$c^{(2)}_{tgt}(S,R,t) \;=\; \chi(S,R)\;(Tr_F(S)-S)\;\chi(Tr_F(R)) \tag{5.9}$$

where the χ gate ensures the expression is only 'active' when both S and R are inactive. The second term forms the window discussed above, and the third is to reduce the magnitude of the decrease if R has recently been active, i.e. when S and R switch off virtually simultaneously. The exact biological response to this situation is undefined, but this implementation causes a slight decrease in synaptic weight, defendable mainly because the behaviour has already run for longer than it would if the delay was exactly t_{exp} (see section 6.1.5)

Illustrated example

Figure 5.4 shows the evolution of the factors involved in determining $c^{(2)}_{tgt}$ in a S–R–B triplet over time for an S–R synapse in which the S neuron is active without the R. Note how the second graph shows the window opening for update at around $t = 13$. The command register target is impervious to the first (negative) window at $t = 1$ because the χ gating shown in the next graph, does not permit it to propagate to the command register target[3] .

5.5.5 Command register dynamics

The key players have been introduced, but so far only as soloists, each responding to his own rhythm oblivious to the others. Now it is time to move

[3]Close inspection will reveal a small blip at $t = 1$ in the value of $c^{(2)}_{tgt}$ due to this window, however it is exists for only an infinitesimal period of time and causes no change to the weight (see figure 5.7).

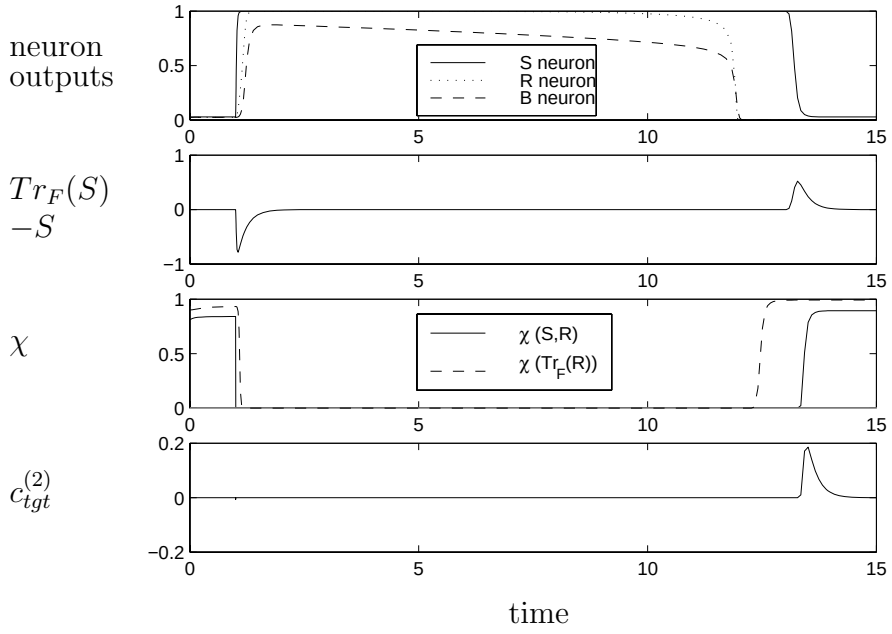


Figure 5.4: Value of command register variables for a S on without R

to the mathematical mixing deck to combine these individual voices, add some backing singers, and create a harmony from the current din.

Conducting the changes in the command register is the following equation:

$$\begin{aligned} c_{tgt}(S,R,t) &\simeq c^{(1)}_{tgt}(S,R,t) + k_{rule3}\, c^{(2)}_{tgt}(S,R,t) \\ &= \chi(S,R) \\ &\quad \times \Big(Tr_F(R) \int_0^t c1(\tau, t_{exp})\, St(S,\tau)\, d\tau \\ &\quad + k_{rule3}\,(Tr_F(S) - S)\ \chi(Tr_F(R)) \Big) \end{aligned} \tag{5.10}$$

where k_{rule3} is a scaling parameter to allow the adjustment of the relative strengths of the two sections[4] .

This equation forms the target value for the command register which

[4]The 'approximately equal to' sign ('$\simeq$') exists on the first line because a factor of $(1 - k_{rule3})\chi(S,R)$ is taken from $c^{(2)}_{tgt}$ when the expansion is taken in the next line. The expanded form is the implemented form, although the difference in insignificant for reasonable values of k_{rule3} (i.e. $k_{rule3} \approx 1$).

again is governed by a first order lag equation, modified slightly as described mathematically below:

$$\tau_c \frac{dc}{dt} = (c_{tgt} - c) \tag{5.11}$$

where

$$\tau_c = \begin{cases} \tau_c^{(F)} & \text{for } |c_{tgt}| > |c| \\ \tau_c^{(S)} & \text{for } |c_{tgt}| \leq |c| \end{cases} \tag{5.12}$$

Since most of the time c_{tgt} is approximately zero, this system causes c to move rapidly away from zero to store a new value when $\tau_c^{(F)}$ is set to a suitably small value. This value is retained in the command register, only decaying slowly under the command of $\tau_c^{(S)}$, set to a much larger value (similar to equation 5.2). Thus, when a significant value for a weight change is detected by the methods developed in sections 5.5.3 and 5.5.4, c moves quickly to a value to increase or decrease the weight accordingly. The command register then decays slowly back to zero.

5.6 Sinclair's rest principle

The only term remaining to be redefined from equation 3.4 is $\chi(\vec{\alpha})$, which implements Sinclair's rest principle. The function $\chi(.,.)$ has already been considered in section 5.4, leaving only $\vec{\alpha}$ needing a makeover.

The purpose of the $\chi(\vec{\alpha})$ term is to prevent synaptic weight changes until the neurons have been inactive for a period. Thus the natural choice for $\vec{\alpha}$ is a trace of recent inputs,

$$\vec{\alpha} = (Tr_S(S), Tr_S(R)) \tag{5.13}$$

bearing in mind that this slow trace is only slow in its decay towards zero; it reacts rapidly to increases (section 5.3). This definition provides an adequate mathematical implementation of Sinclair's rest principle, as illustrated in the graphs in section 5.7 (figures 5.5, 5.6 and 5.7).

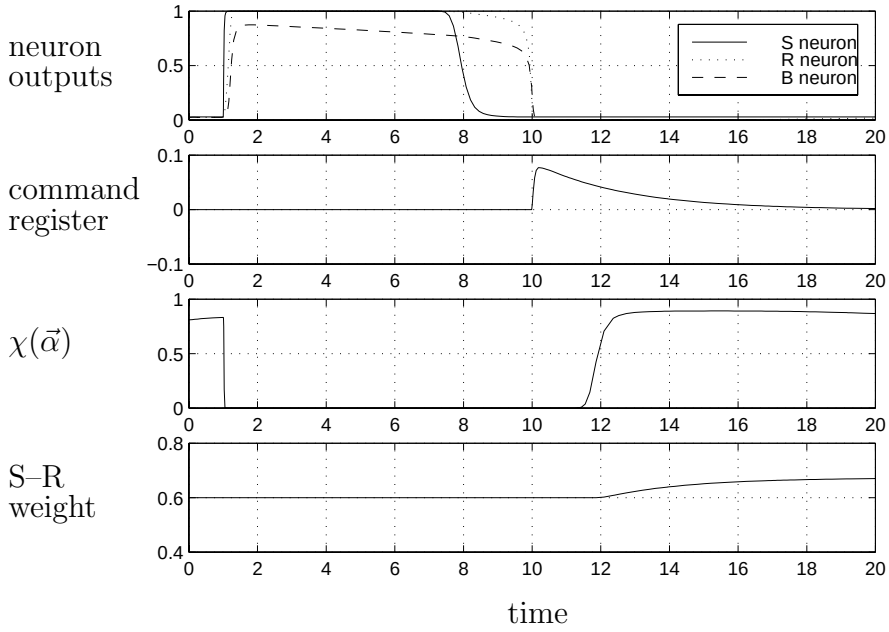


Figure 5.5: Value of weight change variables for a good S–R correlation ($t_{exp} = 2$)

5.7 Tying it all together

All the components of the neuro-connector model have now been translated into analogue form. Having spent some time working through the dynamics of the analogue neurons in chapter 4, a similar exercise on the synaptic weight update algorithm will clarify the relationship between its terms and hopefully enable any undesired (or even just unexpected) behaviour to be highlighted. The three examples considered when studying the command register (sections 5.5.3 and 5.5.4) will be carried on to investigate the weight changes.

Figures 5.5 and 5.6 show how the two main parameters in equation 3.4 (the command register, c, and the Sinclair rest function, $\chi(\vec{\alpha})$) vary with time for an S–R synapse in a simple S–R–B triplet, both with good and bad correlations between the S and R neurons. Figure 5.7 displays the evolution

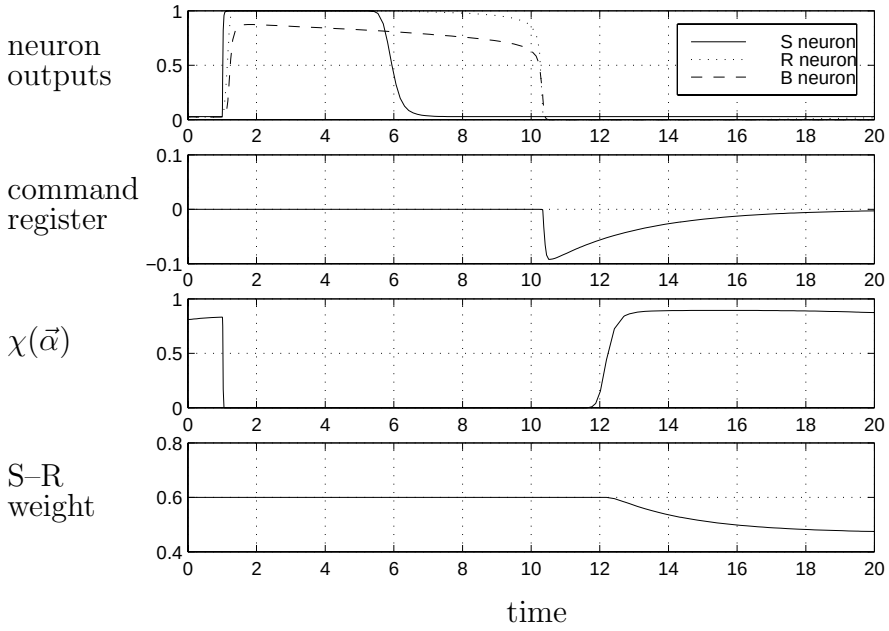


Figure 5.6: Value of weight change variables for a poor S–R correlation ($t_{exp} = 2$)

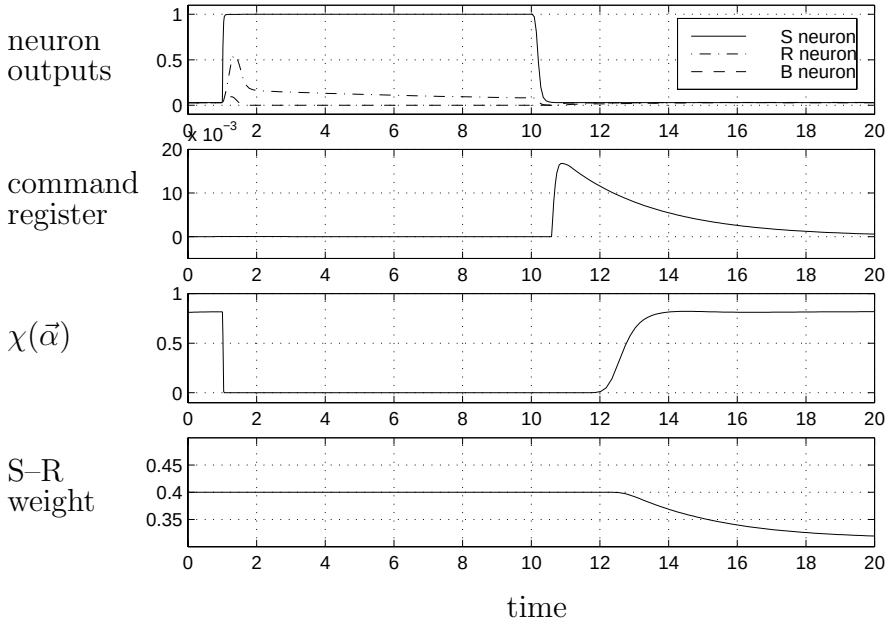


Figure 5.7: Value of weight change variables for S on without R

of a weight decrease caused by the mechanism described in section 5.5.4. Note how in each example the product of the command register (c) and $\chi(\vec{\alpha})$ opens a small window in which the weight update must occur. The onset of this window is controlled by the value of $\tau_{Tr_S}^{(\downarrow)}$ (section 5.3) which feeds into the Sinclair gating function, $\chi(\vec{\alpha})$. The natural offset depends on the decay of c, controlled by $\tau_c^{(S)}$ (equation 5.12), although the offset can be brought on prematurely by activity on either the pre- or post-synaptic neurons, via $\chi(\vec{\alpha})$ decreasing.

5.8 Evaluation

This section has implemented Halperin's rules (section 3.2.3) within the new context of an environment of analogue neurons, introduced in chapter 4. However, the mathematics is arduous at times and in places highly unlikely to be in any way biologically possible, let alone plausible. The numerous trace functions used can be defended as an approximation to short term memory of biological neural circuits (for example [Schmajuk 97, page 11]) however, of particular difficulty is formula 5.7, which calls for the multiplication of two functions over all previous times (and thus implicit storage of neural output rates over all previous times), then the integration of this new function to form a target value. How can this be defended within such a biologically based model?

5.8.1 Halperin's model

Before addressing this question, it is worth noting that Halperin's binary neuron model contains mathematics equally biologically implausible, primarily in the co-activation surface (figure 3.5) and its frequent use of the *delayed* S activity ($\alpha_S(t-t_{exp})$), thereby also needing accurate records of the activity of neurons at previous times. So moving to the analogue rules has not created

any new philosophical problems in this area.

5.8.2 Signal processing

The awkward mathematics of equation 5.7 can be recast in terms of a digital signal processing[5] (*DSP*) algorithm with the integration becoming a convolution [Bozic 94, chapter 2], producing a system capable of running in real time, yet functionally equivalent to the method suggested above. The DSP algorithm would still require a store of all (or many, depending upon exact implementation) previous results in order to calculate the high level derivatives required for the convolving function, as in general $n+1$ previous values are required to calculate the nth derivative of an input[6] . Again, this can be defended by considering an analogue (signal processing) system where neuron outputs are sampled continuously, rather than discretely, thereby enabling high order derivatives to be calculated over an arbitrarily short space of time and removing the need for storage over long periods of time.

Although functionally equivalent to the method used and perhaps more biologically plausible, a DSP implementation would be more computationally expensive and less accurate than the current method (see also the discussion in section 4.5.1).

5.9 Summary

In this chapter, the effect of the introduction of analogue neurons on the synapses has been considered. The essential properties of the weight update algorithm were reviewed (section 5.1) and the weight update formula

[5] *Digital* in this context refers to a system where signals are sampled at discrete times and stored digitally (as in a computer) and thus are subject to rounding errors etc., and must not be confused with *binary* in the sense of only being able to have two values.

[6] Also, inputs are normally sampled with a constant time interval, bringing the problems of implementation described in section 4.5.1.

revisited (section 5.1). Trace functions and gating functions were introduced (sections 5.3 and 5.4) as tools required in the redefinition of the rest of the weight update formula. The terms of the formula were then examined to determine their response to the new neurons, highlight any problems or unexpected behaviours, and solutions to those problems in keeping with the spirit of the model were suggested. First the command register, c, (section 5.5) was considered and found to require a complete rewrite to deal with the inadequacy of Halperin's co-activation function with respect to the continuous nature of outputs from the neurons. New sub-components were suggested and tested. Then the component modelling Sinclair's rest principle, $\chi(\vec{\alpha})$, was redefined (section 5.6) before some worked examples of the weight update rule in action presented (section 5.7). Finally, there is a short evaluation of the methods used and their biologically plausibility.

Chapter 6

Network dynamics

This chapter investigates the emergent properties resulting from the newly defined model components, described in chapters 4 and 5, being joined together to form a network. First, the question of what an active B neuron means is addressed and current conflicts in definitions exposed (section 6.1.1). A working definition is suggested and the properties of the network, both essential and desired, set out as a goal to aim for with the implementation (section 6.1.3). The mathematical bounds on the space of possible parameter values, subject to the constraint that they must produce behaviours consistent with these properties, are considered qualitatively in section 6.1.4. The dependency of behavioural run times on S burst length is discussed in section 6.1.5, before a new implementation of one of Hallam's classical conditioning experiments is described in section 6.2.

6.1 Behaviour adaption

6.1.1 What is a B neuron?

Confusion exists over exactly what an active B neuron represents. According to [Halperin 90, page 162], once a B fires, it triggers an action of fixed du-

ration (fixed action pattern, or *FAP*—see [Plymouth www, Cardoso www, Halperin www]), which carries on independent of action of the B. By this logic, an impulse output from a B would be all that is required for the FAP to be executed. [Halperin www] informs us that FAPs, however, can be interrupted and seems to suggest that inhibition between B neurons is the manner in which this is achieved. [Hallam *et al.* 97] makes this assumption concrete, stating that "B neurons inhibit each other so that only one behaviour happens at a time; thus it is here that behaviour selection occurs," which contradicts Halperin's initial description. [Hallam 00b, page 59] notes the initial description however with a hint of skepticism, and goes to great lengths to specify the length of time B neurons run for. Which, then, is the 'correct' implementation? It is easily possible to think of examples where each would be appropriate—an activity such as reaching for an object can be interrupted at any time, where as a reflex such as swallowing cannot normally be stopped once started.

6.1.2 Adaption time

This implementation shall follow the idea implicit in the majority of the descriptions above that an active B neuron represents carrying out a behaviour (FAP, or collection of FAPs). The FAP terminates when the B neuron becomes inactive, either prematurely via B–B inhibition, thereby ignoring the situations of the swallowing type above, or naturally by the B neuron adapting out. The active B neuron is determined as follows:

1. select the B with the largest output value then
2. compare this value with a B threshold.

If the B with the largest output is greater than this threshold, then it is selected as the active B. If not, no behaviour is selected (alternatively, a default behaviour can be carried out). This behaviour selection is purely to

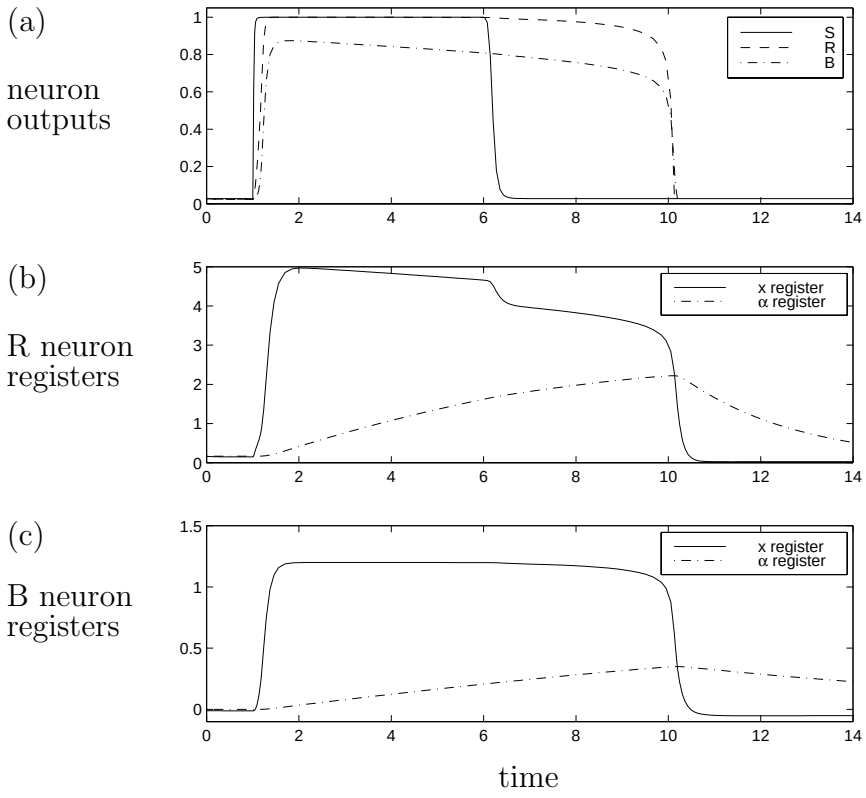


Figure 6.1: Neuron outputs and registers showing adaption time

determine which behaviour to carry out; all of the outputs from the B neurons are fed back to the associated R neurons as shown in figure 3.1 (although only feeding back from the active B could be easily implemented if desired).

The length of time a B neuron runs for in Hallam's implementation is set explicitly as an *adaption time*. If a B has been active for this length of time, it is switched off and prohibited from switching on again for a further *refractory time*.

In the analogue model, the adaption and refractory times are implicit in the network elements, namely each R–B feedback loop. Figure 6.1(a) shows how the neuron outputs change in an S–R–B triplet. The following synaptic

weights were used:

$$\begin{aligned} w_{SR} &= 0.6 \\ w_{RB} &= 0.82 \\ w_{BR} &= 5.0 \end{aligned}$$

At time $t = 1$, the S is turned on[1] , and it remains on until $t = 6$. This causes the R's input register, x_R, to rise slightly (figure 6.1(b)). Its threshold register, α_R, has a much longer response time than x_R (i.e. $\tau_\alpha \gg \tau_x$), so the R output, a function of $x_R - \alpha_R$ (section 4.4) rises. This causes a rise in x_B (via w_{RB}), which again reacts much more quickly than α_B, causing a rise in the B output. This is fed back in, via w_{BR}, to the R neuron, which causes x_R to rise rapidly and feeds into x_B rising rapidly too. The slight dip in x_R at around 6 time units is due to the S neuron switching off—more on this in section 6.1.5. From $t = 1$ onwards, the threshold registers in both the R and B neurons rise slowly toward the value of the respective x register. There exists a critical point, however, at about $t = 10$, where the rising threshold registers have lowered their neurons outputs to such an extent that the R–B feedback loop is no longer able to maintain a high output and both the R and B outputs drop rapidly.

Considering an R–B doublet, the key parameters determining the length of time the B runs for are the weights between the R and B (w_{RB} and w_{BR}) and the decay constants for the threshold registers (see section 3.2.2), $\tau_\alpha^{(R)}$ and $\tau_\alpha^{(B)}$.[2] The weights and number of S neurons inputting the R also play a role but are generally not as open to "tweaking" as these four parameters. The squashing function (section 4.4.4) also plays a key role, but again, altering it will have more knock-on effects.

[1] Achieved by giving it a large input and short response time—see section 6.1.5.

[2] It is actually the value of the τs for *rising* registers that is critical for adaption times. Their values when registers are falling control the refractory time of the behaviour.

6.1.3 Desired properties of adaption

The wide range of parameters available give the implementation a huge degree of variation depending upon their values. Before assigning values to parameters, the goals must be set so the correct behaviour of the net can be measured (or at least qualitatively judged). Below the desired properties of B run times are outlined.

1. Once started, a B neuron should run for a time approximately independent of the S inputs to its associated R neuron.

2. It should be possible to interrupt the active B neuron by another B neuron with a higher priority.

3. R neurons should require a significant input from their S neurons to commence firing.

The first property is to permit FAPs to run for a fixed length of time subject to property two, as discussed above.

The second desired property is achieved via the weights of B–B inhibitions. High priority behaviours must have higher inhibitory weighted synapses connecting them to other lower priority behaviours. The weighted input from an interrupting behaviour (or an inhibiting behaviour) must be strong enough to break the R–B feedback loop of the current behaviour. Also, "inactive" neurons do not have zero output (section 5.4.1), so the output from an inactive B neuron, when multiplied by the B–B synaptic weight, must not be enough to prevent other R–B doublets from becoming active.

The third property was included following the observation that some R–B pairs would spontaneously become active if they had high weights between them, either on their own, or after another behaviour finished (when the B would have an inhibitory input removed similar to the situation described in section 4.4.2). An additional feature was observed in this regime of strongly

coupled R and B neurons, whereby an R–B doublet would exhibit periodic oscillations[3] spontaneously. Time did not permit an investigation of this phenomena within this model, which may prove useful in modelling biorhythms such as breathing, heart beat, walking, running, etc., although again clarity needs to be taken as to what an active B neuron represents and what level the net is acting at. (I.e. does the artificial neuron represent a single biological neuron, or an entire brain region or circuit, or somewhere in between? Does an active B mean "walk forward continuously" or "raise your left leg" or even "muscle fibre #1015 contract"?) Also the assumption that only one B is active at any time would need to be reviewed if these neurons were to represent semi-automatic actions such as breathing. Some mathematical results in this area are presented in [McMillen *et al.* 99]. Much work has already been carried out in the area of neural pattern generators, for example in modelling locomotion in lampery- and salamander-like animats [Ijspeert 98].

6.1.4 Mathematical implementation

As the previous section has shown, specifying run times for each B neuron is a non-trivial task, certainly much harder than the explicitly set values in [Hallam 99]:

- The weights w_{RB} and w_{BR} must be set high enough that the loop will be self-sustaining once the S neuron (or neurons) which initiated it switch off.
- The weights w_{RB} and w_{BR} must not be set so high that spontaneous activity occurs.
- w_{RB} must be low enough that inhibition from B–B synapses can interrupt the loop.

[3]It is also not clear if the oscillations really were periodic, or if chaotic behaviour has crept in [Coombes & Lord 97, Bressloff & Coombes 98].

- w_{BR} must be large enough that the input to R is enough to render input from S neurons irrelevant once the behaviour has become active (desired property 3).

- The decay constants, $\tau_\alpha^{(R)}$ and $\tau_\alpha^{(B)}$ for rising threshold registers need to be set to give desired run times, in conjunction with the other values.

The mathematics of rigorously defining parameter values in this regime is complex and values were instead determined by an iterative method, which took some time and depends upon the values already set in the neurons (for example, parameters of the squashing function, section 4.4.4), and the number of S neurons inputting each R. The large number of parameters defines a huge space of possible mathematical instantiations of the model, each with a different behaviour. The tedious process of setting each parameter removes generality, but in real life generality is often sacrificed to get a job done too. The possibility of setting these parameter values by a genetic algorithm (*GA*) exists, but as with many GAs, finding a useful fitness function would prove difficult, and the computation required for evaluations would be immense, see section 7.3.3.

6.1.5 S–R neuron dynamics

Despite the hope of having an R–B doublet which runs for a length of time independent of its S inputs, this was not possible within the current neuron model. This section examines the dependency of the time an R–B pair is active with respect to a tonic S input.

The graph shown in figure 6.2 displays the relationship between the time an R neuron switches off, and the time its S neuron switches off in an isolated S–R–B triplet. For simplicity, the S rapidly switched on after 1 time unit, and off at the time indicated on the graph; whilst on, it did not decay but gave an output of almost one. The switching of S (either on or off) was a

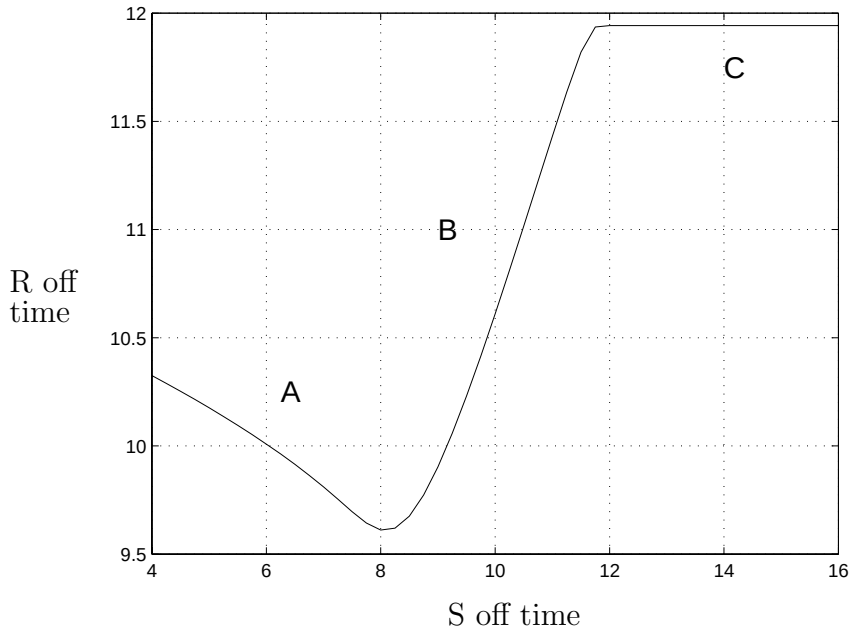


Figure 6.2: Variation in time of R switching off with S off time

decay over a very short time period, about 0.2 time units. (The top diagram in each of figures 5.5, 5.6 and 5.7 shows the evolution of the neural outputs in a similar regime.)[4]

The shape of the graph requires some explanation. Consider starting with a long S burst length (S off after 16 time units) then gradually decreasing it. In region C, the R off time is independent of S because the R adapts out (i.e. its threshold has risen above the level of its inputs, causing it to switch off and break the R–B reinforcement loop) after about 11.5 time units. Maintaining S on after this point does not change the situation.

At the boundary of regions B and C, R is on the point of no longer being able to give a large enough output to sustain the R–B loop. Reducing its input by switching off S in this region causes R to decrease its output faster,

[4]The S off times were defined as the time when the S starts to switch off, and the R off times are defined as the times when the B became inactive (as in section 6.1.2). With only one R and B, the effects of B–B inhibition are removed, and this is a good measure of the time R switches off.

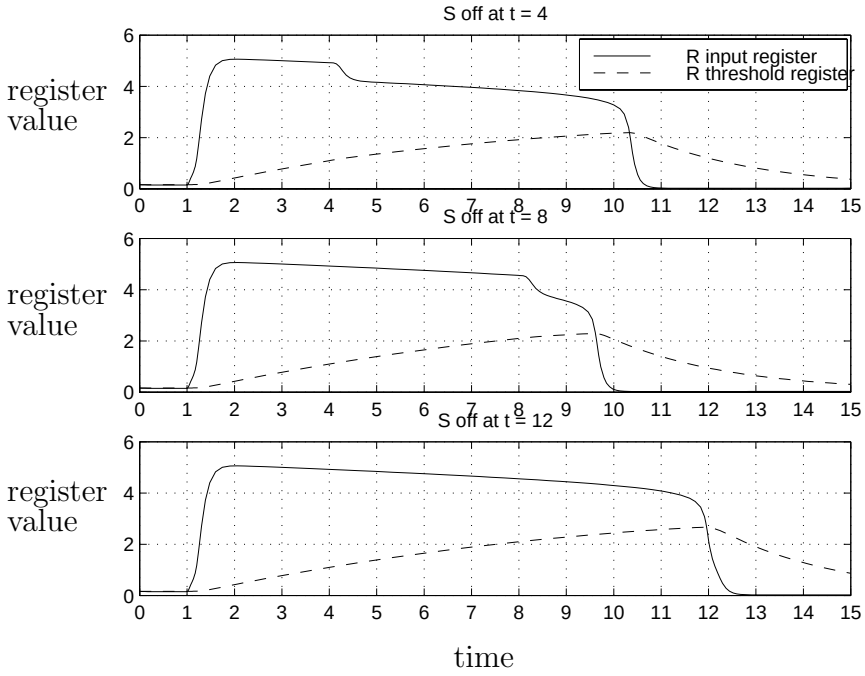


Figure 6.3: R input and threshold register evolution for various S off times

thereby the R–B loop adapts out more rapidly as the S off time is decreased in region B. Hence as S goes off earlier, so does R.

Region A, at first appears somewhat paradoxical as it shows R running on longer as S switches off earlier, contrary to the logic for region B. To solve this paradox, the internal workings of the neuron must be examined (see figure 4.1). As stated above, the output of R must stay above a critical value to maintain the activity of the R–B loop. The output is directly dependent on the difference between the input register, x, and the threshold register, α. Also, since α tends towards x (section 4.4.3), the larger the value of x, the faster α rises. Removing the input to R from S (when S switches off) causes a drop in its x register which in turn causes its α to rise more slowly. In all of region A, the S switches off before the R, so the R's x register is at an approximately constant value when it switches off, leaving its α register as the only variable. The earlier S switches off, the lower the value of α at any

time shortly after, the longer R runs for. To illustrate these subtleties the values of x and α in the R neuron are shown in figure 6.3 for S off at three different times.

6.2 Classical conditioning experiment

An attempt was made to repeat the classical conditioning experiment performed with the original model [Hallam 00b, section 5.4.1] as a brief test of the new model. This experiment was selected because it would not only test the ability of the model in the classical conditioning paradigm, but the results could be directly compared with the results produced by Hallam's software. Also, the response of the components is well defined in her experiment, as are the operating conditions.

6.2.1 Conditions

The experiment was carried out using a network of six neurons: two S neurons representing the conditioned and unconditioned stimulus (S_{CS} and S_{US}), and two R–B pairs, representing the conditioned and unconditioned responses (R_{CS}–B_{CS} and R_{US}–B_{US}). The reasons for using separate behaviour neurons for the CR and UR, rather than one B neuron representing both CR and UR, are discussed in [Hallam 00b, chapter 5].

Hallam's experiment switches the S neurons at the following times:

Time	0	5	8	10
Event	S_{CS} on	S_{US} on	S_{CS} off	S_{US} off

Due to a bug in the simulator, it is impossible to set events to occur at time $t = 0$, so each time was shifted forward by one time unit to times 1, 6, 9

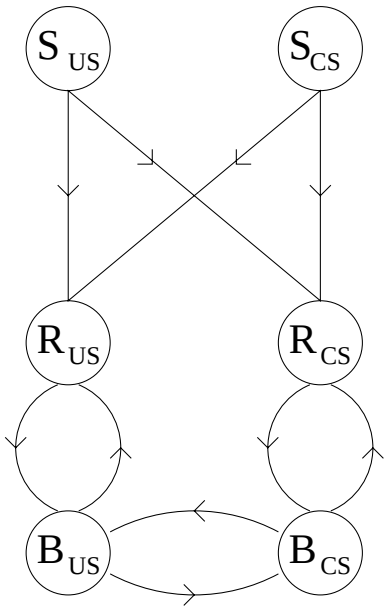


Figure 6.4: Network used in classical conditioning experiment

and 11 respectively in the following tests. The setting of all t_{exp} values to 1.7, the only value of t_{exp} in the original which actually implements the textual rules correctly, was not required as the new algorithm deals correctly with any value of t_{exp} (see chapter 5). All t_{exp}s were set to two time units for these runs. Also, the S neurons have now been correctly implemented as artificial neurons, so step switching of them is impossible—instead, when an S is set to switch on, its *input* in stepped up, typically from 0 to 4, as this gives an output of approximately one, with a short reaction time (i.e. $\tau_x^{(S)} = 0.1$ for both rising and falling values). Similarly, for switching off, the S input is reduced to zero. The S neurons are also tonic (i.e. non-adaptive) in this experiment, achieved by setting $\tau_\alpha^{(S)} = \infty$.[5]

The adaptation times for B_{CS} and B_{US} were set to 12 and 7 time units respectively, the synapses from S_{US} weighted to 1, those from S_{CS} weighted to

[5]This is possible in simulation because the parameter used is actually the inverse of τ, which can be set to zero.

0.001, and B_{US} given a higher priority than B_{CS} in the original experiment. These values were maintained as far as possible, however, as explained in section 6.1, setting the adaption times is a more complex process in the analogue implementation. The following parameter values

$$\begin{aligned} w_{RB} &= 1.2 \\ w_{BR} &= 4.0 \\ \tau^{\uparrow}_{\alpha_{R_{CS}}} &= 0.04 \\ \tau^{\uparrow}_{\alpha_{R_{US}}} &= 0.08 \\ \tau^{\downarrow}_{\alpha_R} &= 0.4 \\ \tau^{\uparrow}_{\alpha_B} &= 0.04 \\ \tau^{\downarrow}_{\alpha_B} &= 0.1 \\ w_{B_{US}B_{CS}} &= -3.0 \\ w_{B_{CS}B_{US}} &= -0.82 \end{aligned}$$

found by an iterative method, give a good approximation to the values in Hallam's experiment.

Another factor requiring slight modification was that the $w_{S_{US}R_{CS}}$ had to be set to 2 rather than 1, in order that R_{CS} did not switch off as soon as B_{CS} was interrupted by B_{US}; this need is illustrated in figure 6.6(b), where a marked drop in R_{CS} can be seen as S_{CS} switches off, at about time 9. Initially, when $w_{S_{US}R_{CS}}$ was set to 1 rather than 2, R_{CS} switched off completely at this point and reinforcement did not occur. Although this value for $w_{S_{US}R_{CS}}$ is outside the range in which the weight update rules work, it can be set as a fixed synapse to avoid this problem. Also, since the main interest is in the connection between the conditioned stimulus and response, $w_{S_{US}R_{US}}$ was also set as a fixed (i.e. reflexive) weight, thereby avoiding the need to engineer the timing of UR to make it self reinforcing.

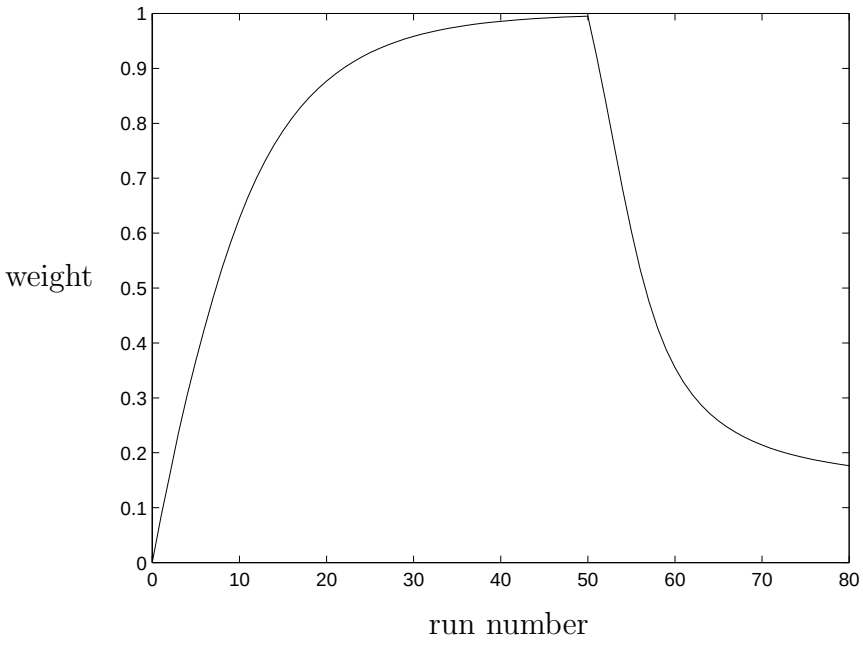


Figure 6.5: S_{CS}–R_{CS} weight throughout experiment

6.2.2 Results

Figure 6.5 shows the changes in $w_{S_{CS}R_{CS}}$ through the experiment. As can be seen, it rises quickly to a maximum of 0.9951 during the first fifty acquisition trials, in a similar manner to Hallam's tests. The extinction phase, however, diverges somewhat from Hallam's results, in that it was unable to completely unlearn the association in this experiment. It started off promisingly, showing a sharp decrease in the weight, up to about run 60. After this the rate of change decreased sharply. This is because, as $w_{S_{CS}R_{CS}}$ decreased, the burst length of R_{CS} (i.e. length of time R_{CS} is active for) *increased* by the mechanism discussed in section 6.1.5—in figure 6.6(c), the burst length is approximately 12 time units, whilst after 30 extinction runs it has increased to 14 time units, figure 6.6(d). This lengthens the time between S_{CS} switching off and R_{CS} switching off from about 4 time units to closer to 6 time units which, under these conditions, causes the simulator to implement

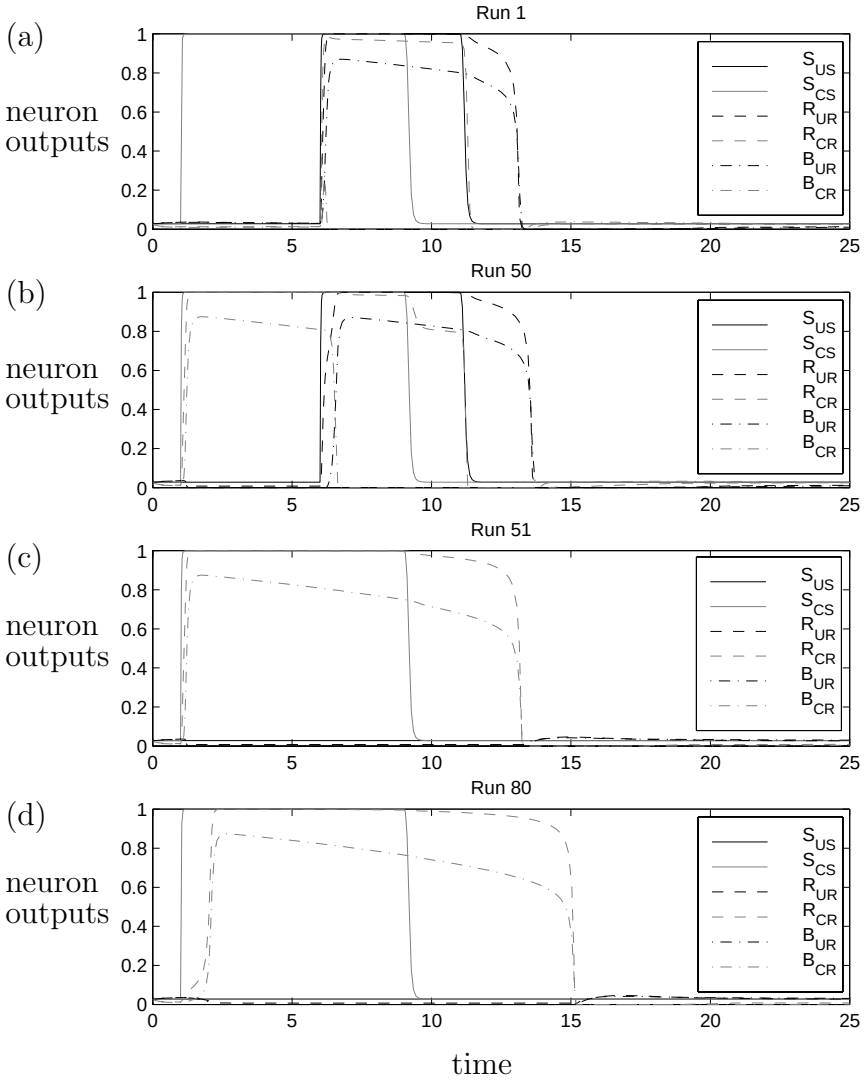


Figure 6.6: Neuron outputs (a) before conditioning, (b) after conditioning, (c) before extinction and (d) at end of experiment

Halperin's rule 3 rather than rule 4 (see section 3.2.3). That is, the software implementation treats the relationship more as an S neuron on without an R (i.e. virtually no synaptic weight change) than as a poor correlation (i.e. strong weight decrease), which was desired.

This can be overcome either by remodelling the function $c1(t)$ (figure 5.1) or by increasing t_{exp}, (which effectively stretches $c1(t)$). Time did not permit this to be tested, and also these modifications tread dangerously close to the trap [Webb 00] warns of: "it is not uncommon for models to have their parameters 'tuned' to improve the match [between simulation and target behaviour] ... this practice can in fact conceal substantial errors in the model equations or data." That is to say, with the number of free parameters in the analogue model (or the binary model for that matter) it is possible to tweak them such that any situation can be made to fit the hypothesis.

6.3 Summary

This section has provided an insight into some of the finer details of the new implementation of the model, and highlighted both new and existing problems. Even with a mathematical definition of the hypothesis, many parameters still need to be selected to enable the components to be 'glued' together. The mathematics behind these choices has been touched upon, but the problem is still not defined well enough (in terms of being able to recognise the correct behaviour when it is seen) either from the biological or engineering viewpoints (see chapter 2). Therefore even after mathematically excluding much of the parameter space, an infinite space of mathematical models is still left to pick from, with no compass to guide the choice.

The re-creation of Hallam's classical conditioning experiment, which was in itself selected somewhat artificially in order to be solvable by the binary implementation, [Hallam 00b, section 5.4], revealed shortcomings compared

to her results. This could be overcome by tweaking parameters further (i.e. selecting another instantiation from the parameter space), but this runs the risk of the new instantiation losing some desired features of the old one. The problem of selecting parameter values that will simultaneously permit one instantiation of a model to describe a large number of situations (animal learning paradigms in this case) is difficult in the extreme and well beyond the scope, or time available, in this project.

Chapter 7

Conclusion

This chapter gives a review of the work successfully completed, in the light of its original goals (section 7.1). Problems, unforeseen situations and questions arising during the course of these investigations are then summarised (section 7.2), before suggestions for future work are put forward (section 7.3), and the success of the project evaluated (section 7.4).

7.1 Aims and achievements

To recap, the aims of this project were (from section 1.2):

1. To replace the existing binary output neurons with neurons capable of a continuous output.

2. To design an algorithm to implement Halperin's update rules compatible with the new analogue neurons.

3. To test the new system on a small set of animal conditioning paradigms, similar to the simulations in [Hallam 00b, chapter 5].

The first and second aims have been carried out successfully by the mechanisms described in chapters 4 and 5 respectively. The third was only tested

on one of the animal conditioning paradigms, simple classical conditioning and extinction (section 6.2), and shown to give some promising results.

The new implementation more accurately implements Halperin's own rules (section 3.2.3) than the binary version. Specifically, the value of the expectation time is no longer constrained to being 1.7 time units to produce the behaviour consistent with these rules.

To implement the model, Matlab software was created which was as fast, if not faster, than Hallam's implementation in C, mainly because Matlab has optimised algorithms for solving differential equations, which were not used in Hallam's software (see section 4.5). Also, Matlab is a higher level language than C, is interpreted rather than compiled, and has a large number of well documented and supported libraries, making it much easier to maintain, test and alter than C (although an implementation in C with the same algorithms would be much faster). The software simulation permits sensory inputs to be altered either at a fixed point in time (as a simulation of classical conditioning) or as a result of the active behaviour (simulating instrumental conditioning).

7.2 Problems arising

During the course of this project it became clear that inconsistencies existed between the textual model and the binary implementation, primarily because the biology is not understood at a mathematical enough level to enable a full model of a biological system to be created. Some of the problems have been corrected, for example, the requirement for a fixed expectation time has been eradicated. Others have been noted, discussed and any assumptions required to produce a working implementation made explicit—most notably, the inconsistent views of B neurons (section 6.1.1) and the confusion existing over the level at which the S and B neurons operate (sections 3.3 and 6.1). Inter-

twined with these uncertainties is the idea that only one B can occur at any time thus, in the animal model, if the animal is breathing, it cannot do anything else. The preposterousness of this feature needs no further explanation, except to say that this problem must be addressed, perhaps by extracting automatic actions from the net and controlling them elsewhere. [Halperin 90] seems to imply this extraction by only discussing the control of quite high level actions. (Yet breathing in the example is only semi-automatic, so how should it be controlled? What mechanism would permit voluntary actions to override automatic reflexes?). Alternatively, several actions could be allowed to occur simultaneously, for example, one active B for each *system*. Or this may be a problem that can only be resolved on a case by case basis, guided by intuition and rules of thumb. Many other problems and implicit assumptions most likely still remain undetected and undiscussed at this stage—more may even have been inserted, however this is a side effect of not being omniscient!

7.3 Future work

Although the initial aims of the experiment have been more than fulfilled, many areas still need to be investigated to fully evaluate the new model and discern its usefulness to both biology and engineering (see chapter 2).

7.3.1 Comparison with binary model

An in-depth comparison between the performance of the analogue and binary models to ensure the modifications have not caused loss of functionality (i.e. to ensure the new model can replicate the desired results in the experiments where the first model gave outputs consistent with biology) is essential. Next testing on experiments where the binary model failed to see if they can be solved would also be beneficial—the conditioning experiments in [Schmajuk 97, chapter 2] would be a good place to start. Comparison met-

rics would also need to be defined for this due to the fact that the analogue neurons are never completely active or completely inactive, thus terms from the binary model such as 'on', 'off' and 't_{obs}' would require at least working definitions in the analogue model.

7.3.2 Hardware implementation

An attempt has been made throughout this dissertation to show the uses of the neuro-connector model as a robot controller. Reasons for needing to implement control strategies in physical bodies (rather than just simulation) to avoid focusing on irrelevant problems whilst overlooking fundamental ones, are given in virtually all robotics books. [Webb 00] also makes this case, focusing on the fact that it is impossible to simulate all environmental features, and therefore all simulators contain biases as their designers pick which features are included. Or, as [Arkin 98, page 15] succinctly puts it: "To conduct robotics research, robots are needed."

Creating a robotic implementation would also force the questions posed above regarding the level at which the model works to be addressed, including the coupling of sensors to the net, and the coupling of the net to actuators (attempted with the binary model in [Weßnitzer 00, Scott 00]). The current computational algorithms for solving differential equations can only work in simulation, so new low level algorithms would be needed for this task, perhaps from the realms of digital signal processing—see sections 4.5.1 and 5.8.2.

7.3.3 Parameter selection

As described in section 6.3, the selection of parameters is a difficult procedure, and more mathematical analysis of the parameter space to prune combinations which provide poor implementations would eliminate much of the need for the trial and error approach used in this work. Along the same lines,

genetic algorithms (GAs) have proved valuable in solving problems which involve selecting parameters for a given purpose, and so could be employed here.

The main problem with the GA approach is that a fitness function is required to automatically evaluate the strength of each implementation, which would probably involve testing it over a large number of different conditioning paradigms to avoid the problems discussed in section 6.2.2. This function would almost certainly prove difficult to define as it involves evaluating the degree of success of an implementation mathematically, most of which was carried out qualitatively in this project because the benchmark of biological behaviour is not known quantitatively.

7.3.4 Memory and the Really Useful Robot architecture

At a functional level, this model bears an uncanny similarity to the Really Useful Robot (RUR) architecture, a full description of which can be found in [Nehmzow *et al.* 89]. Briefly, in the RUR architecture, associations are learnt between sensory states (or vectors of sensory inputs) and the motor commands to satisfy (i.e. switch off) "instincts"; the learning of the associations works by carrying out a behaviour for a fixed period of time, then checking to see if this satisfied the instinct. If not, another behaviour (from a fixed set) is executed for a longer period. This is virtually analogous to the S–R learning and activity in the neuro-connector model. The RUR model has one useful extension: a memory of past sensory states. For example, if only two sensors are present, a four component sensor vector can be constructed from the current sensor states (two elements) and the value of the sensors at a previous time (another two elements). This can be expanded over numerous previous times, hence permitting the solution to the compound 'T' maze

problem to be found. (See [Weßnitzer 00] for difficulties with the current model in this situation.) A memory of previous actions may help the neuro-connector model solve these problems. (For example, by having B neurons outputs feed back in to S neuron inputs. A time-delay synapse may also be needed.)

7.3.5 Entertainment applications

An interesting application of the model would be in the computer gaming industry to devise intelligent opponents for gamers. This has the advantage that the environment is accurately defined and actions are often limited to a well defined range of character moves, so the problems of selecting appropriate S and B neurons, along with pre- and post-processing of data becomes trivial. The fast learning rate also means that the computer could adapt to particular playing style of the different users making play more interesting.

7.4 Evaluation

All the aims of the project have been successfully carried out. However, as with much of science, probably more questions have been raised than answered, and a generic model of animal learning is still a long way off. Nevertheless, such is the nature of science that the first attempted answer is rarely the last, and, in this case, neither is the second! Instead, incremental changes and adjustments need to be made to improve the models performance, and the definition of performance also needs to thought through and refined at the same time, until it produces 'satisfactory' results in a 'wide range' of experiments, or is shown to have flaws to great to be overcome by this patching, and a new model adopted.

7.5 The big picture

Back in chapter 2, at the beginning of this thesis, the neuro-connector model was placed in the context of work in robotics and biology. Now, at its end, it seems fitting to reflect on the science's broader goal of trying to "understand" the universe:

> Even if there is only one possible unified theory, it is just a set of rules and equations. What is it that breathes fire into the equations and makes a universe for them to describe? The usual approach of science of constructing a mathematical model cannot answer the questions of why there should be a universe for the model to describe. Why does the universe go to all the bother of existing? Is the unified theory so compelling that it brings about its own existence? Or does it need a creator, and, if so, does he have any other effect on the universe?
>
> —Stephen Hawking, *A Brief History of Time.*

Bibliography

[Arkin 98] Ronald C Arkin. *Behavior Based Robots*. The MIT Press, 1998.

[Bozic 94] Svetozar Mile Bozic. *Digital and Kalman Filtering*. Edward Arnold, second edition, 1994.

[Bressloff & Coombes 98] P C Bressloff and S Coombes. Desynchronization, mode locking, and bursting in strongly coupled integrate-and-fire oscillators. *Physical Review Letters*, 81(10):2168–2171, September 1998.

[Brooks 86a] Rodnet A Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, pages 14–23, April 1986.

[Brooks 86b] Rodney A Brooks. Achieving artificial intelligenge through building robots. MIT AI Memo 899, May 1986.

[Čapek 20] Karel Čapek. R.U.R. (Rossum's Universal Robots). Play, 1920. Quoted at www.pima.edu/ gmcmillan/rur.html.

[Cardoso www] Silvia Helena Cardoso. Instinctive behavior: Fixed action patterns. World wide web. www.epub.org.br/cm/n09/fastfacts/comportold_i.htm.

[Coombes & Lord 97] S Coombes and G L Lord. Desynchronization of pulse-coupled integrate-and-fire neurons. *Physical Review E*, 55(3), March 1997.

[Garcia 94] Alejandro L. Garcia. *Numerical Methods for Physics*. Prentice Hall, 1994.

[Gaudiano *et al.* 97] Paolo Gaudiano, Frank H Guenther, and Edward Zalaman. The neural dynamics approach to sensor-motor control. In Omid Omidvar and Patrick van der Smagt, editors, *Neural Systems for Robotics*, pages 153–194. Academic Press, 1997.

[Hallam 99] Bridget E Hallam. Software implementation of the neuro-connector model. Code for impliminting neuro-connector model in C, 1999.

[Hallam 00a] Bridget Hallam. Animal learning models as robot controllers. University of Edinburgh, 2000.

[Hallam 00b] Bridget E Hallam. *Simulating Animal Conditioning: Investigating Halperin's Neuro-connector model*. Unpublished PhD thesis, University of Edinburgh, 2000.

[Hallam *et al.* 97] Bridget E Hallam, John C T Hallam, and Gillian M Hayes. A dynamic net for robot control. In Omid Omidvar and Patrick van der Smagt, ed-

itors, *Neural Systems for Robotics*, pages 227–269. Academic Press, 1997.

[Halperin www] Janet R P Halperin. An outline of cognition and emotion for animals and machines. World Wide Web. www.zoo.utoronto.ca/janh/3cogem.html.

[Halperin 90] Janet R P Halperin. *A Connectionist Neural Network Model of Aggression*. Unpublished PhD thesis, Department of Ethology, Toronto University, Canada, 1990.

[Ijspeert 98] Auke Jan Ijspeert. *Design of artificial neural oscillatory circuits for the4 control of lampery- and salamander-like locomotion using evolutionary algorithms*. Unpublished PhD thesis, University of Edinburgh, 1998.

[Koch 97] Christof Koch. Debunking the digital brain. *Scientific American*, 3 Feburary 1997.

[Luger & Stubblefield 98] George F Luger and William A Stubblefield. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Addison Wesley Longman Inc, third edition, 1998.

[McMillen *et al.* 99] David R McMillen, Gabriele M T D'Eleuterio, and Janet R P Halperin. Simple central pattern generator model using phasic analog neurons. *Physical Review E*, 59(6):6994–6999, June 1999.

[Mitchell 97] Tom M Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[Nehmzow *et al.* 89] Ulrich Nehmzow, John Hallam, and Tim Smithers. Really useful robots. In T. Kanade, F.C.A. Groen, and L.O. Hertzberger, editors, *Intelligent Autonomous Systems, Proceedings of IAS 2*, pages 284–293, 1989.

[Pfeifer 96] Rolf Pfeifer. Building "fungus eaters": Design principles of autonomous agents. In Pattie Maes, Maja J Mataric, Jean-Arcady Meyer, Jordan Pollack, and Stewart W Wilson, editors, *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behanior*, pages 3–12. The MIT Press, 1996.

[Plymouth www] Department of Psychology University of Plymouth. Study and learning materials on-line. World wide web. salmon.psy.plym.ac.uk/year1/ethexpt.htm#FAP.

[Schmajuk 97] Nestor A Schmajuk. *Animal Learning and Cognition: A Neural Network Approach.* Problems in the Behavioural Sciences. Cambridge University Press, 1997.

[Scott 00] Marietta Scott. A robot implementation of Halperin's neuro-connector model. Unpublished M.Sc. thesis, Division of Informatics, University of Edinburgh, 2000.

[Shepherd 94] Gordom M Shepherd. *Neurobiology.* Oxford University Press, third edition, 1994.

[Webb 00] Barbara Webb. Are 'biorobots' good models of biological behaviour? World wide web, 2000. www.stir.ac.uk/psychology/staff/bhw1/biorobots.htm.

[Weßnitzer 00] Jan Weßnitzer. A robot implementation of Halperin's neuro-connector model. Unpublished M.Sc. thesis, Division of Informatics, University of Edinburgh, 2000.