Differentiating noise and modulators in artificial neural networks

Thesis

Philip John Docking BSc. MSc.

Differentiating Noise and Modulators in Artificial Neural Networks

Offered for the degree of PhD
at The Open University,
Milton Keynes,
U.K.

In the Discipline of Artificial Intelligence

June 1992

Date of submission : 29th July 1992

Date of award : 1st April 1993

EX12

HIGHER DEGREES OFFICE

LIBRARY AUTHORISATION FORM

STUDENT: DOCKING, PHILIP SERIAL NO: M7063177

DEGREE: PhD

TITLE OF THESIS:

DIFFERENTIATING NOISE AND MODULATORS IN ARTIFICIAL NEURAL NETWORKS

I confirm that I am willing that my thesis be made available to readers and maybe photocopied, subject to the discretion of the Librarian.

SIGNED: _____ DATE: 30·6·92

Abstract

Research in Computational Neural Networks is currently taking place at many different levels; from coarse–grain symbolic models to fine–grain representations of neurons and cell processes. One feature that the different approaches share, is that they are all in relative infancy. Thus, most research concentrates on gross aspects of neural communication and methods of computational simulation.

Recently, some clues have been found which point to more subtle mechanisms underlying the information processing capability of neural 'nodes'. These clues are the improvement in network operation by the injection of random noise; and the neurobiological finding that neuropeptides may exist as slower signal transmission channels between neurons.

This study concerns the difference between random noise injection, and directed, low–level, activity injections which are postulated to be produced by neuromodulators such as neuropeptides. The findings of this study are that random noise does, indeed, enhance the operation of coarse–grain neural models; and that a 'neuropeptidergic' analogue also enhances operation; but to a different extent, and probably through a different mechanism. Further testing of a medium–grain computer model gives some indication of how a neuropeptidergic modulation might affect real neurons, by extending the time–course of the activation of the neuron. This appears to be a similar mechanism to that postulated for the coarse–grain 'neuropeptidergic' simulation model.

Abstract

Given these findings, is it possible that signal transmission in real nervous systems assume these mechanisms? If so, it may be possible that a process of concurrent propagation, through different signal channels, also occurs in real nervous systems, making the nervous system much more complex than current models allow.

Acknowledgements

There are many people that I ought to thank for their time and energy, patience and commitment, encouragement and support; and, of course, I have already attempted to thank these people personally. However, some of the most important of these may not have realised that they were being thanked, so here I will re–iterate somewhat.

My earlier interests in Artificial Intelligence were encouraged, perhaps unknowingly, by Charlie Sharp, who has given me much support throughout the years – from allowing access to equipment which I needed to both train myself and perform research, to finding work for me when I needed the money. (But not actually to the extent of giving me a job working for him!) Nevertheless, this research is a *direct* result of his encouragement, and I would like to take this oppurtunity to thank him formally.

Longer term support (in many different quantities) has been furnished by my friends and family. I will not quantify the support I have gained from these sources, because in many regards, they are beyond measure, and in dimensions to which I

do not have ready access. Many thanks to my family. Many thanks to all of my friends, for their help.

Finally, and most importantly, my utmost thanks and gratitude goes to my very own island of sanity and warmth in an ocean of insecurity - Shehina. – to whom I owe most of all.

Contents

Contents

1

Introduction

The Neural Networks field is a multi–disciplinary and divergent field, characterised by a number of very different approaches to what is, essentially, the same subject matter. These approaches share a common ancestry, but having diverged in the later part of the 20th century, each has begun to have a different development. Characterisation of these approaches requires a somewhat false reduction and polarisation of the aims of each sub–field into primary constituents, but such reduction is necessary to cover the full scope of the field.

One of the sub–fields of the Neural Networks paradigm can be referred to as the 'classical' neural network field, made up of researchers whose main aim seems to be the production of an intelligent system, regardless of the method. This is expected to be brought about using network systems principles, sometimes established from the neurosciences, but also often from physics, and most often using a hybrid of the two. The products of this group need not have any direct relevance to the architecture of the living nervous system, and are often coded in mathematical and engineering terms. This approach merely requires an input/output transformation path, with a favourable transformation function. Another sub–field, which works along similar lines, is produced by more symbolic artificial neural network researchers and

psychologists, and is a largely symbolic approach. This is referred to as the 'Top-Down' approach, based on its focus of the display of symbol manipulation as 'intelligent behaviour', as its primary goal. This approach is also known as 'connectionism', after its theory of symbols connected by differing weight values. At the other extreme, a sub–field known as 'Computational Neuroscience', generally seeks to model the properties of single and combined neuron groups as accurately as possible, at the lowest tractable level, in order to characterise the exact workings of the living nervous system; hence its regard as the "Bottom-Up" approach. The reliance of this approach on the actual physical parameters of neural tissue also leads to the labelling of this approach as 'sub–symbolic', assuming that a 'symbol' occurs at a higher level than a single neuron. It can be said that computational neuroscience is merely continuing the work of the biological sciences, utilising modern computational tools where possible. This approach is relatively recent, hence the reference to the physics/engineering/symbolic sub–fields as 'classical' neural networks.

The Classical Neural Networks field has generated many different designs of systems which are able to learn presented patterns, auto-associate different patterns and develop topographical maps of 'concepts'. Computational Neuroscience has been able to simulate groups of hundreds of neurons, which give similar patterns of activity to that found in populations of real neurons in a particular organism. Both of these approaches rely heavily on computational power. All networks are designed to appear to operate in parallel, mimicking the operation of real nervous systems. The major bottle-neck at this time remains the

availability of parallel architecture machines capable of running large scale models in a scientifically feasible time–scale. Many researchers must rely on networks tailored to run on serial machines. If the aim is to build networks which can perform as well as those of the human brain (and thus be 'artificially intelligent'), then we must build machines which can, in reality, operate at much faster speeds than the human brain. (Consider how long it takes a child to acquire efficient language use, and the ramifications of this on the development time of a human-scale neural network.) Until such machines are made available, we must continue to establish the ground rules, by which such systems will operate.

Despite the advances made in the classical neural networks field, it seems that any further advances would require a greater degree of insight or perhaps serendipity. In contrast, at the moment, neurobiological research seems able to furnish us with more details of the mechanisms of neural transmission at an enhanced rate, and in the light of some recent discoveries, we can further develop the Bottom-Up approach, and, perhaps add some more tools to the repertoire of the Top-Down approach.

The gross organisation of the brain shows that the nervous system is a complex, interdependent network, even at the highest level of communication between specific areas of the brain. The complexity increases as lower levels of structure are revealed, at the neuronal level. This gives rise to the theory that intelligent behaviour can be produced from networks of artificial neurons. Historically, similar scientific achievements have inspired optimists to believe that animals can be fabricated, by mechanical

means, to behave exactly as their real counterparts do (see Changeux, 1985 for an interesting discussion of these attempts).

At the present time, we are learning that the neuron is not the end of the chain as far as neural processes are concerned. To understand the mechanisms of memory, we have to delve deeper into the biochemistry of the neuron and learn the chemical mechanisms by which the cell organises its behaviour.

How can such high level models as the engineering neural networks actually hope to achieve intelligence when the level of complexity inherent in the nervous system counters such a reductionistic approach? The answer lies in the expected goals in the study of this area. The entire neural networks community has to be satisfied with approximations to reality. In some cases the approximations are gross, but fast to produce and test, while at the other extreme the models are too complex to realistically produce the behaviour of a single neuron in a reasonable time–scale. The engineering neural networks field can test and explore, on a larger scale, some of the functional behaviours of networks – with the aim of producing networks which work well as pattern correlators and differentiators. This practical application of the behaviours inherent in the lower level biophysical data is an important sphere of operation. The computational neuroscientists must, necessarily, concentrate on smaller models at the sub–neuronal level, their results fuelling the projects of the classical neural networks field.

One of the themes of this thesis is that both of these approaches should be interdependent, and should be studied in parallel. Both of these fields can contribute to an increased understanding of nervous systems and artificial intelligence. On

the one hand Computational Neuroscience can, as well as elaborating on the behaviour of nervous systems, uncover previously unknown methods of network computation for network theorists; Classical Network theorists can guide computational neuroscience in looking for specific types of network activity. In concert, the two approaches should achieve an accelerated understanding of both network structures and nervous systems.

The main theme of this thesis concerns progress towards a deeper understanding of factors affecting signal transmission in a nervous network. The increase in interest of the neuron itself as a complex computational entity increases the focus of attention on the more subtle processes of the neuron. We know that not all neurons exhibit the same, or even particularly similar, behaviour. Recent neuroscientific research is showing that cellular communication systems are not limited to the action of the 'fast', 'classical' neurotransmitters; some effects seem to be produced by slower methods of communication. Some attention is also being given to aspects of the cellular environment which may have an effect on transmission within the nervous system. The nervous system must be an inherently 'noisy' place. Thermal, chemical, and, perhaps, electrical 'noise' must be a factor of any nervous system: How much of a part does this have to play in altering the efficiency of signal transmission? What is the exact nature of signal transmission in the nervous system? Is it as straightforward as the current single–channel theories presume, or do the channels cross each other to form dual– or multi–channel systems? If more than one channel is available, this may mean that the 'not so humble' neuron can manage multiple input and/or output signalling,

depending on the needs of the system. The answers to many of these questions are begun (albeit, on a minor scale) in this thesis.

Many fields have been reviewed in the preparation of this thesis. Some have not made it into this document, not because they are irrelevant to Neural Network research, but because they are not contextually relevant. Such ideas as chaos theory and associated fractal theories are now being assessed for their utility in 'node' formation stages of network design and operation, but inclusion here would have made the models over–involved and excessively slow to run. Many of the lower level compartmental model theories would also have proved difficult to implement sensibly on the available equipment. On the other hand, some of the currently fashionable classical neural network models do not lend themselves to being placed in three dimensional space - a requirement for this study – so these were not used.

Obviously, 'classical' neural network theory is of great importance in a study of this kind so this material is covered in a detailed manner, as, indeed, is the neuroscientific background.

This thesis is divided into seven chapters. The following chapter deals with the developmental history of the classical neural networks field, from the ideas of neuroscience, through the fallow period, up to the current ideas of the 'renaissance' workers. The third chapter is a parallel history of computational neuroscience ideas, artificially separated to enable a clearer discussion of developments in both fields, even though certain personalities are common to both fields.

The fourth chapter outlines the proposals of this thesis. The foundations of the theory are discussed, and some preliminary

thoughts about the mechanisms of action are proposed. The fifth and sixth chapters deal with the experimental phase: The definition of the models to be used and the results of the experimental phase and the possible ramifications of the mechanisms under discussion within the human nervous system and in the neural networks field.

The final chapter presents the findings of this study in context, and attempts to resolve some of the questions posed throughout the thesis.

PAGE HAS NO CONTENT

2

Artificial Neural Networks

The artificial neural networks field is composed of several different sub–fields. These sub–fields can be broken down on the basis of the philosophical 'view' taken by the researchers concerned. The relevant views in neural networks are usually taken as a 'level' of representation of the real neural environment, although classification is difficult, because researchers in each sub–field often indulge in variable levels of representation. At the highest level of representation (coarsest grain approach) we have 'connectionism', which is based largely on the work of psychologists. This approach uses a neural connectivity analogue to perform operations on symbolic quanta; the symbol being the lowest level of representation in the network. The 'Neural Networks' sub–field covers a multitude of approaches. Included in this sub–field are the mathematical/physical theoretical network analyses, computational/operational analyses, and other 'medium' level representations, generally defining 'nodes' or 'neurons' as the lowest level of representation in a network. The finest grain representations are those of the 'computational neuroscientists', whose field encompasses all neurological and biophysical data obtained from past neuroscientific research. The aim of computational neuroscience is to model real neural networks at the

finest level of detail possible; a largely sub–symbolic approach. Computational neuroscience is described in the next chapter. This chapter will describe the history of, what is called, 'classical' neural networks research. It is not appropriate to list all the types of models that have been developed since the beginning of the field, but some important representative models will be given. The second part of this chapter is devoted to a detailed description of the particular model to be used later in this study, the well known 'Back–propagation' model.

History

Classical Neural Networks is a fairly young field, compared to the study of neuroscience. As mentioned previously, the 'Neural networks' level of representation borrows heavily from physical, computational and mathematical sciences and so the threads are somewhat interwoven with other paradigms at different times during the development of the area. The main thrust of neural network research falls into fairly recent history, although work was being carried out in the mid–twentieth century, already based on neurological data, until the 1960's when the progress of the field was temporarily halted. This is described below. Since much of the early work has revolved around the contentions of different researchers, the headings will reflect the importance of these researchers on the development of the field.

McCulloch and Pitts

The first ventures into neural simulations began with Warren McCulloch and Walter Pitts in the 1940's, with their modelling of the theory behind the first primitive networks, consisting of binary elements (McCulloch and Pitts, 1943). This was more of a computational/engineering approach to neural modelling, although the basis of their paper was at the limits of what was known of the nervous system at the time. It is unfortunate that the only references given in their paper are for the basis of their logical symbols, so that it is now unknown from where they took the biological details upon which their theorems are based. McCulloch and Pitts attempted to describe the action of their primitive 'neurons' in terms of logical primitives. It must be assumed that this may have been done in the assumption that the brain works logically.

Interestingly enough, in their seminal paper, McCulloch and Pitts began by summarising what they knew about the physical structure of the nervous system, including, implicitly, the time variant behaviour of nervous impulses. Unfortunately, they then continued to propose their logical calculus without reference to this behaviour. In other words, they excluded the wider behaviour of the nervous system, seemingly to concentrate on the actual nervous impulse itself. It is left to the following paper in the journal (Landahl, McCulloch and Pitts, 1943) to do some kind of treatment on this. This latter paper deals with the statistics behind the mean frequencies of impulses impinging on a particular cell. This paper still deals only in the McCulloch and Pitts logic which precludes the properties of asynchrony in the nervous

system. This may not be an important consideration, if the models proposed were not intended to be representative of the biological nervous system.

The importance of McCulloch and Pitts' paper, however, is that it defines neurons in the logical rôles of AND, OR and NOT functions (See figure 2.1). This means that these logical neurons in combination should be able to perform any operation which could be performed by logic gates, or on a higher level, by modern computers. A McCulloch-Pitts neural net, with enough nodes and with enough prior experience, should be on a comparable level with a serial computer, Conversely, a serial computer large enough and with enough 'knowledge' at its disposal could show intelligent behaviour - if the McCulloch-Pitts definition of the neurons' behaviour is close enough to the way in which real neurons work. It is largely on this basis that researchers have followed this approach, even to the present day. This may be a mistake for if the sub-symbolists are 'correct', then even our symbolic, serial implementations on modern computing equipment are incapable of truly intelligent behaviour in human time-scales and spaces.

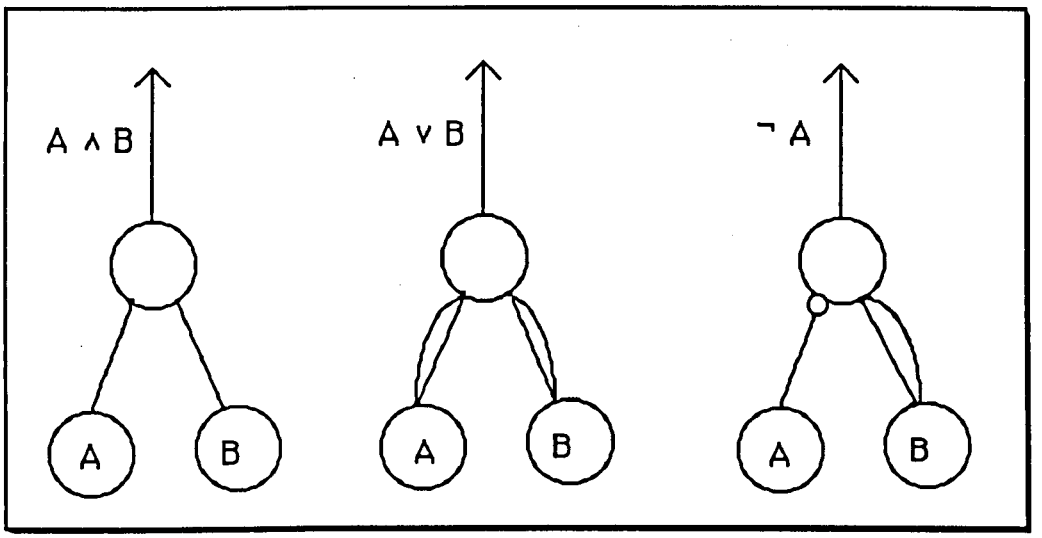

Figure 2.1 McCulloch and Pitts implementation of the basic logical functions using binary neural elements. See text for explanation.

McCulloch and Pitts' model requires that each unit has a threshold which is exceeded by the summation of the attached units, in a certain configuration. For example, in the figure above, each arc represents a value of +1, each arc terminated with a circle represents an inversion of sign for the arc. If we set the threshold value for the central unit at +2, any figure equal to or greater than +2 will 'activate' the central unit. Now, in the case of the AND unit, when both units A and B are active, this will achieve the threshold value of +2, and the central unit will become active. For the OR case, either A or B becoming active will exceed the threshold value and the central unit will become active. In the NOT case, despite the activity if the B unit, the central unit will not be active if the A unit is active, due to the sign inversion of the arc from the A unit.

Hebb

The McCulloch and Pitts approach, however, showed only the structure required for representing the solution of a logical proposition. It did not deal with the way in which the network would acquire such solutions. The work of Donald Hebb (Hebb, 1949), following the general principals of Lashley, Lorente de Nó and Cajal, was where work on a necessary theory of learning began to gain momentum. Hebb's contribution was to propose that synaptic changes were the basis of learning in the nervous system. He expressed this generally in his 'Neurophysiological Postulate' (ibid P. 62):

> *When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.*

This theory has been one of the major contributions to neural network theory. Most current models use a mechanism of this form, which allows a network to adapt to experience. There was, at the time, little evidence for this theory of synaptic change, which had many opponents. Later in his book (Ibid P.231), Hebb continues along his line of thought and proposes that synaptic decay occurs through disuse of a particular junction, although often repeated

activities make such a degeneration a much longer process. This was the theory he proposed to help explain the process of forgetting. This theory has, unfortunately, not made it into the intellectual currency of neural networks, and despite there being, apparently, no evidence against this theory, seems never to have been voiced again.

Hebb also noted some other interesting points: He believed that reverberations around a network were responsible for attention, and that the firing rate of neurons was an important factor in recognising such things as brightness in the visual field. Hebb may have been a man ahead of his time.

Through the late 1940's and early 1950's, work continued in this field. During this time, Pitts and McCulloch even proposed a method by which 'universals' might be encoded in the brain. (Pitts and McCulloch, 1947)

Minsky,Edmonds,Papert & Rosenblatt

In 1951, Marvin Minsky and Dean Edmonds produced a working 'Neural Network' using potentiometers and electronic valves. In a much quoted interview with New Yorker magazine, Minsky described the experimental process (Bernstein, 1981). Minsky's PhD dissertation was also on the subject of neural networks.

Rosenblatt appeared on the scene next, with the publication of his book "Principles of Neurodynamics" (Rosenblatt, 1962). In this, Rosenblatt (a Psychologist) summarised the work that he had been doing on the Perceptron. This network model assumed almost

mythical proportions, and became a cause célèbre, at the root of most of the argument from 1962 to the present time. The first Perceptron consisted of an array of photocells connected to an association area made up of units, whose input arrived from a random selection of photocells. These inputs were summated and passed on to 8 response units. The connections between the associator units and the response units was where the Hebbian paradigm was used. Each connection was weighted using a potentiometer positioned by automatic control (as in the Edmonds and Minsky machine). In addition, each of the response units had lateral inhibition channels, such that an input pattern would be discriminated by competitive learning. (See figure 2.2, below)

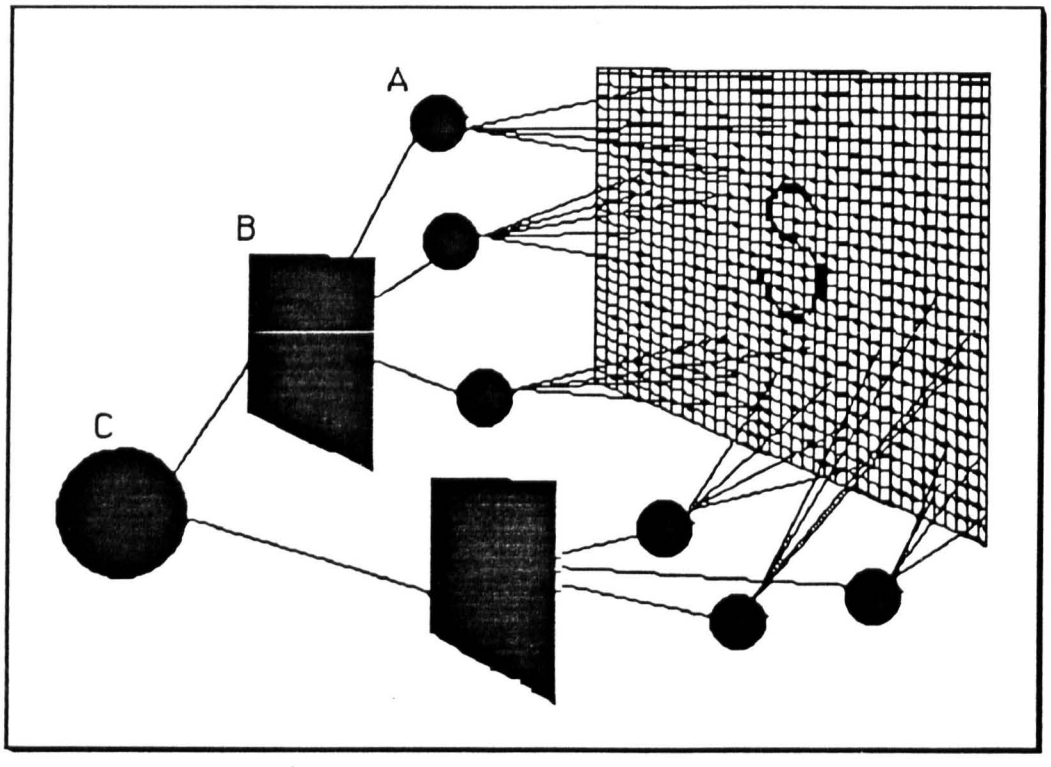

Figure 2.2 A One-layer Perceptron model. Point A is an associator unit, point B is a weighting unit and point C is the comparator.

The Perceptron was a basic two dimensional pattern recognition system, based on Rosenblatts' idea of an approximation to the brain (Rosenblatt had never intended the Perceptron to be an analogue of the human brain). In general terms, it was a trainable feature detector, where arbitrary patterns could be presented which it would 'learn' to recognise over a number of trials with a teacher. Its architecture consisted of a digitising 'retina' which transliterated light intensity on a picture plane into binary digits. These binary data were passed on to the associator unit or predicate, using a pre-specified weighting scheme which sampled the attached inputs, propagating a signal only when the summation of the number of active units reached a given threshold. A propagated signal from the associator units was passed on and multiplied by the weighting system before being compared to a threshold in a comparator. If the received value was found to be above the threshold a pattern was said to have been detected - otherwise, no pattern was detected.

The one-layer perceptron is so called because it only has one layer of modifiable connections, implemented by the weighting schemes used between the associator and the response units.

The weighting scheme used was fairly simple. The weights would be adjusted following an erroneous response in a proportional incremental manner, subtracting the error factor and the feature value, using various methods that Rosenblatt devised. It was very important that these weights should be shown to actually achieve a stable configuration within a finite time, and thus the 'Perceptron Convergence Theorem' was born.

Rosenblatt made some rather grand claims about the abilities of the Perceptron, but later in the sixties it was found to be very

hard to scale the perceptron up to cope with anything other than so called 'toy' problems.

This early sixties phase of interest in Neural Networks more or less ended after the publication of Minsky and Papert's book, "Perceptrons", in 1969, in which the one-layer perceptron was shown to be unable to perform certain types of operations, such as Exclusive–Or-ing its inputs in order to perform such functions as parity checking, and deciding on the connectedness of figures. This stems from the 'order' of a predicate formed from perceptron units. A perceptron, they argued, was only capable of forming a predicate of the first order, whilst the Exclusive-Or function requires a predicate of the second order. In addition to this, parity checking (a useful function to have in the computational world, although not necessarily present in the human nervous system) requires a non-finite order predicate, to allow for any number of input units, so that the parity could be computed.

Minsky's criticisms referred only to the one-layer Perceptron, (Minsky and Papert, 1969) and, as has been proved, multi-layer Perceptrons are capable of calculating the Exclusive-Or function, and hence, in principle, any calculation that can be performed by a Von Neumann architecture machine.

There are many authors who attribute the publication of "Perceptrons" to competition for funds and other political ends, with the 'advancement of science' as a by-product. See (Pollack, 1989).

After "Perceptrons"

Work continued in the field throughout the seventies, although funding was reputedly hard to come by. Most notable research being done by only a few workers, such as Kohonen, Grossberg, von der Malsburg, Amari, Anderson, Fukushima, Aleksander and a few others, (see the Reference section for some sampled works) but generally from within other fields than Artificial Intelligence (Psychology, Engineering, Physics.....).
Much work was done on associative memory models, and the theoretical basis of network modelling, in particular the establishment of the mathematics to be used in connection with such research.

Renaissance

The apparent rebirth of neural network research in the 1980's has been linked to the work of Hopfield (Hopfield, 1982), who, as a respected physicist, was interested in the collective computational properties of physical systems. On a physical basis, any system which exhibits locally stable states according to a particular attractor can be regarded as an associative memory. Since then, physics has found what it sees as a parallel between neural networks and other collective physical systems. In particular, the Hopfield model of neural networks is similar to the two-state model of spin glass materials. (Spin glass materials are

molecular materials with magnetic moments on each molecule, On a material wide scale these moments normally cancel each other, but on a local level they lead to stable attractor states which show local magnetic patterning.)

Interesting as it is, this model is not generally held to be biologically feasible. However, Hopfield's paper was apparently enough to rekindle interest in the wider community and bring neural network research back on a wider scale into the artificial intelligence field.

In 1982, Feldman and Ballard published a paper which proposed that symbolic Artificial Intelligence was in the throes of death and that connectionist principles should be the way forward into the study of intelligent systems. In this paper was one of the most driving observations in favour of the connectionist approach. This is, that:

> Neurons whose basic computational speed is a few milliseconds must be made to account for complex behaviours which are carried out in a few hundred milliseconds. This means that entire complex behaviours are carried out in less than a hundred time steps.

This has become an important consideration in current research, and has become known as the "100-step program" constraint (Feldman, 1985).

However, a little earlier, in 1981, David Rumelhart and James McClelland published reports postulating a spreading activation model of letter and word recognition, implying a connection between different letter and word combinations. Although these were psychological reports, it is probably a point which one could indicate as being the foetal period of the current rebirth in connectionism.

Later in the eighties, one of the more enthusiastic groups of workers under the title of the *Parallel Distributed Processing Research Group,* published a three volume set sub-titled "Explorations in the Microstructure of Cognition". (1986-1987) This group consists mostly of psychologists, under the editorship of Rumelhart and McClelland, who sought to approach connectionism from the psychological angle (and of course, other angles), including in the group some of the more active workers in the field (Geoffrey Hinton, Paul Smolensky, Terrence Sejnowski) and a few other famous names (Francis Crick, Donald Norman).

This 'PDP research group' has become a basis for a particular paradigm within connectionism, as has the Hopfield approach. There are now a number of other researchers working along different lines throughout the world, as has been noted. The Physics/Physical systems approach encompasses Hopfield and those working on 'Spin Glass' type associative networks. The Neo-connectionists, such as the PDP workers, seem to aim at a more psychological approach. The Engineering approach seems to work towards practical, immediately implementable goals; and the Neuroscientists are interested in modelling the biological nervous system, which is by no means an immediately realisable goal.

These groupings represent the authors' view of polarities in the field rather than any explicit attribution by the groups mentioned. There are other groups which encompass more than one of the above approaches (for example the engineering approach, which would probably impinge on each of the groupings), and many other groups which have not been mentioned at all.

General Characteristics of Neural Networks

There are several characteristics of current neural networks that are common to most network models: The assumptions on which most of the theory and the concrete models are based. A large majority of these assumptions come mainly from McCulloch and Pitts, and Hebb. Most researchers make some adjustments to the implementation of their own models, but generally preserve the basic structure from these sources.

These common characteristics are:

A set of elementary units - each of which exhibit similar behaviour, depending on the model in use.

A set of connections between these elementary units - these may be simple unidirectional, bidirectional or complex additive connections.

An activation state of a unit - A state of potential energy for each unit.

An activation equation for each unit - How incoming signals are interpreted by the unit. Usually an additive function.

An output rule for each unit - How outgoing signals are produced from each unit.

A propagation system - Defining how activation will be transferred throughout the system.

A learning system - Defining how the system will learn representations of input patterns.

An admirable review of how these characteristics are combined can be found in Rumelhart, Hinton and McClelland (Rumelhart, Hinton and McClelland, 1986). The general differences in properties will be outlined in the following paragraphs.

The major differences between network architectures are highlighted by the way in which the connection topology differs. However, differences also exist in the methods used to calculate and control the spread of activation throughout the network. In the first instance, differences exist in the way a processing element is allowed to calculate its activation level. Many types of unit will define their activation level as the sum of all input connections, multiplied by the weights present on each of these connections. The activation level of the unit itself will depend on what type of activation rule is in force. There are three widely used types of activation rule:

hard limiter - a binary rule which has a discontinuity threshold. If the value input to this rule exceeds a certain threshold value the unit is 'turned on'.

sigmoid - a non linear activation rule, based on an exponential function, which allows an analogue activation value, but is limited between certain values by the exponential function. Greatest change in activation value occurs at the mid-range of the input values.

Pseudo-linear - an activation rule which allows analogue ranges of values but limits the activation value between two extremes. The activation value corresponds linearly to the input over this range, however.

The output of the unit will depend on these activation levels, and will become the basis of the input to another neural unit. The output function is often combined, in the implementation, with the activation state calculation. In many cases the output function is no more than the transmission of the activation state along a connection, where the output will meet the connection weighting scheme (theoretically representative of biological synapses).

The weighting scheme is the most important function of the network. It is on the adjustment of these connection weights, that the onus of learning falls. The weighting adjustment rule is generally very similar in most networks - being that of Hebb - which is that an updated connection weight is the product of the old connection weight and the state of activation of connected neurons. That is, if a processing element takes a major part in activating a second processing element, then the weight between the two elements will increase proportionately to the part played in activating the second processing element. The calculation of how much of a part one unit takes in activating another is often

taken to be 'how much of a part one unit *should* take in activating another', and therefore the calculation of the new updated weights is based on the calculation of how much the current weight values are *in error*, compared with what values they *should* possess. This calculation is generally a global process, the error value being calculated over many units.

In general, processing units are arranged in layers. The geometry of each layer is unimportant, as each layer can represent any number of dimensions, depending on its connectivity. Figure 2.3 shows a type of regular lattice structure, similar to the way in which the innards of a crystal may be imagined. The lattice is only two dimensional, but gives a good approximation to what a network of processing units look like with their connections. Different network models may have different numbers of layers. Very few have more than three, and quite a few have no more than one layer, interconnected in such a way that it may as well be a multi-dimensional model. The connections between units are quite variable, as indicated above. In most cases the connections are uni-directional, allowing values to pass only one way, 'forward' through the network. Other networks, because of their nature (generally the error propagation networks) require bi-directional connections between units so that an error value can be passed 'backwards' along a chain of units, in order to modify the connection weights.

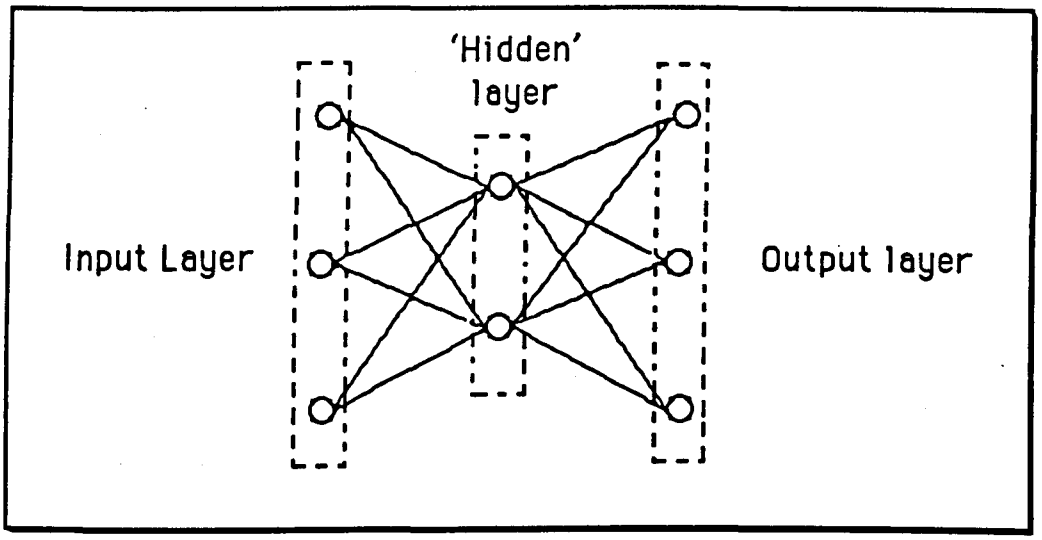

Figure 2.3 Structure of an imaginary network. Processing units are shown by circles, connections by lines.

In addition, a network (or its operating environment) must have some means of accepting input and communicating its output. These are usually taken care of by presenting input to a special case of processing units, found on an outer surface of the network (usually known as the first layer or input layer), these units will, in most cases, not have modifiable activations as this would change the input pattern (although some models, used to form generalised representations, are allowed to change the input pattern). Output is achieved by allowing another 'outer surface' of the network (the output, or last layer) to propagate out into some form of sample and hold mechanism. The values obtained may then be thresholded to obtain a representation of the activation pattern imposed on the output layer.

Inputs to a network are characterised in two different ways. They may be binary or analogue, depending on the networks' activation function. They might also be continuous or discontinuous

- in the temporal sense - and actually be 'presentations' of patterns for the network to learn, in a discontinuous manner.

A network will have different phases of its operation. The first phase will be the learning phase. This is characterised by repeated presentation of a stimulus material, usually encoded in a vector representation. The stimulus pattern is presented until the network learns to associate the pattern presented with some other pattern, or just accommodates to the input pattern (shown by some stability within the network). The first type of network is known as supervised, (because it has some kind of teacher, which 'tells' the network to associate an input pattern with an output pattern) and the second type is known as unsupervised (because it is left to associate the input pattern in any way it sees fit).

The second phase is the recall stage. This tests the storage efficiency of the network, and its ability to 'recognise' the input despite degradation in the input pattern. Performance is typically measured on an arbitrary scale, although some of the current networks, as we shall see later, calculate an error term, allowing some concrete idea of just how 'right' or 'wrong' a network can be.

Some Representative Models

The earliest popular model was, as previously reviewed, the Perceptron. Recent work has extended this into a multi-layer device, using many different computational functions, in order to produce a working, efficient system.

The structure of the Perceptron has already been discussed in general terms, but for the sake of completeness it will be dealt

with in greater detail here, as it is the basis of most other models in current use. It is interesting to note that in the prologue to the Extended edition (1988) of *Perceptrons*, Minsky and Papert relate that:

> "little of significance had changed since 1969"

and that:

> "One reason why progress has been so slow in this field is that researchers unfamiliar with its history have continued to make many of the same mistakes that others have made before them."

in this case, an exposition of the Perceptron would seem a good place to start.

Perceptron

Symbolically, the Perceptron can be represented as in figure 2.2. Each of the 512 Associator units sampled a number of the photocells from a 20x20 matrix which formed the retina, and logically combined them, the logical functions used being 'AND' and 'OR'. The output from a particular associator unit would be the result of this combination (0 or +1, depending on the function used), which would be fed into one of the eight response units. Each of the associator units had a fixed weighting associated with each input, which could be set at -1,0 or +1, randomly assigned. The

input to the response units was based on the sum of the outputs from the associator units, modified by the existing weights of the connection between the associator unit and the response unit. Thus, for each response unit:

$$\text{sum} = \sum_i w_i x_i \qquad \text{(equation 2.1)}$$

where w_i is the weight on the output from the i^{th} association unit, and x_i is the output of the i^{th} association unit. The output of the response unit was based on a comparison of the sum of inputs and a threshold value, such that the output would be +1 if the sum was greater than the positive threshold value, -1 if the sum was less than the negative threshold value and 0 if the sum was between the two values.

The learning rule which is most cited in connection with the perceptron (many were tried), is an adjustment of the weights between the associator and response units according to the degree of accuracy obtained from the output of the response units. If the output was correct, no change would be made. If the output was -1 and should have been +1, then the weight would be adjusted to "plus the value of x_i". If the output was +1 and should have been -1, then the weight would be adjusted to "minus the weight of x_i".

Adaline

The Adaline (for ADAptive LInear NEuron) was developed by Widrow and Hoff (Widrow and Hoff, 1960). It is an extension of the Perceptron in that it is intended to model the behaviour a single

neural unit, within a network of such units. In its general layout, it is similar to the Perceptron, and uses similar summation and thresholding functions. A single adaline unit consists of several input lines along which polar values are presented (-1,+1). These are combined using equation (2.1) above. This sum is compared to the threshold value, such that if the sum of inputs and weights is less than 0, the output of the unit is set to -1, otherwise, the output of the unit is set to +1.

Probably the most significant advance represented by the Adaline, however, is the scheme used to perform the learning function. This has become known as the Delta Rule (or least mean squared error rule). This is an extension of Hebb's learning postulate. In this method, the activation value of a unit is compared to the expected (or target) value of the unit, such that weight changes initiated by this method take into account the discrepancy between the actual and target values of a unit so that the modified weight will reflect the degree of influence of a particular unit in the overall output pattern, and thus the system adjusts quickly to its input. The equation is:

$$\Delta W_{ij} = e\ (t_j - a_j)\ a_i \qquad \text{(equation 2.2)}$$

where 'e' is a learning constant, 't_j' is the target activation of unit j, 'a_j' is the actual activation of unit j, and 'a_i' is the activation of the input node. An extension of this equation is used to propagate errors through several layers of neural units and is therefore called, unsurprisingly, error propagation - in this case a proportion of the error calculated to have derived from a particular layer is additionally used to modify the weighting delta.

Neocognitron

The Cognitron was developed by Fukushima, who later also developed the Neocognitron (Fukushima, 1983). These models were developed specifically for character recognition purposes. They were developed in order to simulate the process of the human neural mechanism, providing an orientation insensitive recognition machine. Their architecture borrows from Hubel and Wiesel (Hubel and Wiesel, 1965), using a nine layer model in the Neocognitron, hierarchically organised into simple, complex and hypercomplex layers. Simple cell layers have a receptive field of photocells from the 'retinal' layer, and are connected via synapse–like modifiable structures to complex cells. These complex cells have a large 'innervation' from the simple cells and a smaller number of outputs. The structure thus resembles a tree of units, each higher level with a more significant effect on the overall functioning of the network.

The biological borrowing continues with an analog activation range which is meant to be analogous to biological neural firing rates. The Neocognitron has specifically designed inhibitory cells in the complex layers, which have inputs from previous layers. These inhibitory cells provide a means of tempering the outputs of the complex layer units, which have unmodifiable synapses.

Earlier models of the Cognitron and Neocognitron were auto-associative, but later versions, intended for use in hand-written character recognition applications, are supervised systems. Each layer is reinforced separately by the tutor mechanism, which chooses a particular cell which 'should' respond to an input pattern, known as the 'representative' cell. The other cells in the

layer then have their synapses updated according to the values computed for the 'representative' cell.

The basis for the operation of the Neocognitron, as in most other models, is that the change in weights between simple and complex cells, and the inhibitory effect of the inhibitory cells on the complex cells represents the reduction in difference between the presented input pattern and the required output pattern, such that the input and output vectors become as close to co-linear as possible.

Self-Organising Map

Research into self-organising map networks has been going on for a long time. The major work in this area has been done by Kohonen (Kohonen, 1989) who currently has an application of self-organising neural networks which can produce written text from speech (in Finnish and Japanese) at over 90% accuracy.

The architecture of the self-organising map is based on the known biological facts of the human brain. That is, the cortex of the brain (when stretched outside its bony prison) seems to be a two-dimensional sheet of neurons, apparently topologically organised into different functional areas. Within each of these areas, further organisation seems to have taken place - so that in visual areas there is evidence of line segment orientation specialised regions - and in auditory areas there appears to be an auditory frequency analysis region, which itself is mapped out in order of increasing frequency.

In the network model, the neural units are ordered in a two dimensional hexagonal grid. In common with the Neocognitron its control structure selects a unit which it believes to be responding most to an input pattern, and values are propagated through the network on the basis of how physically close the neighbouring units are to the chosen unit. The weights of the connections between the units are modifiable and are adjusted according to a difference measure (how close the unit is to the input pattern) and according to a temporal measure (a scalar multiplication which decreases over time, known as the adaptation gain), and as already stated, by the physical proximity measure. The corresponding activation over the whole network can be shown, in a three dimensional representation, as a set of concentric (hexagonal) rings radiating out of the central chosen unit (looking rather like a computer generated fried egg, where height corresponds to activation, and the other two dimensions are the topology of the network). Over repeated training, each input pattern will come to be represented inside the topological map, such that a presented pattern will activate the relevant portion of the network, which can then be interpreted by a post processor.

The training of the network is a long process, requiring fine tuning by tutorial systems which analyse the expected response from the network for a given input and provides the correct response for the network to follow.

Kohonens' current application uses pre– and post–processing computational tools to introduce the input into the network and to interpret the output, but it is nevertheless, an impressive system.

Hopfield nets

Hopfield's most notable contribution to Neural Network theory, as noted earlier, was to show that fully interconnected networks of neuron-like cells worked in either a binary or an analog fashion to produce stable states. The network model with his name can be said to be a relatively simple one layer system, each unit being fully connected to the other units in the system. On the other hand, this full interconnection makes it effectively an n-layer system, where n is the number of units in the network. Each of his units is defined in terms of electronic circuitry. The neural unit itself is an operational amplifier, which produces a sigmoid output for a linear input (these may be either inverting or non-inverting amplifiers). Attached to the unit is a resistor-capacitor unit which acts as the membrane input capacitance of a biological neural cell. Resistances form the connections between the outputs of one unit to the input of another.

Hopfield networks are generally used to solve optimisation problems. These are problems that are computationally complex, such as those found in scheduling, where the amount of computation required corresponds to looking at the entire search space, before the solution is found (np-complete problems like the Travelling Salesman problem). In these cases, for example, one of a pair of feature sets of the search space are assigned to one set of neural elements, while the other set of features are assigned to another dimension of the neural elements. The interaction of these feature sets within the network quickly converges to the networks' stable state, and therefore, the solution of the problem.

Adaptive Resonance Networks

Adaptive resonance theory is the basis of work done by Grossberg (1976) and Carpenter and Grossberg (1990). This has resulted in three specific models known as ART1, ART2 and ART3. ART1 was designed as a self-organising system which responded to binary input sequence by creating category structures within the network which characterised the input sequences, without becoming trapped in local minima (a pitfall explained in the following section on back–propagation). ART2 was designed as an enhancement to ART1 - it takes analog values, as input, and self-organises, as in ART1, to find stable categorisations of its input.

Adaptive Resonance networks are interesting in that they attempt to maintain some of the basic psychological structures associated with human cognition: Attention, Orientation, Short term memory and Long term memory. These form the basis of the network architecture, including the theoretical method of rehearsal for maintaining information in short term memory.

ART networks use competitive learning and a mixture of top-down and bottom-up processing in order to classify input patterns. The long term memory remembers the previous input patterns already learned and the short term memory holds the current input pattern, after it has passed through some input filtering to enhance the contrast of the image. Thus the long term memory supplies the short term memory with the *expected* pattern which will be compared with the input pattern. If the two patterns are similar, the input pattern is classified with the pattern from the long term

memory, otherwise a new classification scheme is established into which the new input pattern is placed.

The resonance part of adaptive resonance theory comes into practice when the system has decided that the competitive learning process is complete. The representation of the input pattern is repeatedly cycled through the long term memory from the short term memory (as in psychological 'rehearsal').

ART3 is an extension of ART2, using a chemical neurotransmitter analogue system to add pre– and post–synaptic realism. (Carpenter & Grossberg, 1990)

The Boltzmann machine

Hinton and Sejnowski (1986) developed the Boltzmann machine as an extension to the basic Hopfield network architecture, in an attempt to circumvent the problem of local minima in the solution space trapping the network in a locally stable state (Hinton and Sejnowski, 1986). This model is allowed layers of neural units rather than one layer as in the original Hopfield model, and the neural units are binary threshold units with modifiable weights which are updated by a probabilistic method.

One of the major features of the Boltzmann machine is (as its name implies) the use of a 'temperature' lowering process to reach a 'thermal' equilibrium using the process of simulated annealing. It is this process which is intended to overcome the problems of local minima stability within the solution space. It does this by slowly reducing the value of a global parameter (known as the

temperature), which adds 'noise' to the system (as in physical systems - temperature adds 'noise', or random energy, to a group of atoms allowing them to vibrate more vigorously). The noise allows the learning algorithm to escape from small local minima, and, by constantly reducing the value of the 'temperature', allows the network to settle into a global minimum.

The major advantages of the Boltzmann machine are therefore, that it can generally reach a solution if there is one to be found. It has also been found to be fairly resistant to local damage, relearning lost information very quickly. However, the major disadvantage is that the initial learning is very slow, as learning must include both the stochastic procedures and simulated annealing.

Bi-directional Associative memory

A Bi-Directional Associative memory is a two layer, fully connected network which was developed by Kosko (Kosko, 1987) as a cross between Grossberg's ART and a Hopfield network. This network has modifiable synapses which are updated by a Hebbian rule. This network is interesting in that it uses only two layers, each of which may be used as an input or output layer. In operation, patterns are presented to the first layer which is then allowed to resonate between the two layers until a stable pattern is achieved on the second layer. During the recall phase, an input pattern can be presented to either of the layers and the output will be read from the non-input layer.

This model was extended to include competitive learning among the units of each layer, analog and binary values and sigmoid activation functions on all units.

CounterPropagation

Counterpropagation networks were developed by Hecht-Nielsen (Hecht-Nielsen, 1987), and is another hybrid of others' ideas - in this case, those of Kohonen and Grossberg.

A counterpropagation network as set out in Hecht-Nielsen (1987) consists of five layers. Two patterns to be associated are presented to layers one and five, and are allowed to propagate through four layers of the network. Representations of both inputs go to layer three and end on the next layer. The third layer is the part where most of the work is performed. Layer three learns by competitive inhibition, and when equilibrium is reached layers two and four will be found to have learned the average of the input vectors.

After learning, if vectors are re-presented at the input layers (1 and 5) layer three will be found to output the associative result of competitive inhibition and layers two and four will be found to contain the average of the input vectors. If a new pattern is presented, the output will show the closest match for the units at layer three, and if incomplete vectors are entered at the input layers, the network will 'fill-in' the missing elements and return the usual layers two, three and four outputs.

There is another version of the counterpropagation network which uses only three layers and is called a 'forward only'

counterpropagation network. This network takes inputs on a split input layer (ie. one vector will be input on layer 1, units 1-12, and the other vector will be input on layer 1, units 13-24, for example) and the output will be read from the second layer, which will be the competitive learning layer, while the third layer yields a transformation of the two input vectors.

Logical Neural Networks

Logical neural network models are an application oriented system devised largely by Aleksander (Aleksander, 1989). Logical node neural networks use Random Access Memory (RAM) hardware to implement a fast input/output neural network with a pattern discriminatory function. There are two versions of logic nodes, bi–state nodes and probabilistic nodes. Bi–state nodes build on the binary logic of the neuron as described by McCulloch and Pitts. If the 'neural' representation can be encoded using a McCulloch and Pitts type network, then it is also translatable into AND and OR functions within an electronic component. These logic functions are able to be stored as a truth table within a random access memory, so that incoming patterns form the input to the truth table, and the output of the truth table is the result of the sum of truth table operations. Such a system can be 'taught' to discriminate between different patterns, and even generalise among similar patterns. The WISARD (see (Aleksander & Morton, 1991)) is one example of a system built to perform such a task. The WISARD takes a digitised video picture as its input, and the

network is taught to discriminate between different categories of input by activating various output devices.

An extension of the bi–state logic node is the probabilistic logic node, which can assume three states, two known states and an indeterminate 'unknown' state. The unknown state defines the probability that an input to a particular RAM 'cell' will generate a '1' state, and this can be changed, with experience, to the most attractive state for the discriminatory operation. This extension allows greater generalisability, and introduces a simulated annealing function for the learning algorithm.

Summary

We have seen that, apart from the middle years of relative inactivity, research into neural network architectures has progressed steadily, encompassing more complex ideas from other disciplines and more efficient implementations.

In recent years, however, the back-propagation algorithm has proved itself as one of the most efficient systems in use, and possibly one of the more promising avenues of research - From its inception in 1986, extensions to the back-propagation algorithm have improved it still further, and in the following discussion we will see how this method has evolved and outline some of the recent improvements that have been made. Since Back Propagation will be used as one of the models in later chapters, its mechanism will be given a fairly detailed exposition.

The principles of Back Propagation

Back Propagation or back-prop are shortened names for "the Back Propagation of errors". This principle was introduced by Rumelhart, Hinton and Williams (Rumelhart, Hinton and Williams, 1986) of the PDP research group and is based on a variant of the Widrow-Hoff Learning Rule (1960). Since 1986, some work has been done on attempting to extend this principle, as it has been recognised as perhaps one of the more powerful, yet simple, of current designs.

A back propagation network uses a number of layers of units, without interconnection along the plane of the layer (in the Rumelhart et al model). It does, however, form bi-directional links with the previous and next layers. (See figure 2.4) Generally, the PDP models of back propagation networks have three layers, so that there is at least one layer which can perform intermediate computations, such that the network is able to deal effectively with the parity and Exclusive-OR problems, as we will see later.

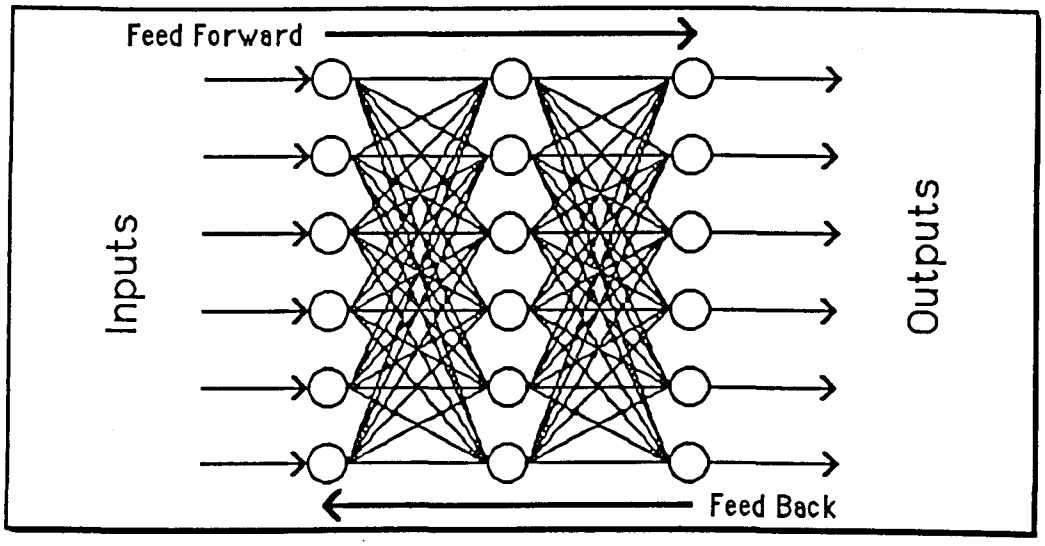

Figure 2.4 A typical arrangement of the units in a back propagation network. Note that the links are bi–directional.

Another reason for the presence of the intermediate layer is to provide an extra computational layer which allows the association of widely different input and output patterns. Such a system allows for what Rumelhart, Hinton and Williams call an "internal representation", this is a representation of the input pattern which is a sort of half-way-house between the input and output patterns. The intermediate layer is also known as the "hidden" layer, as the value of its' inputs and outputs are not directly knowable by anything outside the system.

The generalized Delta Rule, as Rumelhart et al have dubbed their version of the Widrow-Hoff (or just 'delta') rule, is important in a number of respects. Firstly, it is able to produce a solution for all the problem areas noted by Minsky and Papert, namely, the Exclusive-OR problem, the Parity problem and the Connectedness problem. It does this by the use of hidden units and its method of gradient descent into the solution space. Mathematical proofs for the nature of the solution paths of generalized delta rule models

are given in Rumelhart, Hinton and Williams (Rumelhart, Hinton and Williams, 1986) and McClelland and Rumelhart (McClelland and Rumelhart, 1988). In this treatment, we shall only go into the details of the network model using multiple layers.

Multi-layer networks using the generalized delta rule

Multi-layer networks are a special case of two layer networks. Whereas in two layer networks, a linear function can be used for activations of units, multi-layer systems gain no benefit from having additional layers using a linear activation function, as shown in Rumelhart, Hinton and McClelland (Rumelhart, Hinton and McClelland, 1986). This, they say, is because the step by step activation in linear models is merely the weighted sum of the inputs to each unit. Thus, the activation at time *t+1* is simply a function of the weight matrix times the activation at time *t.* Then, with each proceeding step the activation is only a linear function of the activation at time zero, and therefore, such a state could have been reached at time zero+1.

To overcome this limitation, back propagation models use what is known as a semi-linear activation function. This is defined as a nondecreasing and differentiable function of the net total output, which is the sum of the weights times the activations over an exclusive set of units with a particular input pattern. In this case it is also not possible to use linear activations as these are not sensibly differentiable. The semi-linear activation function used by Rumelhart, Hinton and WIlliams (1986) uses the reciprocal of a logistic term:

$$a_{pi} = \frac{1}{1+e^{-net_{pi}}} \qquad \text{(equation 2.3)}$$

(where a_{pi} is the activation of unit i for pattern p, net_{pi} is the total input to unit i)

for which the derivative is:

$$\frac{da_{pi}}{dnet_{pi}} = a_{pi}(1 - a_{pi}) \qquad \text{(equation 2.4)}$$

The non-linearity of back propagation models appears to be the only major difference between the back propagation model and the principle of the original Rosenblatt Perceptron (1962). The propagation rule is exactly the same, being the summation of inputs multiplied by the weights of the connection links:

$$net_i = \Sigma_j w_{ij} a_j \qquad \text{(equation 2.5)}$$

(where a_j is the output activation of a connected unit i and w_{ij} is the weight of the connection from unit i to unit j)

In addition, the weight updating function is the same. This is shown by equation 2.6 below:

$$\Delta w_{ij} = \eta a_i \delta_j \qquad \text{(equation 2.6)}$$

(Where w_{ij} is the weight from unit i to unit j, η is a constant known as the learning constant, a_i is the activation value of unit i and δ_j is the error from unit j.)

This brings us to the error value, which is the basic mechanism of the back propagation model. The error value is calculated from the derivative of the activation function in equation 2.4, and is the basic mechanism of gradient descent, as discussed later. The value is calculated differently, depending on whether the calculation is for the output or hidden units, since the output units depend only on themselves and a presented target value, the error signal - δ_{pi} - for an output unit will be:

$$\delta_{pi} = (t_{pi} - a_{pi})\, a_{pi}\, (1 - a_{pi}) \qquad \text{(equation 2.7)}$$

and for a hidden unit, it will be:

$$\delta_{pi} = a_{pi}\, (1 - a_{pi})\, \Sigma_k\, \delta_{pk}\, w_{ik} \qquad \text{(equation 2.8)}$$

(Where δ_{pi} represents the error function on the i^{th} unit for input pattern p, t_{pi} is the target value for output unit i, for pattern p, a_{pi} is the activation on unit i for pattern p, δ_{pk} is the error propagated back from an arbitrary unit k for pattern p and w_{ik} is the weights on the connections from the set of k units to the unit in question.)

The back propagation of the error signal is a recursive process starting with the output units and progressing backwards through the network, through the hidden units and into the first layer connections. The equation for the output units (equation 2.7) shows that the error term is largely dependent on the derivative function $a_{pi}\, (1 - a_{pi})$, (from equation 2.4) which is largest when a_{pi} has a value midway between its extremes, but is minimal when a_{pi} is at its extremes.

Gradient Descent strategy

The method of gradient descent used in the back propagation model can be explained in simple terms. Back propagation is a variant of a procedure known as Least Mean Squares (LMS) which was proposed by Widrow and Hoff (Widrow and Hoff, 1960). This system uses the Widrow-Hoff (delta) rule, the precursor of the generalised delta rule, to adjust the weight of connections in a network in order to minimise an index of the errors produced by comparing the output of the network against a target output pattern. The index of errors was taken to be the sum of squares of all errors, minimising local variations in the pattern of errors, such that:

$$E = \Sigma_p \Sigma_i (t_{pi} - o_{pi})^2 \qquad \text{(equation 2.9)}$$

(Where E is the total error of the network, t_{pi} is the target of unit i for pattern p and o_{pi} is the actual output of unit i for pattern p)

McClelland and Rumelhart (1988) produced a mapping of the solution space for all values of two weights and this type of error function, for linear and their own logistic non-linear activation functions. The mapping for linear systems resembles a valley, the gradient of which gradually gets less steep as the bottom of the valley is approached, rather like a three dimensional hyperbolas. (See figure 2.5) The minimum error function - the Least Mean Square of the system - resides at the very bottom of the valley. The gradient descent function is designed so that a solution can be reached quickly, (so that the minimum error position is reached

quickly) and therefore requires that when the gradient of the weight to error mapping is steepest (which is further away from a solution) a larger step can be taken in the solution space (hopefully, towards the solution). This means that to take a larger step, we must increase or decrease the weight of the connection by a larger proportion when the gradient is steep, than when the gradient is shallow. As figure 2.5 illustrates only a simple linear system with two weights, it is a fairly simple mapping. However, it is not always possible, with more weights and with non-linear systems to produce a mapping, and therefore, we require a mathematical function to calculate gradients local to the units when we are attempting to assign a new weight in order to achieve a solution. McClelland and Rumelhart (1988) use the negative of the error derivative to do this. In the case of the generalised delta rule (which is non-linear) the weight of the connection is changed according to equation 2.6.

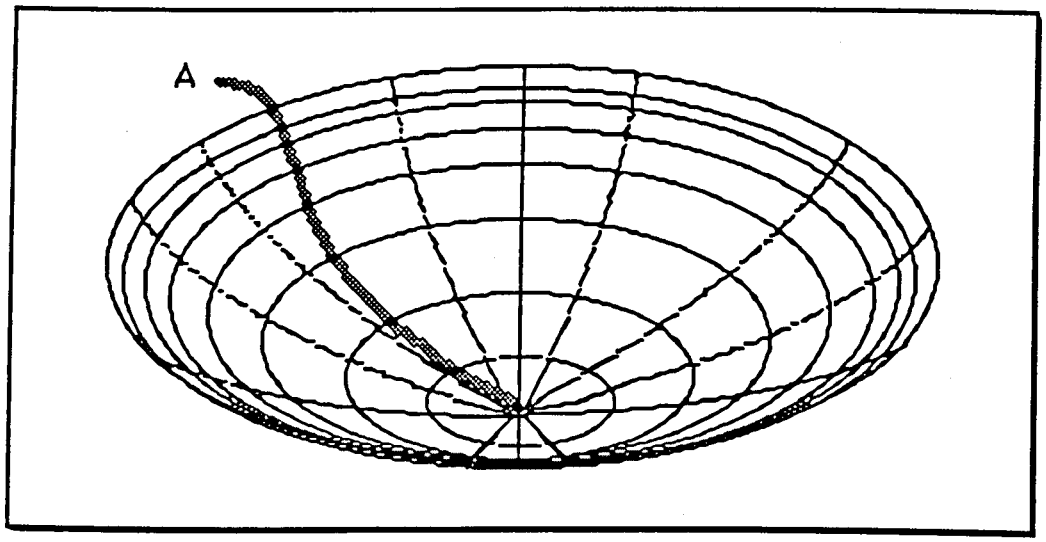

Figure 2.5 A map of the solution space produced by a two weight system, produces by mapping weights against LMS error. An imaginary problem starting at point A, proceeds less steeply as a solution is reached.

It is important to emphasise that the mapping in figure 2.5 is of a simple two weight, linear system. The mapping for a two weight non-linear system shown in McClelland and Rumelhart (1988) resembles a saddle shaped figure, which means that there is *more than one* minimum value for the error measure. This means that depending on the starting values of the system, it is possible that the system will move towards a *local* minimum at another point in the solution space and give the wrong answer! In addition to this, the mapping shown in figure 2.5 is actually only a mapping for a particular problem - Each new problem has its own different solution space mapping.

The learning rate of the system is also an important factor in the gradient descent strategy. Equation 2.6 shows the equation used to update the weight of each connection, detailing the effect

of the three terms; unit activation, error function (which we have just dealt with) and the learning constant η . The effect of the learning constant is to proportion the changes in connection weights which determines how large the steps down into the solution space (of, for example, figure 2.5) will be. The learning rate is generally a value between zero and one, and is, in general, closer to zero than to one, so that changes are as small as possible, without leading to a ridiculously long learning time. If the learning rate were closer to a value of one, we run the risk of producing larger jumps in the solution space than are really safe, meaning that the system may find itself in a situation where oscillation sets in (perhaps across the bottom of a valley or across the peak of a local hillock) - leading to an inability to find a solution. In practice, it is tempting to use a large learning constant to improve the learning rate of the system.

A final problem in setting up a gradient descent network system is the problem of the initial weights of the network connections. In this symmetrical back propagation model, the inclusion of the connection weights in the calculation of the error term means that if all the connection weights were to begin with the same value, all units in a particular layer would receive the same error value from the preceding layers' error calculation, and the network would fail to develop unequal weights. The solution to this problem is to initially assign random values to the connection weights, which starts the network in a random position in the solution space, and is not generally harmful, unless this position is close to a local minimum.

Extensions to the back propagation model

Momentum

The first extension to be made to this model was included in Rumelhart and McClellands' (1986) original proposal. This is the use of a momentum function inside the generalised delta rule. The momentum term is used as an accelerator for the learning rate, without causing the problems described earlier, when just increasing the learning constant. The momentum function is a kind of history mechanism. It remembers the weights from the previous error pass of the network, and uses this value to proportion the change made to the weight during the current iteration. So the extended generalised delta rule (from equation 2.6) looks like this:

$$\Delta w_{ij(n+1)} = \eta\ a_i\ \delta_j + \alpha \Delta w_{ij(n)} \qquad \text{(equation 2.10)}$$

which just adds in a proportion of the weight from the previous iteration, depending on the value of α. The effect of momentum is to filter out high frequency variations in the solution space. These high frequency variations are caused by sharp curvatures in shallow trenches within the solution space, which cause oscillations across the trench.

Activation range

A second extension proposed by Dahl (Dahl, 1987) was to extend the activation range of the neural unit. Dahl found that the extension from a three point scale used in Rumelhart et al (1986) to a four point scale enhanced the learning rate of the network under most conditions. The four point scale used was from -1 to +2, essentially widening the discriminability of the network.

Added noise

Von Lehman, Peek, Liao, Marrakchi and Patel (Von Lehman, Peek, Liao, Marrakchi & Patel, 1988), of Bell Communications research proposed the extension of adding noise to the connection weights during the update process, and clamping weights at extreme values. They found that when noise is added, instead of a momentum term, they could build networks (for the XOR problem) which had a 100% probability of convergence. The noise term was generated randomly and added to the normal calculation of the weight at each iteration. Their figures show that a wide range of noise values could be used to achieve the 100% convergence rate.

These researchers also found that the initial weighting of a network significantly affected the performance of the network. They tested three different ranges of initial weight settings and found that although the performance graph produced is grossly similar, there are significant deviations in the probability of convergence.

Calculation of hidden unit numbers and learning rate

Kung and Hwang (Kung & Hwang, 1988) produced a method of determining how many hidden units to use in optimising a network, and how the optimal learning rate should be assigned to the network. They used simulation methods to prove that the optimal number of hidden units is dependent on the regularity of the input pattern. They determined that patterns which are entirely irregular should use the same number of hidden units to input units, whereas patterns with definite regularities could use a lesser amount of hidden units, in the same ratio to the input units as the patterns of regularity within the input patterns.

Solutions to classical problems posed to the neural networks community

The XOR problem

The Exclusive-OR problem was recognised by Minsky and Papert (1969) in relation to the perceptron. The perceptron was shown to be able to solve problems of the first order, but not able to solve higher order problems. First order problems are those where the sum of products for a predicate are not more than 1. Needless to say, Exclusive-OR is of second order, ie. it needs a unit which combines two predicates in order to decide if the conditions of Exclusive-OR are fulfilled.

Rumelhart, Hinton and Williams (1986) show a three layer back propagation model which solves this problem. They used networks with one or two hidden units, as shown in figure 2.6, both of which managed to solve the problem with the observed weights as shown in the figure. They report that very occasionally the network failed to converge on the correct solution due to finding local minima instead of the global minimum.

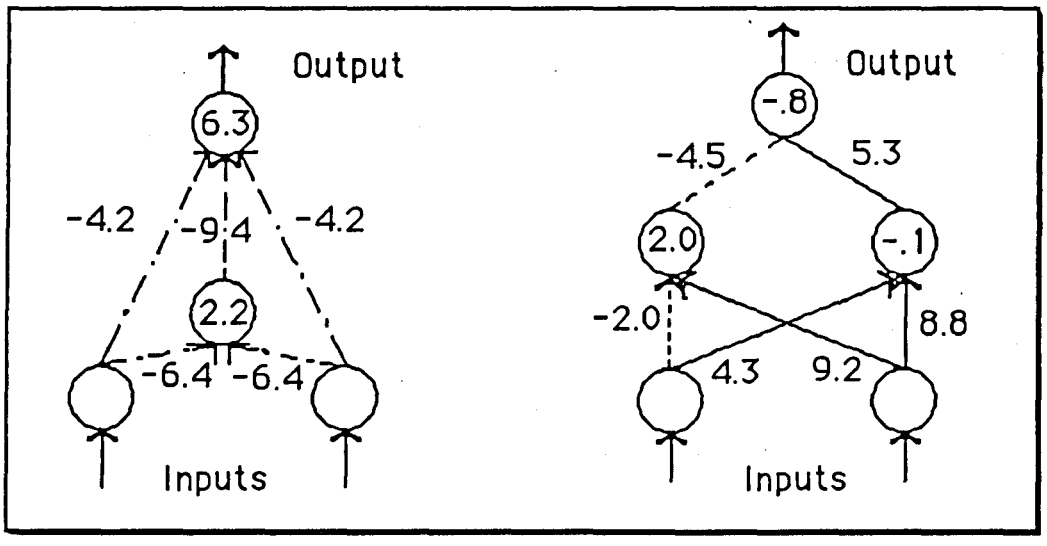

Figure 2.6 Rumelhart, Hinton and Williams (1986) solutions for the XOR problem, using one and two hidden units. Connection weights are written on arrows, negative weights are indicated by broken arrows, Unit biases are in the unit circles, if a bias is positive the unit will be on unless turned off by the weight of the connection.

Note that in the figure, the network with two hidden units gives an indeterminate output if the right input is on (ie. the input is 01), this turns on both of the hidden units, leading to the output unit having a net input of zero, and therefore the output will be 0.5,

so the configuration shown is not the optimum number of hidden units.

The Parity Problem

The parity problem is an extension of the XOR problem. Minsky and Paperts' analysis show that to determine parity a network is needed with at least one term which has the same size as the number of inputs to the network, which again was impossible to do with the perceptron. Rumelhart, Hinton and Williams solve this problem too; but not very elegantly. Their parity checking network requires, as in Minsky and Paperts analysis, the same number of hidden units as inputs to the network. The hidden units arrange themselves in such a way as to count the number of inputs (a hidden unit turns on, when another input unit comes on). The output unit gets the sum of the positive and negative weights from the hidden units, which cancel each other out when there are an even number of input units active, and produce a net activation when there are an odd number of active input units. (See Figure 2.7)

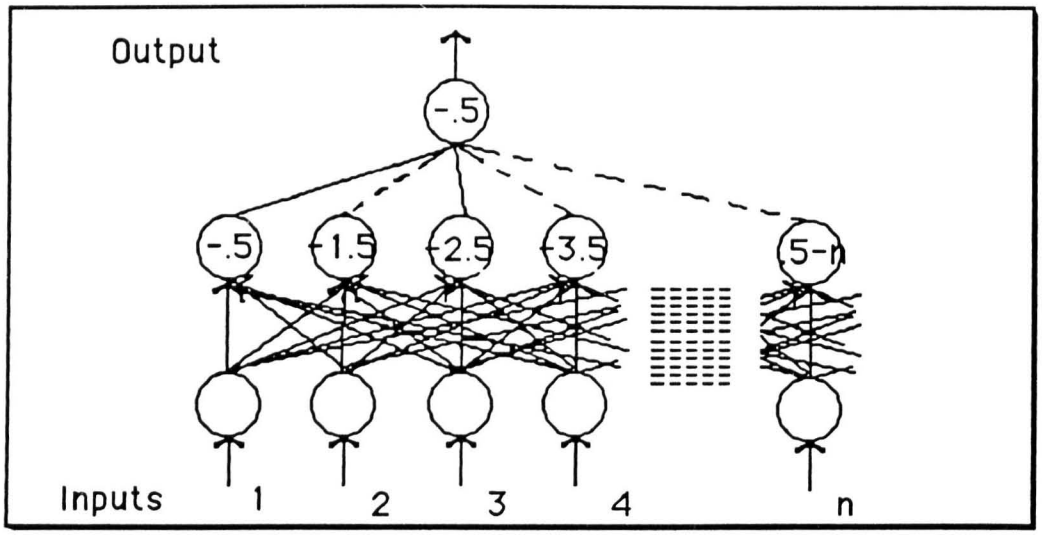

Figure 2.7 Rumelhart, Hinton and Williams (1986) solutions for the parity problem, using n input and hidden units. Connection weights are not shown, but negative weights are shown by broken arrows before the final layer. Unit biases are shown in the unit circles.

The T- C Problem

The T - C problem stems from the necessity of being able to recognise a shape independent of its position or orientation. This problem, as formulated in Minsky and Papert (Minsky & Papert, 1969), is composed of a pair of figures of five connected pixels, one in the shape of a 'T' and one in the shape of a 'C'. The difference in these figures lies in the position of only one of the pixels (it was not actually known as the T - C problem in 'Perceptrons', just as a pair of figures that could not be distinguished by a network of order 2).

Rumelhart, Hinton and Williams solution to this problem is derived from the configurability of a three layer system. They

adjusted the physical structure of a three layer network to produce hidden units with a limited set of inputs from the input layer. The inputs were localised into a 3x3 grid, or receptive field, while there was a single output unit. (see figure 2.8) In addition each receptive unit was constrained to change its weights in the same way as each of the other units, so that each field learned only the main figure and not partial figures.

They found that a series of templates developed inside the network, which discriminated between the two figures, such as a diagonal bar detector for detecting T's, and a 'compactness' detector for detecting C's.

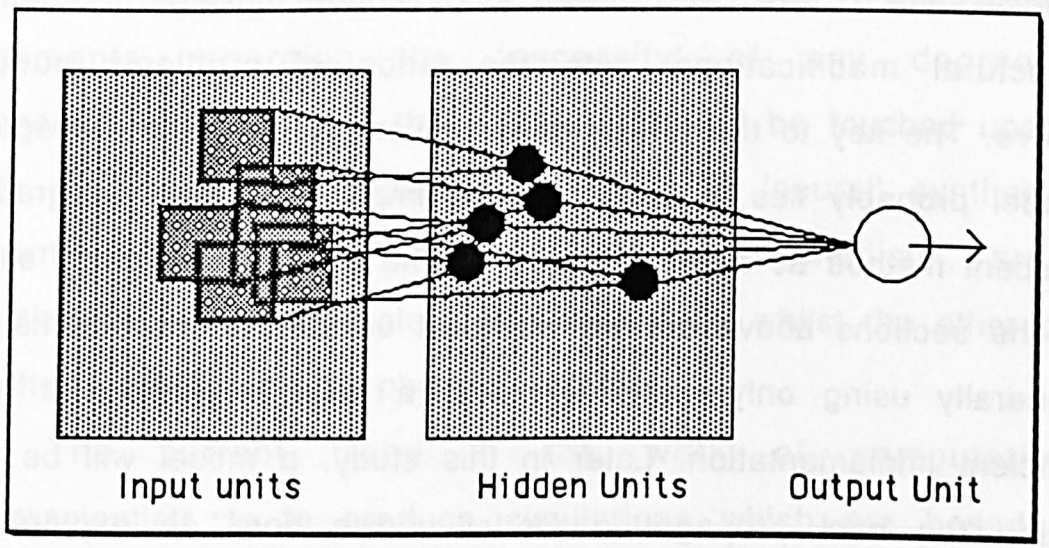

Figure 2.8 Rumelhart, Hinton and Williams (1986) solution to the T – C problem. Each of the hidden units connects to only a 3x3 array of input units. The output unit acts as a comparator.

Conclusion

We have seen that the back propagation model produces a fairly efficient family of pattern recognition systems. It has been shown that these types of model can solve many of the problems associated with neural network models in the past, and copes (but not always elegantly) even with the criticisms levelled in the Perceptrons treatment by Minsky and Papert.

The extensions referred to, in the later part of this chapter, will hopefully form the basis of more powerful and efficient models, which have been shown to deal more quickly, and without structural modifications, with the kinds of problems outlined above. The key to the further enhancement of the back propagation model probably lies in the continued improvement of the gradient descent method as mentioned above. The extensions already shown in the sections above are all examples of such enhancements, but generally using only one method at a time to produce a more efficient implementation. Later in this study, a model will be built using the back–propagation model shown here, to explore the addition of noise and graded extraneous activation under different conditions. These models will be compared to assess the efficiency of a method proposed in chapter 4.

3

Real Neural Networks and Computational Neuroscience

Most of the workers in the field of artificial neural networks will admit (sometimes grudgingly) that their field is at least loosely connected to the study of the behaviour of Real Neural Networks. The degree of this connectedness in actual simulations is often highly variable, and there are many philosophical arguments regarding the 'necessity' of any degree of connectedness. Some of these arguments will be touched upon in later discussions. One particular branch of 'neural' synthesists, known collectively as 'Computational Neuroscientists', are in practical terms, at one pole of this argument, whilst the other pole is often represented by physical scientists.

The current trend in the work of computational neuroscientists is to produce simulations which are based, as closely as possible, on the known behaviour of living nervous tissue. This stands on the traditional scientific method of reductionism followed by synthesis. In this manner, they hope to gain an insight into the actual operation of living neural systems, which, if studied in great enough detail, should yield at least a working model of a correctly functioning neural circuit, if not an understanding of the principles behind the operation of the circuit.

The aim of this chapter is to detail some of the foundations and behaviour of real nervous systems in a way that is amenable to

a quantitative simulation in computational terms. This exposition will in no sense be complete, as many of the fundamental principles of operation of even the simplest neural systems are still under investigation. Thus we must, for the moment, be content with a partial description of the known parameters of neural membranes, ion channels and neurotransmitter behaviour.

The second part of this chapter deals with the ways in which the knowledge of detailed behaviour of the actions of real nervous tissue have been used in low-level simulations of neural circuits by computational neuroscientists. Most of these simulations are still at the level of single cell models, while some deal with the interconnections in small-scale circuits in simpler life forms.

The 'Wetware'

This section forms an introduction to neural tissue from gross organisation to molecular level structure. There is little contextual relevance in describing the overall physical organisation of the entire nervous system, but some description of the immediately relevant (to Neural Network researchers) areas will be attempted. The organisation is taken from the human brain, since producing an intelligence on a par with the human system is the implicit goal of artificial intelligence. Later, obviously, some of the behavioural parameters referred to will be from experimental preparations of simpler life forms. These will be indicated where appropriate.

The human nervous system is an extremely complex network of many different varieties of neural elements. The basic element is often considered to be the neuron, which is a protoplasmic body surrounded by a lipid bi–layer membrane. There are approximately 10–50 Billion neurons in the human brain (depending on the information source) – with more located in the peripheral nervous system and ganglia. Neurons can be differentiated into several different varieties. The distribution of neuron types is often highly organised, depending on both regional and local factors. Some types of neurons are found only in specific areas, while others appear to be widely distributed. (Only one area will be studied in this section: The Cerebral cortex (assumed to be the seat of cognition))

The Cerebral cortex

The Cerebral cortex consists of the 'grey' matter of the cerebral hemispheres. It is only a few millimetres thick, but is heavily convoluted into well–recognised patterns, dipping into many fissures and bulging into gyri.

The cortex is thought to be separable into three phylogenetic regions. These are the Neocortex, the Mesocortex and the Allocortex. The Allocortex represents the oldest formations, making up the Hippocampus and olfactory regions of the cortex. The Hippocampus is a particularly interesting region which is receiving much attention in current research, as a major functional region in memory formation. This is significant on an evolutionary scale. The Mesocortex is represented by the Cingulate gyrus, which is a

structure overlying the Corpus Callosum in both hemispheres (see Figure 3.1) This structure has many connections with the Hippocampus and other areas of the limbic system. The Neocortex covers the major part of each cerebral hemisphere, where it is assumed to take part in a large variety of higher sensory and motor processing.

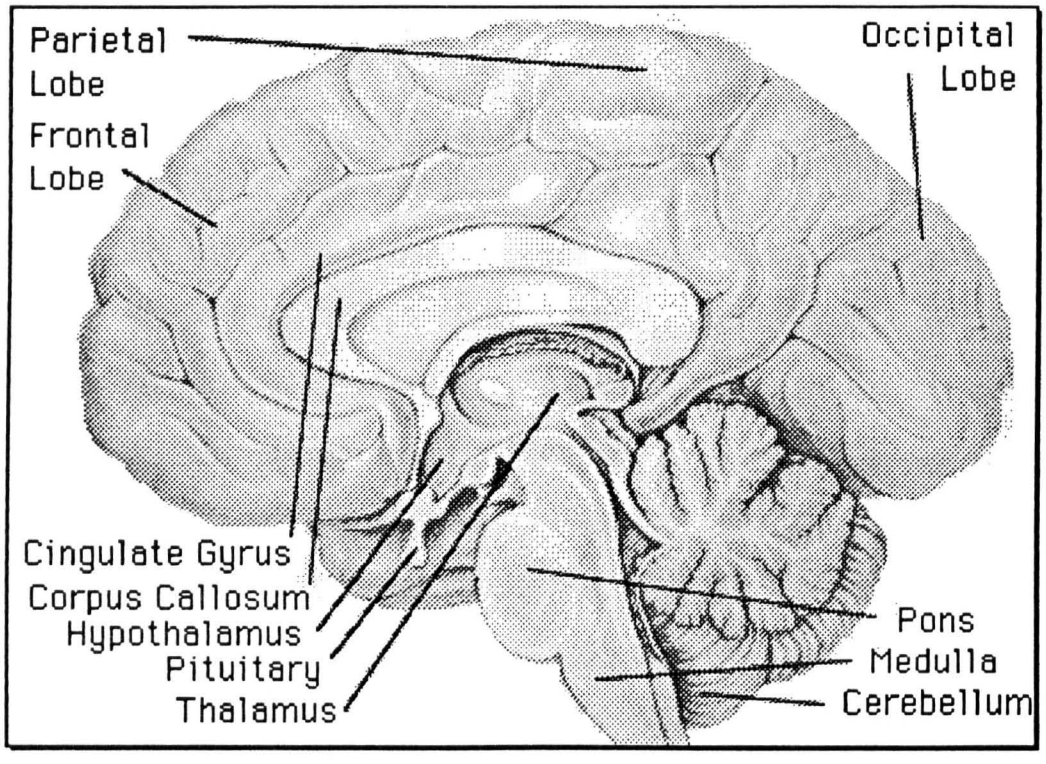

Figure 3.1. General view of hemisphere showing major structures.
(adapted from Kupfermann, 1985)

Each type of cortex has been studied extensively and shown to exhibit a stratified pattern of cell distribution. Allocortex and Mesocortex appear to show segmentation into 3 layers, whilst Neocortex seems to have a 6 layer structure. Other levels of organisation have also been postulated within the cortex. Recent research has also offered columnar and mini–columnar theories of neuronal organisation. These organisational structures will be

described after an exposition of the types of cell to be found in the cortex.

There are five basic varieties of cell types found within the cortex (see Figure 3.2):

Pyramidal cells are about 10–50μm in size and have a roughly conical shape characterised by dendrites at both the upper and lower extremities of the cone. Axons usually leave the cell body at the base of the cone, acting as association (within the same hemisphere) or commisural (across hemisphere) connections. These types of cell can be described as 'communicators' as their axons are generally those which connect different regions together, although many pyramidal neurons also connect locally.

Stellate cells are about 4–8μm in size, and as the name implies, are star–shaped, with extensively branched short dendrites projecting from almost any part of the cell surface. Their axons may project to the local area or other cortical layers in the vicinity.

Spindle cells are small elongated bipolar neurons, oriented vertically with short basal dendrites and longer apical dendrites. The axons of these cells usually project into the white matter underlying the cortical laminations. A variety of spindle cells, oriented horizontally are found in newly–born infants. These disappear in the early stages of post-natal life. These are known as Horizontal cells of Cajal.

Martinotti cells are small multipolar cells with short branching dendrites clustering around the cell body. The axon ascends vertically to more superficial layers of cortex, with horizontal branches being produced en route.

Glial cells are often forgotten in descriptions of cell types, but they do seem to play a major support role in the life of the axonic neurons. At the least, they support other cells in a matrix of their cell processes, possibly providing a pathway for metabolic processes. At the most, they may provide a pathway for slower forms of local signal transmission in the nervous system.

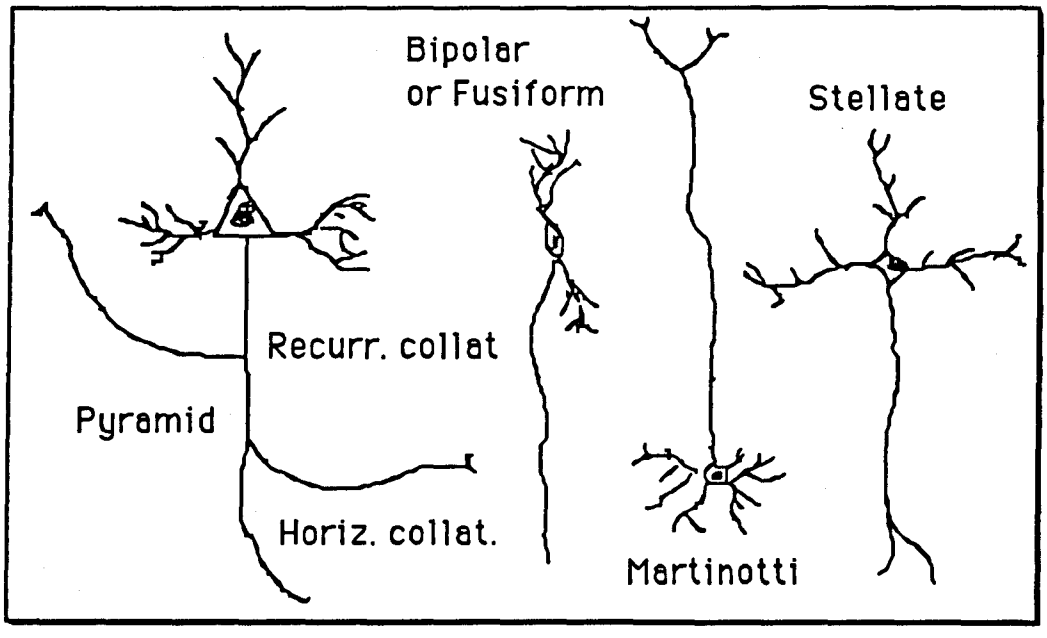

Figure 3.2. Cortical Cell types 1-4.

Cortical Laminae

The cortex has been stratified horizontally into various layers, depending on the distribution of cells and myelinated fibres. It is usually agreed that the Allocortex and Mesocortex appear to have 3 layers of cells. The lamination of the Allocortex is most easily demonstrated in the Hippocampus (Figure 3.3) The Hippocampus has three obvious layers based on cell distributions,

although other stratifications have been proposed. The three layers are made up of the molecular layer, pyramidal layer and the polymorphic layer.

The Molecular layer is seen as a communication layer, being made up of a dense collection of cell processes, interspersed with few neuroglia.

The pyramidal layer is composed mainly of pyramidal cells, the axons of which are thought to be the only ones to leave the Hippocampus. The cells in the pyramidal layer are more compact in the superior region than in the inferior region, although the cells in the inferior region are larger. The apical dendrites of this layer generally form in the molecular layer, where much branching takes place. The axons, too, branch off recurrent collaterals in the polymorphic layer. Pyramidal neurons are known as the principal cells of the Hippocampus.

The Polymorphic layer is composed of cells of many forms. Their axons generally remain inside the Hippocampus, connecting in a highly branched 'basket' to pyramidal cells of the pyramidal layer, where they appear to exhibit an inhibitory influence.

The Hippocampus is a highly connected area, which studies have shown to have a high activity rate and metabolism. Experimental work on the Hippocampus continues to associate this area with the formation of memory.

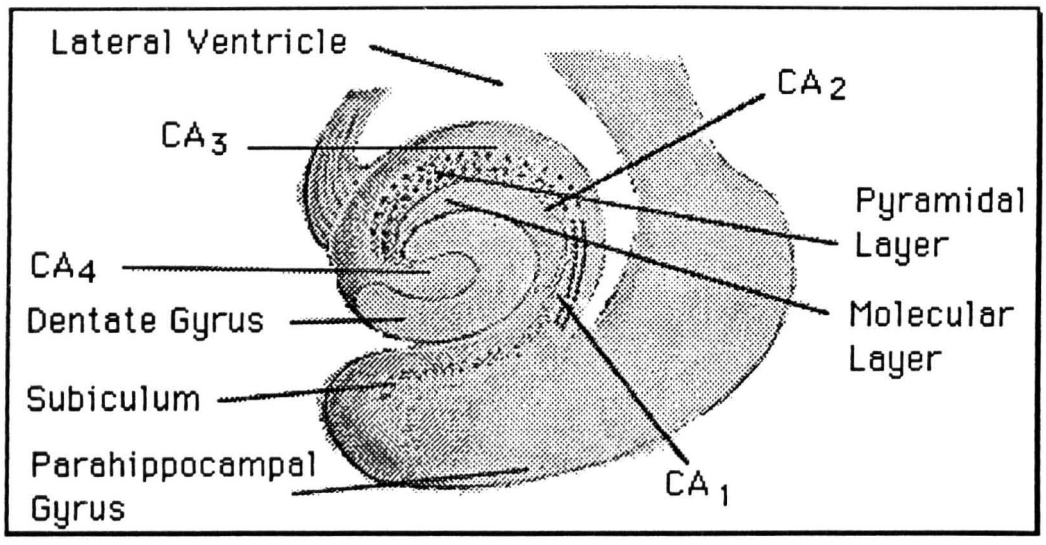

Figure 3.3. Hippocampus, Laminae and sections (adapted from Afifi & Bergman, 1986).

The layers of the neocortex vary from region to region. In the motor cortex region, for example, several layers appear to be missing. This is thought to be a result of specialisation in this area. In general, six layers are agreed to be present. (see Figure 3.4.) The top layer (layer 1) is a molecular layer, as in the Hippocampal description above, consisting of nerve fibres, with sparse glial cells. It is here that incoming axons terminate on the apical processes of cells in deeper layers. The second layer is known as the external granular layer (in Brodmann classification) which contains small pyramidal and stellate neurons. This is a dense region with incoming axons from other cortical regions. The third layer is known as the pyramidal layer, which contains mostly pyramidal neurons, with apical dendrites extending into the first layer. The axons of these cells connect to other cortical and sub–cortical areas. Layer four appears to be a local processing zone containing small stellate cells with cell processes which do not

leave the layer. Axons from other areas, however, do pass through this layer in both vertical and horizontal planes, allowing for some extraneous modulation. This layer is known as the internal granular layer. The fifth layer is known as the Ganglionic layer, having larger pyramidal and stellate neurons. This layer also contains some Martinotti cells. The dendrites from the principal cells pass to the upper layers, whilst their axons pass mainly to sub–cortical regions. Layer six consists of many different varieties and sizes of cell, hence, it is known as the Multiform layer. This layer appears to contain more Martinotti cells than other layers. The ramification of cell processes tends to follow the trend of size and type of the cell in their extent.

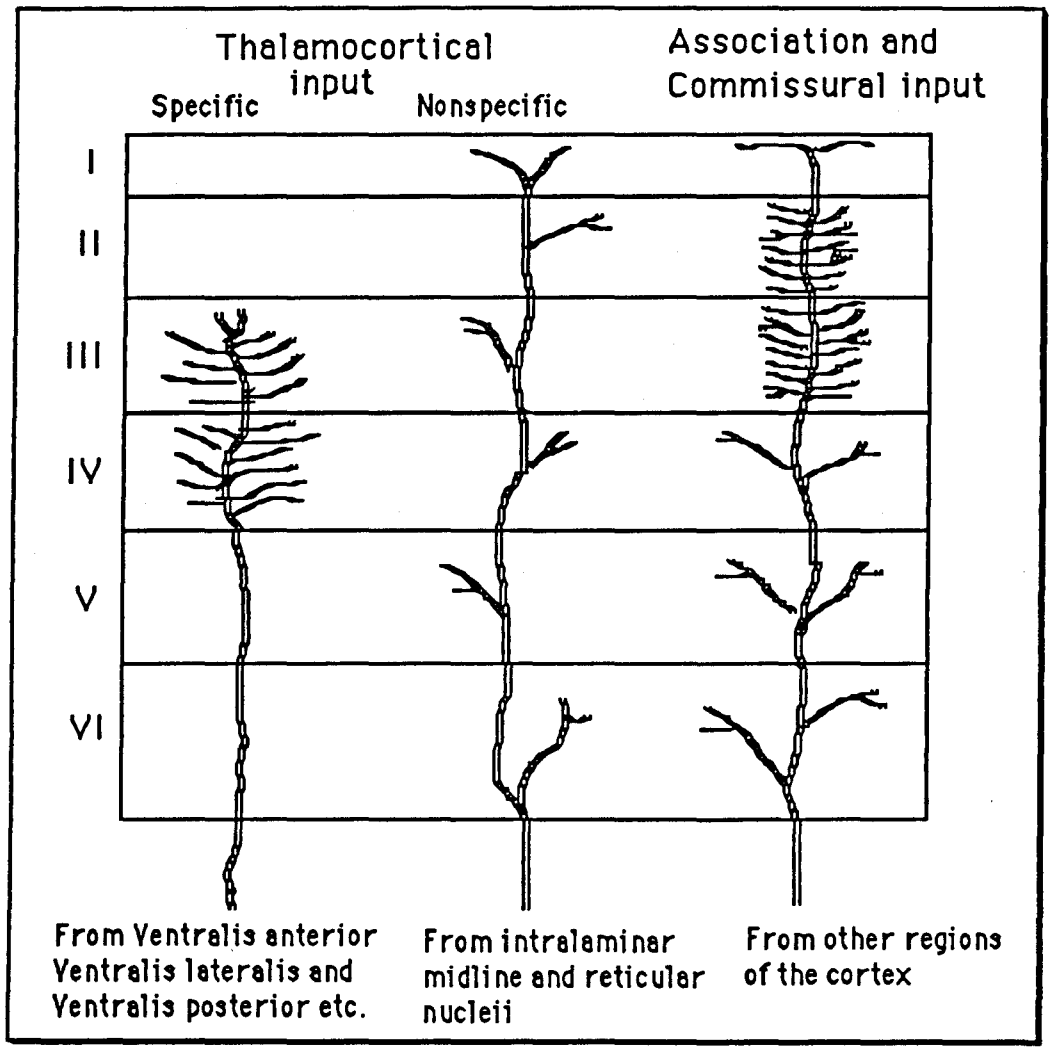

Figure 3.4 Layers of the Neocortex (adapted from Afifi & Bergman, 1986)

Cortical Columns

In addition to the horizontal stratification of the cerebral cortex, many researchers have postulated a vertical structure in the cortex, progressing towards theories of modular processing units within cortical columns. It is of great interest to the neuroscientific community to propose working functional circuits within the nervous system. Many researchers see the emergence of

a functional description of neuronal circuitry as the only progression towards an information theory of neural networks.

The cortical macro–column is assumed to have a diameter of approximately 300μm, which would contain about 4,000 neurons, of which half would be pyramidal cells. Over a surface of the cortex of about 2.5ft^2, this would correspond to around 3 million cortical modules. Each of these modules is said to inhibit surrounding modules (Mountcastle, 1978). The cortical mini–column is an additional structure imposed on the cortex under the macro–column, with a diameter of 110μm, containing just over 1,000 neurons (Peters A & Kara D, 1987). However the existence of either of these structural modules is doubtful (Swindale, 1990).

Neuron Microstructure

Synapses

Synapses are the connection units of the neuron. The term 'synapse' is often applied to the entire structure of the signal transference unit, although, in a more rigorous interpretation the 'synapse' applies only to the junction between two neural cell processes, and is, in fact, a gap of some description.

There are two major types of synapse structures. The least numerous of these are the gap junction or electrical synapse. The gap junction is a region of cytoplasmic continuity between two neurons. This is achieved by a reduction of the distance between

the two cells, and allowing bridging molecules in the cell membranes to line up on either side of the junction. These bridging molecules are known as connexons. Connexons are annular proteins with a central channel of about 2nm diameter, through which the cytoplasm of adjacent cells can be linked, allowing the transfer of small molecules. Simple ions pass easily through these pores, and thus electrical continuity is maintained. Gap junctions allow an almost simultaneous transfer of electrical activity between two neurons. This also means that gap junctions can allow bi–directional electrical propagation.

Chemical synapses are cytoplasmically non-contiguous. The structure of these junctions is variable, but conforms to the general principle of a narrowed extracellular space, with an outgrowth of one cell process towards the membrane of a second cell. The outgrowth is known as the pre–synaptic bouton, the target membrane is referred to as the post–synaptic membrane, and the gap in extracellular space between these two elements is referred to as the synaptic cleft (See Figure 3.5.) Electrical continuity is not achieved in this case. Signal transmission is accomplished by the secretion of chemical messengers from the pre–synaptic bouton diffusing across the synaptic cleft to target sites on the post–synaptic membrane. As the chemical messenger is usually only found in the pre–synaptic bouton, signal transmission is a one–way process. This diffusion method also involves a time delay. The target of the chemical messenger is an ionic channel in the post–synaptic membrane, which allows electrical activity to be induced in the target neuron.

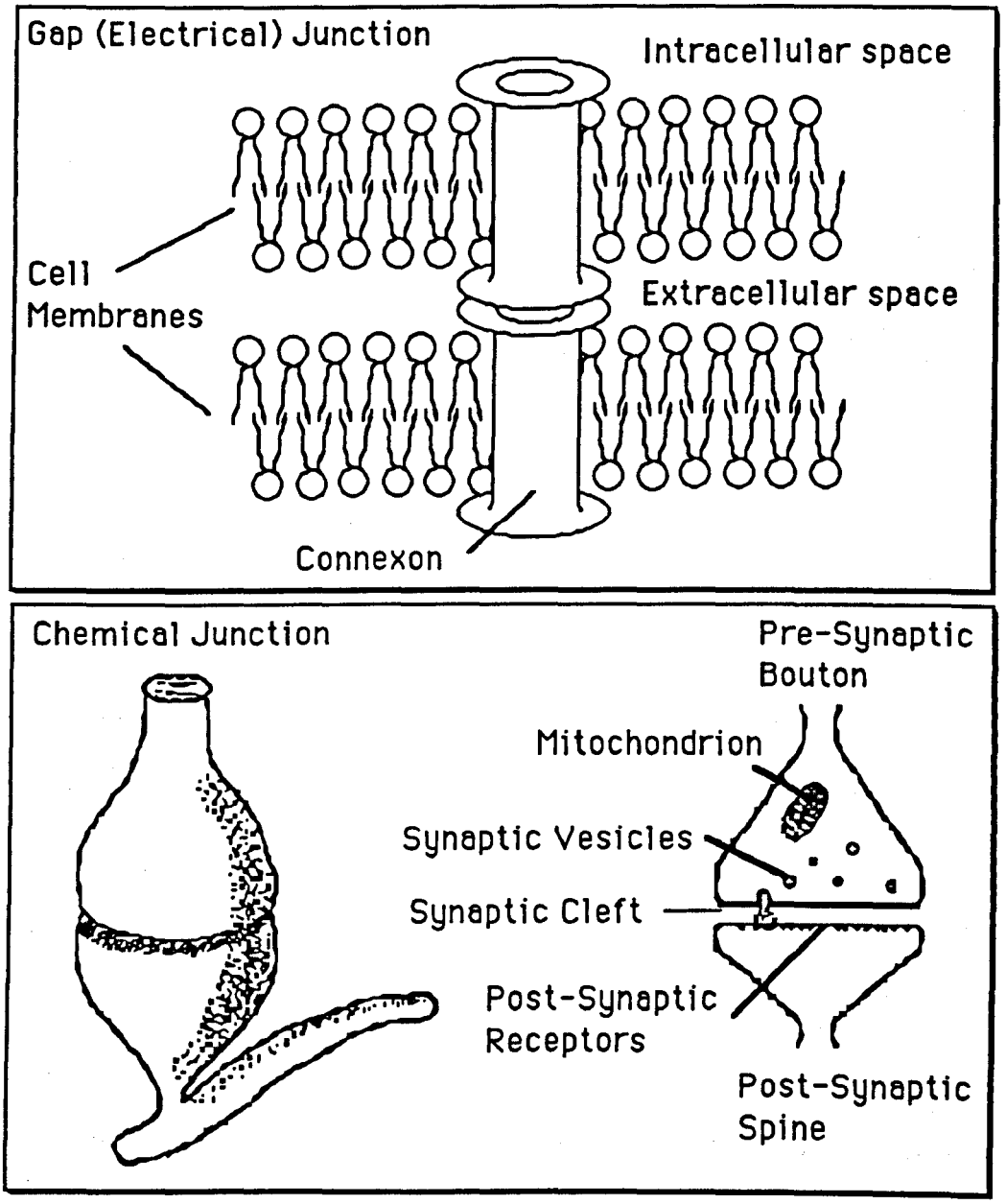

Figure 3.5. Gap junctions and Chemical synapses.

The synaptic transmitter chemicals are stored in vesicles in the pre-synaptic bouton. There are two major classes of transmitter, those which excite the target neuron and those which produce an inhibitory effect. Different amounts of transmitter may be released for different levels of pre-synaptic signal, and transmitters are released in quanta dictated by the nature of their storage in vesicles. Many different transmitters exist. Some of the more common transmitters are listed in Table 3.1 below.

Transmitter	Type
Acetylcholine (ACh)	Molecular
Norepinephrine	
Dopamine	
Serotonin (5–Hydroxytryptamine)	
Histamine	
Aspartate	Excitatory Amino acids
Glutamate	
N–Methyl–D–Aspartate	
Gamma–aminobutyric acid	Inhibitory Amino acids
Glycine	
Taurine	

Table 3.1 A subset of known neural transmitter substances.

The table shows that there are many neural transmitter substances involved in signal transmission within the nervous system. This is not an exhaustive list of substances known to be transmitters. Current research is continuously expanding the list of possible transmitters. In the past twenty years or so, research has been focussed on the existence of several neural peptides which appear to be neuro–active. The description and implications of these neuro-active peptides will be dealt with in the following chapters.

The character of neurotransmitters is determined by the type of receptor upon which it binds. Some neurotransmitters bind to a number of different receptor types and therefore have different characteristics. For example, Acetylcholine binds to two types of receptor, named 'Nicotinic' and 'Muscarinic' after the types of

chemicals which also excite activity in these particular receptors. Muscarinic receptors allow a much slower time course than the Nicotinic receptors, and the activity of the target cell is therefore dependent on the type of receptor at the active sites of the cell.

Ion Channels

All signalling activity in the nervous system is completely dependent on the presence of ion channels in the membrane of the neuron. Ion channels are often passive (non–gated) or are modulated by an exogenous substance which enables or disables the ion channel (gated). The electrical activity of a cell is governed by these ion channels. (Electrical activity of the neuron is dealt with below.) In essence, each ion channel allows the transmission of a small number of ionic species through the membrane of the cell, in most cases, only one ion type can be passed, due to the specificity of the ion channel.

The structure of generalised ion channels is shown in figure 3.6. The non–gated variety allows ions to pass according to the concentration gradient on either side of the cell membrane. The speed of ionic passage is determined by the size and transfer mechanism of the channel. Gated channels are more complex. Gated channels can be opened or closed by the presence of particular substances or electrical potentials. At a chemical synapse, the ion channels are opened by the excitatory transmitters (ligand gated) or by a voltage difference (voltage gated). The presence of a neurotransmitter molecule is thought to produce a conformational change in the ion channel 'gate' molecule, bought about by the

polarity of the molecule. This opens the gate and allows a particular type of ion to cross the membrane. Voltage gated channels are thought to function in a similar manner.

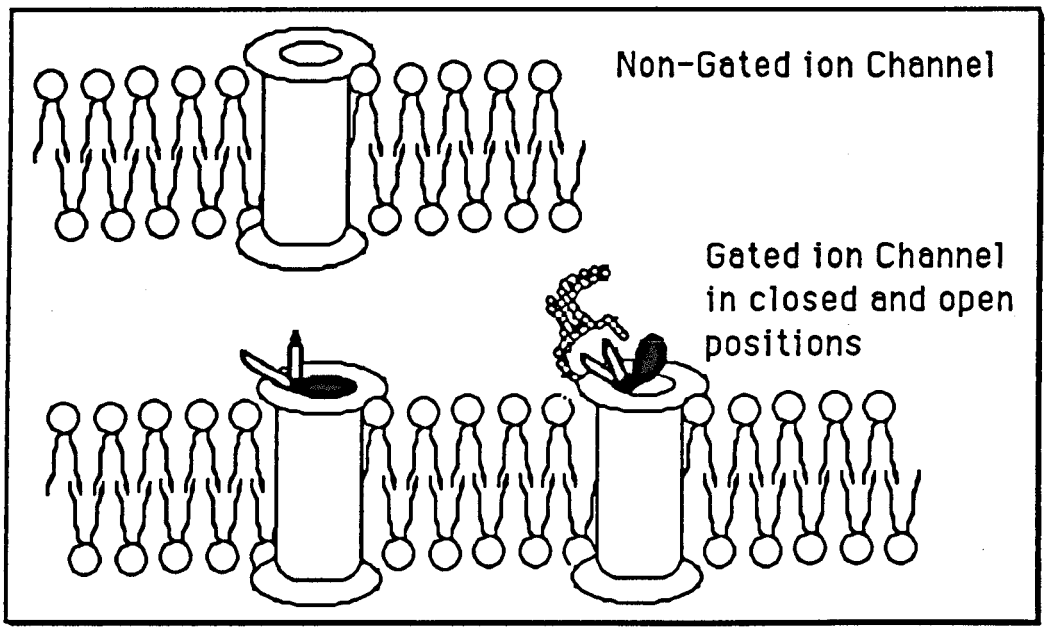

Figure 3.6. Generalised Ion Channels: Non–Gated and Gated.

Electrical characteristics of neurons.

Signal transmission in the nervous system is entirely reliant on electrical properties of the neuron. More specifically, the activity of the cell is dependent on the electrical properties of the ionic species in very close proximity to the cell membrane. The cell membrane represents a barrier to current flow which is breached by the ion channels referred to above. The ion channels allow current to flow (ie. ions) in specific amounts depending on the number of channels which are open at any time.

In the normal (resting) state of the cell membrane, a voltage difference can be measured across the membrane. The difference

for the 'average' neuron is approximately -50mV, this is due to the presence of the non–gated (or passive) ion channels. These are in a constantly open state, allowing ions to enter and leave the cell according to the concentration gradient present on either side. Additionally, an active 'ion–pump' exists within neural cells which expels Sodium ions and draws Potassium ions into the cell. Without going into an extensively detailed account, the resting membrane potential reflects the equilibrium state of the ionic species on either side, plus the action of the 'ion–pump'.

The membrane is vulnerable to changes in potential very quickly, due to the thinness of the charged region on either side of the membrane. This means that it takes very few ions to break the equilibrium imposed by the normal mechanisms of the cell membrane. Signal transmission is based on the fluctuations of charge produced around the membrane, by the action of concerted changes in the state of the ion channels, mediated by the voltage difference and the binding of neurotransmitter to ligand gated channels. Two antagonistic fluctuations of the membrane potential are responsible for the signalling properties of the cell. Depolarization produces the 'action potential', the driving force of neural circuitry. Hyperpolarization limits the cells' ability to produce action potentials.

Depolarization of the membrane is a cascade process, which occurs in the following sequence. Small changes in the voltage across the membrane, from the resting potential, cause voltage gated ion channels to open, allowing Sodium ions to flow into the cell. The change in voltage caused by the Sodium ion flow, opens more voltage gated ion channels, allowing more Sodium ions to flow into the cell. This positive feedback system quickly produces

a large depolarization of the membrane, which is transmitted along the membrane in the same way – Ion movement causes voltage difference along the membrane, opening voltage gated ion channels along the membrane. The cascade effect begins at about -45mV. The maximum likely extent of a depolarization is to a value of about +55mV. In most cells, there is a particular zone of the neuron which responds to changes in membrane voltage preferentially, known as the 'trigger zone', this zone can usually be located at the initial segment of the cell's axon. Recovery from the depolarization of the membrane involves a delayed outflow of Potassium ions through voltage gated channels, and a closing of the Sodium inflow voltage gated channels. This allows the membrane to recover its negative potential. Sodium and Potassium ions are then pumped in opposite directions to restore the chemical balance of the resting state, ready for the next depolarization.

Hyperpolarization of the membrane results in a decreased susceptibility of the membrane to depolarization. This is produced by allowing Chloride ions into the cell, or allowing more Potassium ions out of the cell. The movement of these ions produces an even more negative potential difference across the cell membrane, of up to -70mV. This voltage level means that a possible depolarizing influence has a greater voltage difference to overcome, and hence a larger amount of ions to move, before the voltage gated ion channels will open to begin producing the depolarization.

At the synapse, neurotransmitter is released as a result of electrical activity, propagated along the axon to the pre–synaptic bouton. Sodium and Potassium ions act in a similar way at the pre–synaptic bouton, as in the rest of the cell membrane, although an additional component is necessary for the release of

neurotransmitter substances. The depolarization of the pre-synaptic bouton results in the inflow of Calcium ions, into the interior of the bouton. Calcium ion influx has been shown to be necessary for the release of transmitter, and to determine the amount of transmitter released. The mechanism of action of the Calcium ions is thought to involve a secondary messenger inside the cell. The reliance on Calcium ions for the release of neurotransmitter, means that modulation of the signal can be achieved at the synapse itself, from outside influences such as inhibitory synapses on the axon, near the pre-synaptic bouton.

The Excitatory post-synaptic potential (EPSP) is the potential induced in the post-synaptic receptor of the target cell. The intensity of this potential is dependent on the degree of transmitter released by the pre-synaptic bouton. The length of the signal is also reliant on the speed at which the transmitter is removed from the synaptic cleft. This potential makes up the dendritic potential which adds to other potentials arriving at the trigger zone of the cell, which may induce an action potential in this cell.

Inhibitory Post-synaptic potentials (IPSP) can also be generated by inhibitory neurons synapsing onto a cell. Inhibitory neurons release transmitters which bind to inhibitory receptors on the target cell. These receptors often allow Chloride or Potassium ions to flow into the post-synaptic cell giving hyperpolarising effects, as outlined above. This membrane hyperpolarisation can be summated at trigger zones in a similar way to excitatory potentials

Other factors may modulate the strength of the signal before it reaches the trigger zone of the cell. In the case of post-synaptic

potentials in dendritic trees, the resistance of the electrical pathway to the trigger zone has a large part to play in attenuating the signal before it can be summated. Where a pathway is long, an enormous amount of potential can be lost on the way to the trigger zone. Cell membranes have a certain amount of capacitance that must be overcome for any charge movements to occur, as well as the ordinary resistance encountered in moving a current along a cell membrane. Neurons are thought to use this attenuation to their advantage. In the Hippocampus several different synapse zones are thought to occur on the major neurons of the region, characterised by particular positions on the dendritic tree, in a graded fashion from the cell body. In this case, a zone of the dendritic tree at a particular distance from the cell body receives synaptic contact from specific types of cells. This implies that the neuron is using the attenuation inherent in the dendritic pathway as a pre–wired means of selectively responding to different cell types (Gershon, Schwartz & Kandel, 1985).

Synaptic Plasticity

The heart of the adaptive process within the neural system lies in the long–term modulation of signals at the junctions of neurons. This is 'memory'. Without the ability to vary the modulation of signals, we would be incapable of learning anything new. Neural systems would be pre–wired behavioural patterns which adapt very little to the circumstances of their application in different situations.

It is thought that most adaptation, in the sense of signal modulation, occurs at the synaptic junction as a response to correlated activity conditions within the pre–synaptic and post–synaptic parts of the cell. The changes brought about by these correlations are simplified as changes in the 'strength' of a synaptic connection, implying that varying amounts of 'signal' are passed across a synapse as a result of the adaptive process. This is not strictly true, as certain neurons change the *character* of their signals as a result of synaptic 'strength' modulation, as well as an apparent change in magnitude of the signal passed. (Llinas, 1991)

The correlational idea of synaptic plasticity is attributed to Hebb (Hebb, 1949) as stated in the chapter 1. Correlation of activity in pre– and post–synaptic parts of cells produce (in time) a change in the parameters of the synapse, enabling positively correlated junctions and disabling negatively correlated synapses. The temporal aspect is thought to be composed of a short term memory system, which is then transferred into another form for longer term storage.

Short or long term?

Short term storage of information can be achieved in several different ways, spread across a number of synapses. Some candidates for storage can be studied directly, where simple synaptic effects give rise to short–term changes in the behaviour of the synapse. Other mechanisms must be theoretically proposed on the basis of expected behaviour in larger scale models of several neurons.

Simple synaptic effects have been studied extensively, and include Potentiation and Depression effects. These have both been found in the Hippocampus, which has been a focus for these studies. Long-Term Potentiation takes the form of an increased sensitivity to pre-synaptic input, after high frequency activation by the pre-synaptic terminal (Bliss & Lomo 1973). Sejnowski, Chattarji and Stanton (Sejnowski, Chattarji and Stanton, 1989) outline a further type of LTP known as associative LTP, taking place when different inputs to a neuron are stimulated in phase. They also introduced a form of Post-synaptic depression known as Associative LTD. This is an effect which occurs only when the dual excitation of pre- and post-synaptic cells are out of phase. This produces a post-synaptic depression in sensitivity to pre-synaptic activity.

LTP is usually associated with Hebbian plasticity, but it was found that Associative LTP would occur even when depolarisation is prevented in the post-synaptic cell. Sejnowski, Chattarji and Stanton (1989) regard this is a kind of pseudo-Hebbian process. The implication in this latter work is that the effect is produced in the pre–synaptic terminal. Another type of pseudo–Hebbian process is Post–tetanic Potentiation, also found in the Hippocampus. This is a result of a high frequency tetanus of the synapse and can last for many minutes - producing an elevation of the synaptic strength, without a change in post-synaptic sensitivity (Katz & Miledi, 1968)

Heterosynaptic Depression occurs where a weakly stimulated pathway converges on the same neuron as a path which occasionally carries strong stimulation. The weak pathway experiences a depression in transmittivity after high frequency stimulation of the strong pathway. The effect is known to last for a shorter duration than the effect of LTP (Levy & Seward, 1979)

On a higher level, short–term memory has been thought to be encoded in the activity of reverberatory circuits between neurons. This theory proposes that a circular pathway of activation is formed in a neural sub–system, which continuously cycles around the loop until the longer–term system has had time to encode the pattern of activity. This theory is based on a form of a 'rehearsal' idiom, and is not likely to be a major part of short–term memory.

Longer duration storage is thought to be a result of encoding the shorter term changes into a more permanent form. Accelerated Protein synthesis in stimulated cells has been taken as an indication that long–term memory is laid down in a protein base, but the mechanisms of this process are little understood. Research is continuing in this area, both in localising such changes and in mapping the functional changes in neuron structure.

Molecular mechanisms of mind

Each part of a cell's membrane can be thought of as a relatively complex computational circuit (Matsumoto, 1988). This is due to the molecular nature of the processes underlying the operation of the neuron at specific local areas. Many of the effects of neurotransmitter molecules are defined by their shape and dipole moments. At a molecular level, these properties are easily changed by the effects of enzymes and other binding molecules, which deform the shape of the target molecules (allostery), altering their electropolar properties at the same time. The effect of neurotransmitters on particular receptors is generally the production of a conformational change within the receptor

molecule, which alters its own binding with sub–membrane proteins, producing a change in the biochemistry of the internal environment of the cell.

Molecular 'computation' takes place both pre– and post–synaptically. As an example, in the process of pre–synaptic sensitization, a serotonergic axo–axonic connection to the pre–synaptic terminal produces a conformational change in the receptor protein. The receptor protein is normally bound to a protein known as a 'G–Protein', which is partially released by the neurotransmitter binding process. The G–protein activates a membrane bound adenylate–cyclase enzyme, which catalyses the formation of the secondary messenger Cyclic AMP, from Adenosine Tri–phosphate. The Cyclic AMP activates a protein kinase (a protein phosphorylation enzyme), which phosphorylates a component of a serotonin modulated Potassium channel. This reduces the recovery rate of the Potassium current, thereby extending the action potential, which allows a greater influx of Calcium to trigger exocytosis of neurotransmitter vesicles. (Kandel, 1985) Figure 3.7 shows a representation of this process. Similar processes account for the short–term modulation of most neurons. There is some evidence that the intracellular concentration of Calcium is important in synaptic plasticity, as high concentrations of Calcium are accrued by small sacs in dendritic spine heads. (Fifkova, Markham and Delay, 1983)

Longer term changes must be brought about in a similar manner to short–term changes, perhaps through a cell metabolism modulated by slow enzymatic changes based on secondary transmitters such as Cyclic AMP, or the presence or absence of particular ions.

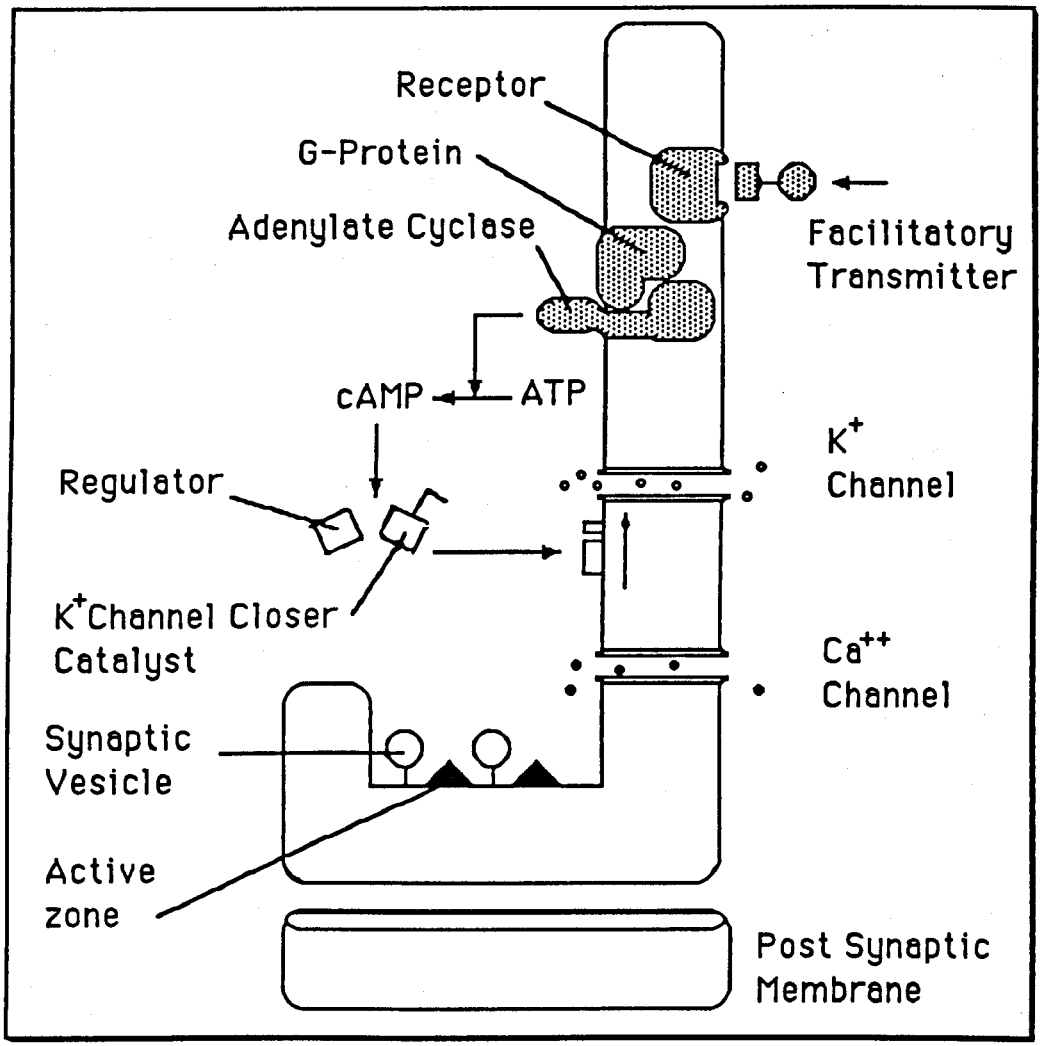

Figure 3.7. Molecular process of pre-synaptic sensitization (adapted from Kandel, 1985)

Summary

The nervous system has a gross organisation which is extremely complex. This complexity is manifested at every physical level from the ramifications and connections of the neurons within the cortical sheet down to the difference in

neurotransmitter chemicals used at the synapses, and the electrical behaviour of single ion channels.

This section has shown that the complexity of the nervous system has to be reduced to the molecular level to understand some of the processes which occur at higher levels. We still do not have a complete understanding of the processes of short– and long–term memory. The mechanisms of memory, particularly of long–term and the encoding of long–term from short–term memory, are vital components for the correct artificial modelling of any sort of working neural model. At the moment neural models must work in an environment of artificial long–term memory production, which has a dubious connection to the real processes merely due to the absence of data on the production of real long–term memory.

Computational Neuroscience

A note on History

Computational Neuroscience does not have much of a history. In essence, the field termed 'Computational Neuroscience' was born with the paper 'Computational Neuroscience' in 'Science' Volume 241 pp. 1299–1306 of September 1988. (Sejnowski, Koch & Churchland, 1988). Work had been done in this field under various guises for many years before this paper. One of the first notable contributions to functional analysis of neural system was the Hodgkin and Huxley characterisation of electrical signal propagation in the squid axon in 1952 (Hodgkin & Huxley, 1952); Since then much work has been done on the functional characteristics of neural tissue.

Computational Neuroscience is based on the preceding neuroscientific research, but its goals are to

> "...explain how electrical and chemical signals are used in the brain to represent and process information." (Sejnowski, Koch & Churchland, 1988).

In order to reach this goal, Computational Neuroscientists expect to produce detailed models of neural function, based on known neuroscientific data, together with some extrapolations from other neural network fields (called simplifying brain models –

referring to connectionism, parallel distributed processing, etc). It is intended that many different models will be built on different levels between the simplifying brain models and real neural networks, each designed to test possible mechanisms of action inside the natural nervous system.

In the major sense, Computational Neuroscience is an 'information processing' field, even though it takes advantage of biological data to produce models in which to investigate this information processing. The field deals with the physics of living systems generally bounded by the nervous tissue of living organisms. Philosophically, the core interest of computational neuroscience is in the way in which the human brain processes information. Ultimately, the unspoken aim of the field is to understand the processes of higher cognition, such as motivation and abstract thought, as patterns of base level activity within a nervous system.

Electrical equivalence of the neuron

Computational Neuroscience deals with the modelling of neural systems in as concise a manner as possible. This requires the use of mathematical models, often culled from other fields, which describe the behaviour of low–level neural elements as closely as possible. Much of the characterisation of neural tissue is reduced to electrical properties and equivalent electrical circuits, which allows an easier modelling route to computational studies. This is the method employed in this section to describe the properties of neural tissue.

The electrical properties of neural axons and dendrites can be modelled using a technique developed in the mid–nineteenth century by Lord Kelvin to model the characteristics of transatlantic telegraph cables. This 'Cable theory' is now the basis of calculations performed to measure the conduction characteristics of axons and dendrites. This is possible because these cell processes are tubes of cytoplasm assumed to have uniform electrical properties. The cytoplasm is surrounded by a membrane, which generally has a higher electrical resistance than the cytoplasmic interior and the extracellular environment. This sort of structure is known as a 'core conductor', which allows current to flow along the cytoplasm in a direction dictated by the boundaries of the membrane.

Cable theory assumes that a conductor has uniform properties, such as diameter, resistance and capacitance. This leads to inaccuracies in the application of cable theory to neural processes, as neural processes are usually not uniform in their properties. These inaccuracies must be tolerated, as exact models of a neural process would have to take all variations in geometry and electrical properties into account. This would be too computationally expensive to model, and would also lead to non-reducible properties. To reduce these inaccuracies, neural cell processes are modelled as short stretches of 'ideal' cables, attaching a set of uniform properties to each 'compartment' of the cable. This is shown in Figure 3.8. In this way, a cell model can be built with realistic overall properties from a number of combined cable 'compartments', each with a different set of electrical properties. Models built using this method are known as

'compartmental models'. These were introduced by Rall (Rall, 1964).

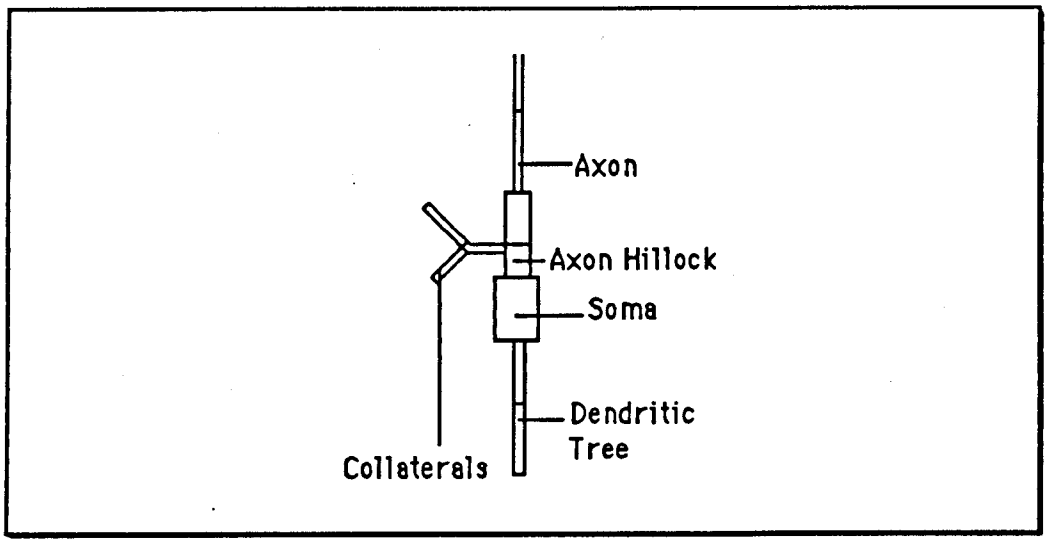

Figure 3.8. Compartmental model of a neuron.

Compartmental modelling is a useful tool. It allows the neuron to be broken down into an arbitrary number of compartments, depending on the level of detail required. Some studies use one compartment for an entire dendritic branch, while others model each dendritic component separately. The modelling of entire branches in one compartment is justified if the aim of modelling is to test the operation of the entire neuron, or a network of neurons. Modelling each dendritic branch as a separate compartment, or even a very short part of a dendrite as a compartment, is useful for comparing a model to a real neuron, but the amount of computational power involved precludes such a detailed model's use in network analysis.

Compartments are modelled on an equivalent electrical circuit based on the core conduction path and the membrane of the cell process (Figure 3.9). The equivalent circuit shows the core

conduction path as equivalent to a resistance along the cytoplasm, and the membrane as a resistance in series with a battery (representing the resting potential of the membrane) with a capacitance in parallel to the membrane resistance. The extracellular space is equated to a 'sink' or ground state, which is isopotential.

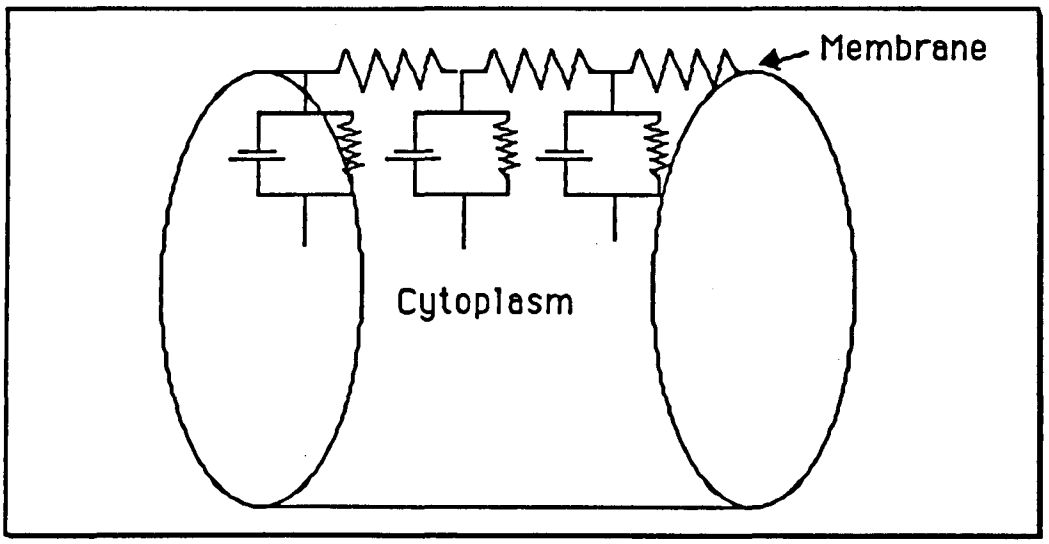

Figure 3.9. Equivalent circuit of a cell process. (adapted from W. Rall, 1989)

The current flow through each compartment can be calculated by differential equations based on the equivalent circuit. The gradient of the current through one compartment is calculated as:

$$\frac{\delta V_i}{\delta x} = -i_i r_i \qquad \text{(equation 3.1)}$$

(where i_i is the core current and r_i is the intracellular resistance, V_i is intracellular voltage and x is distance along the cell process)

because the membrane is a constituent of the compartment the current flow across the membrane must also be taken into account. Adding the necessary current inflow and outflow across the membrane component (without modelling the membrane) produces:

$$\left(\frac{r_m}{r_i}\right)\left(\frac{\delta^2 V}{\delta x^2}\right) = i_m r_m \qquad \text{(equation 3.2)}$$

(where r_m is the resistance of the membrane, r_i, is the resistance of the intracellular space, i_m is the membrane current and the differential part is the rate of the gradient change of Voltage over distance along the cell process)

the membrane properties include the membrane capacitance as a conjugate with the resistance as a time constant (τ_m) giving:

$$i_m r_m = \tau_m \frac{\delta V}{\delta t} + V \qquad \text{(equation 3.3)}$$

(where r_m is the resistance of the membrane, i_m is the membrane current, τ_m is the time constant of the membrane and the differential part is the rate of change of Voltage over time)

Before amalgamating the previous equations into a complete definition of the compartmental current activity, we must define a length constant, which defines the relationship between the membrane resistance and the intracellular resistance over distance:

$$\lambda = \sqrt{\frac{r_m}{r_i}}$$ (equation 3.4)

then, the final equation (the cable equation) for a compartment is:

$$\lambda^2 \left(\frac{\delta^2 V}{\delta x^2}\right) - V - \tau_m \left(\frac{\delta V}{\delta t}\right) = 0$$ (equation 3.5)

(Rall, 1989)

This is a partial differential equation which reduces to differing ordinary differential equation under direct and alternating current conditions, allowing the calculation of input resistance, voltage attenuation and AC impedance.

Patches of membrane which include receptor channels are modelled in a similar fashion, but include active gated resistances and batteries for the receptor (syn) and action potential (act) components. (Figure 3.10) The membrane current in this instance is specified by:

$$i_m = c_m \left(\frac{\delta V}{\delta t}\right) + g_{leak}(V - E_{leak}) + g_{syn}(V - E_{syn}) + g_{act}(V - E_{act})$$ (equation 3.6)

(where g refers to conductance, V is voltage and E is the Battery (Nernst) potential, for each of the three components; c_m is the membrane capacitance and i_m is the membrane current)

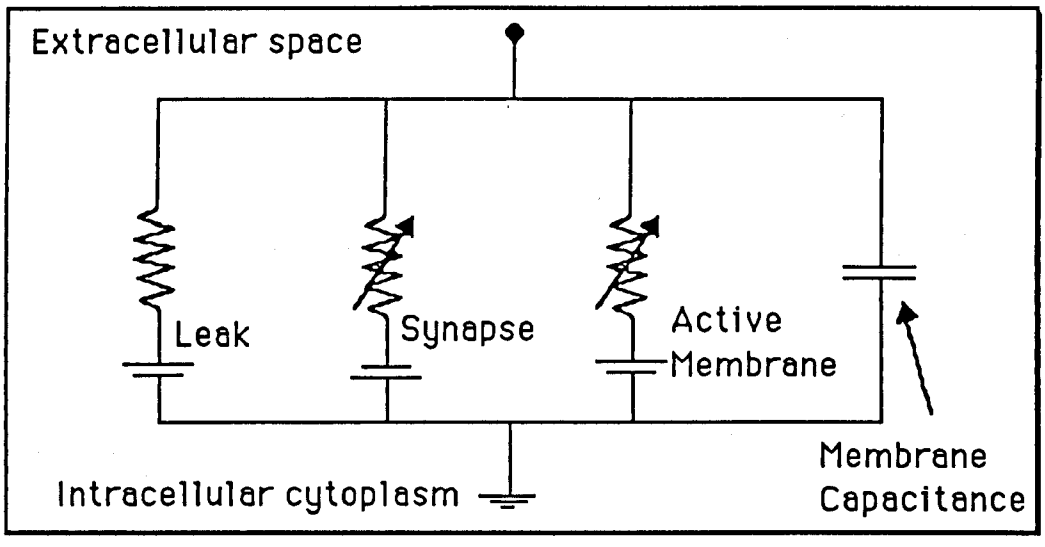

Figure 3.10 Equivalent circuit for a patch of active membrane. (Adapted from Segev, Fleshman & Burke, 1989)

Ionic channels conductance in a membrane, can be modelled as a population by taking the maximal conductance for an ionic species and modulating this by known activating and inactivating parameters, such as Voltage, ionic concentration and time, all three of which affect the state of an ion channel. Yamada Koch and Adams (Yamada Koch and Adams, 1989) model seven ionic currents of a Bullfrog ganglion cell separately and then integrate each of these into a model of neuron activation. The models apparently give an accurate prediction as to the actual behaviour of the neuron under patch–clamp conditions. Their model for Voltage change in the soma of the cell is given as:

$$C_N\left(\frac{dV}{dt}\right) + I_{input} + I_{Na} + I_{Ca} + I_K + I_M + I_A + I_C + I_{AHP} + I_{leak} + I_{syn} = 0$$

(equation 3.7)

(where C_N is the Capacitance, V is voltage, t is time, I_{input} refers to the input current injection, I_{Na} is the inward Sodium current, I_{Ca} is the inward Calcium current, I_K is the net outward Potassium current, I_M is the slow muscarinic Potassium current, I_A is the hyperpolarisation activated current, I_C is a Calcium sensitive outward current, as is I_{AHP}, which is a slower AfterHyperPolarisation Potassium current. I_{leak} is the leakage current through the membrane, and I_{syn} is the synaptic input current.)

Operational equivalents of neural processes

The previous section gives the electrical properties of the neural computation substrate, from which we can extrapolate to the actual computations performed by neural tissue. Most of these extrapolations are based on actual physical observations of the neural tissue under experimental conditions. In information processing terms, only a subset of neural operations is required to produce a usable digital computer on a par with current computer technology, so that a proportion of neural operations offer an additional processing layer which adds to the complexity of neural operations.

Operator: Action potential production
Input: Graded membrane voltage change
Output: single activation spike

The action potential of a neural impulse is an example of an all–or–nothing action. The input to the action potential is a graded build up of voltage leading to a threshold event, generating the action potential. The action potential functions as an analog–to–digital conversion function, as well as an AND gate, depending on the relative strengths of the incoming multiple inputs. It can act as an OR gate, if each incoming input is strong enough to overcome the threshold function. The action potential generator is also a variable gated device which is dependent on the previous activity of the generator. It can be thought of as a variable trigger switch with automatic biasing.

Operator: Repetitive spiking activity
Input: Graded membrane voltage change or current injection
Output: multiple activation spikes

Repetitive spiking occurs in some neurons as a result of prolonged synaptic input or current injection. There is often a linear relationship between the incoming current and the frequency of firing (Schwindt & Crill, 1982). which implies that an analog input current is being modulated into a frequency modulated domain. In this case a change is being made to the method of information representation, allowing for the possibility of

multiple channels of information in one physical channel within the nervous system.

Operator: Impulse Conduction
Input: Action potential
Output: single Action potential (variable attenuation)

Conduction of the action potential is a variable process, depending on the parameters of the axon itself, the action potential source and the frequency of action potentials. If the diameter of an axon is small, the impedance generated by a high spiking frequency may attenuate the signal drastically. In branching axons, these characteristics may allow signals to pass branching points without propagating the signal into one of the branches. This produces a filtering effect, propagating signals into the branch only when the impedance allows, possibly at a different ratio to the incoming activation. In physical terms, the fan out of an axon allows a wider distribution of signal transmission – the opposite of the fan–in implied by action potential production.

Operator: Chemical synapses
Input: pre–synaptic voltage
Output: post–synaptic voltage (non–linear)

The chemical synapse of a spiking neuron produces a non–linear output which appears very similar to that of a transistor device, sigmoidal, with an approximately linear section in the

center. (Katz & Miledi, 1967) This means that small variations in input voltage can act as a transistor, amplifying variations in output voltage if the input voltage is in the linear range. The gain of this amplifier is not necessarily above unity. (Martin & Ringham, 1975) The chemical synapse is defined as a non–reciprocal two–port in informational terms. Communication is one–way only, although reciprocal synapses have been found (Shepherd, 1979) The properties of the chemical synapse imply that it can act as either an analog amplifier, a digital switch or a digital to analog converter.

Operator: Electrical synapses (Gap junctions)
Input: pre–synaptic voltage
Output: post–synaptic voltage

Electrical synapses are a form of direct coupling between neural cells. The cell cytoplasm is almost continuous through a pore which connects the membranes of two cells. This pathway presents a low resistance to signals originating in a source cell, although the input resistance to a specific cell may be higher than its neighbour, giving a directionality to the junction. The transmission properties of these junctions show similarities to those of chemical synapses, depending on the electrical parameters of the cells on either side of the junction. The largest operational difference between electrical and chemical synapses, is that there is a lesser delay in signal transmission in electrical synapses. Gap junctions can behave in a similar manner to diodes, rectifying the

incoming signal through half of a sine cycle. (Furshpan & Potter, 1959)

Operator: Synaptic interaction
Input: Multiple pre–synaptic voltages
Output: Modulated post–synaptic voltage

The action of multiple synaptic inputs along a dendritic branch can produce many different effects. In some cases, additive excitatory inputs will produce a summated dendritic current. In other cases excitatory synaptic activity, coupled with inhibitory synaptic activity along a length of dendrite will produce complex effects, depending on the relative positions of the synapses and the timing of the synaptic activity. In a well–studied interaction, an excitatory pre–synaptic signal can be nullified by an inhibitory pre–synaptic signal on the dendrite at a more proximal position to the soma. (Koch et al, 1983) This is known as silent inhibition when the inhibitory synaptic reversal voltage is near 0mv. This produces no change in the dendritic voltage on its own, but will veto excitatory inputs on more distal parts of the dendrite. Hyperpolarising inhibitory inputs exert a different effect. These are summated with excitatory inputs to form the general excitation level of the cell. Operationally, the most important function appears to be the 'veto' or silent inhibition effect obtained by the inhibitory (0mv reversal potential) synapse on proximal parts of the dendrite. This forms an AND–NOT logical circuit which is important for direction selectivity in the visual

system (Hubel & Wiesel, 1959). Other forms of synaptic interaction operate as a form of analog summation.

Operator: Dendritic Spines
Input: post–synaptic voltage
Output: Modulated post–synaptic voltage

The post–synaptic output of synapses on dendritic spines is dependent on the properties of the spine. In particular, diameter and length of the spine are of importance, due to the differential impedance of the spine to input signals and the electrical saturation of the spine. As the spine neck is generally smaller than the dendrite and the spine membrane is assumed to be passive, then a general attenuation is expected to occur between the spine head and the dendrite. Where the membrane is active, the spine can act as an amplifier, if the number of active channels on the membrane is sufficient to produce a greater effect. It has been postulated that synapse weights can be altered by the change in length of the spine brought about by the contraction of filamentous proteins found in the spine neck (Actin, Fodrin, Tubulin. See (Crick, 1982)). This is assumed to occur in response to the concentration of Calcium in the spine head.

Operator: Quasi–active membranes
Input: Axonal current
Output: Modulated axonal current

Quasi–active membranes are membranes which display a variable output to differing inputs. In particular, some membranes may act as filters by virtue of their resonant frequency properties. Such membranes usually have a large proportion of secondary activation ion channels, such as voltage, time or ionic species concentration activated channels. These exert their effects by showing a peak conductance at a certain time after the onset of stimulation, these are then inactivated, and after a refractory delay, are ready to be activated again. Different membranes show different resonant frequency tuning and are likely to be used in sound transduction, but can be used in other filtering situations (Koch, 1984).

Operator: Transmitter regulation of voltage–dependent channels
Input: Presence of neurotransmitter and voltage–regulated ion channels
Output: Modulated ionic channel current

Neurotransmitter regulated ion channels have been found recently in many cells (Siegelbaum & Tsien, 1983). The modulation of the ion channel is a slow process, thought to take place as a result of secondary messenger action. The incoming neurotransmitter is bound to the membrane where it activates an

intracellular transmitter such as cyclic AMP, which determines the phosphorylation of ion channels. This modulation leaves the membrane in a heightened state of excitability for a long duration. This mechanism is similar to a variable biasing mechanism on a transistor, allowing control of the gain of the component.

Most of these operational functions are summarised in the table below, including some examples of the computation which may be performed by the particular structures under consideration.

Biophysical Mechanism	Neural Operation	Example of Computation
Action potential Initiation	Analog OR/AND one-bit analog-to-digital	
Repetitive spiking activity	Current-to-frequency transducer	
Action potential conduction	Impulse transmission	Long-distance communication in axons
Conduction failure at axonal branch points	Temporal/spatial filtering of impulses	Opener muscle in Crayfish
Chemically mediated synaptic conduction	Nonreciprocal two-port "negative" resistance. Sigmoid "threshold".	
Electrically mediated synaptic transduction	Reciprocal one-port resistance	Coupling of rod photoreceptors to enhance detection of signals
Distributed excitatory synapses in dendritic tree	Linear addition	α, β cat retinal ganglion cells, Bipolar cells
Interaction between excitatory and (silent) inhibitory conductance inputs	Analog AND-NOT, veto operation	Directional-selective retinal ganglion cells. Disparity-selective cortical cells.
Excitatory synapse on dendritic spine with Calcium channels	Postsynaptic modification in functional connectivity	Short- and Long-term information storage
Excitatory and Inhibitory synapses on dendritic spine	Local AND-NOT "presynaptic inhibition"	Enabling/disabling retinal input to geniculate X-cells
Quasi-active membranes	Electrical resonant filter analog. Differentiation delay	Hair cells in lower vertebrates
Transmitter regulation if voltage-dependent channels (M-current inhibition)	Gain control	Midbrain sites controlling gain of retino-geniculate transmission
Calcium sensitivity of cAMP-dependent phosphorylation of Potassium channel protein	Functional connectivity	Adaptation and storage of information in *Aplysia*
Long distance action of neurotransmitter	Modulating and routing transmission of information	

Table 3.2 Some Neuronal Operations and the Underlying Biophysical Mechanisms (From Koch and Poggio, 1987)

Computational Models

With these characterisations of the neuronal substrate shown above, it is possible to model a single neuron in very great detail, as well as larger networks of neurons. Low level models of single neurons are useful for testing theories of single neuron behaviour. Such models are at a stage where a computer model can accurately represent the properties of a neuron under different input conditions. Much work has been done on some of the properties of smaller networks, which underlie the greater processing power of a nervous system, such as the oscillatory networks and Central Pattern Generators (CPG's). In simpler animals, these CPG's operate specific parts of the organism, allowing models to test actual physical operations.

Many methods exist for the simulation of neural circuits. These methods vary from the cable theoretical systems given above to simulation by specific electrical components. In the component oriented models, as usual, some of the lower level details are omitted in order to present a model which exhibits behaviour in reasonable computational time. In most cases, different models are tested to assess their viability for the specific circuit under investigation, and modifications are made to make the model type fit the behaviour of the real neural circuit. This means that each new simulation will be a hybrid of various modelling types, making studies of previously modelled systems only partially relevant to new studies. These facts make it difficult to report the results of modelling experiments in a

shorter form than was given in the original publications, so this section serves merely to indicate the areas in which work has been done, and give references for further reading.

Single Neuron model

Stratford et al (Stratford et al, 1989) produced a model of a cortical pyramidal neuron, from layer 5 of the visual cortex. This was based on their own analysis of the cell type involved. They evaluated three general modelling methods for these cells, including equivalent cylinder analysis, segmental cable models and compartmental modelling. The equivalent cylinder method (Rall, 1977) attempts to collapse dendritic branches into a single cylinder The pyramidal neuron is question did not meet the geometrical criteria for equivalent cylinder models, so this method could not be used. The segmental cable model (Koch & Poggio, 1985) was found to be too mathematically complex for a reasonable computational solution, so the compartmental modelling method was used.

The compartmental modelling method involved the assumption of isopotential compartments over the neuron and the simplification of the structures involved. Here, Stratford et al. attempted a simplification of the apical and basal dendrite structure. They used two methods; Dendritic profiles and dendrite cartoons. Dendritic profiles are a one-dimensional summative measure of a single equivalent dendrite, if all dendrites at a specific distance were collapsed into one dendrite. The measure used is the diameter of the equivalent dendrite and calculations

are based on this measure and the distance from the soma. The Cartoon method is a two dimensional compartmental model, which preserves the characteristics of distance from the dendrite axis, as well as distance from the soma. Figure 3.11 is adapted from the original figure from Stratford et al. comparing these two methods.

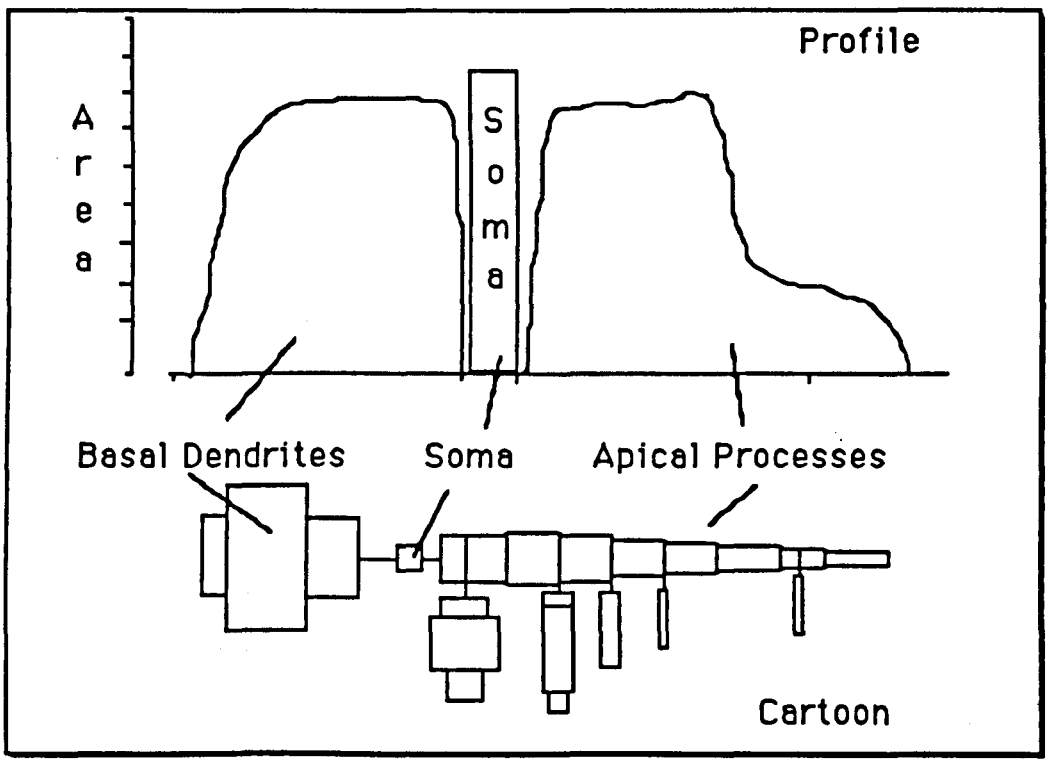

Figure 3.11 Comparison of the profile and Cartoon representations of dendritic geometry. (Adapted from Stratford et al, 1989)

Running the computational models of the neuron showed that the cartoon method closely mimics the output characteristics of the real neuron, when the model parameters were set to biophysically plausible values.

Small Network models

Selverston and Mazzoni (Selverston & Mazzoni, 1989) modelled the Central Pattern Generator of the pyloric sub–network of the lobster stomatogastric ganglion. This is a three neuron oscillating network operating within the ganglion, which generates an oscillatory output when isolated (see Figure 3.12). This oscillator controls some of the events in the lobster stomach system, and shows a particular phase relationship, necessary to the correct treatment of food in the stomach region. Selverston and Mazzoni use a coarse grain electrical model to simulate the physiological mechanism. Their modelling system is derived from the work of Hopfield and Tank, (Hopfield & Tank, 1986) which uses operational amplifiers to simulate the action of the neuron. In order to approximate biophysical detail, gap junction components were added, simulated by specific resistances, and chemical synapses were modelled with a sigmoidal input–output function. The model is shown in figure 3.12.

Selverston and Mazzoni found a high degree of similarity between the biophysical data and their electrical model, both in output functions and, more importantly to the correct operation of the stomatogastric system, in phase relationship.

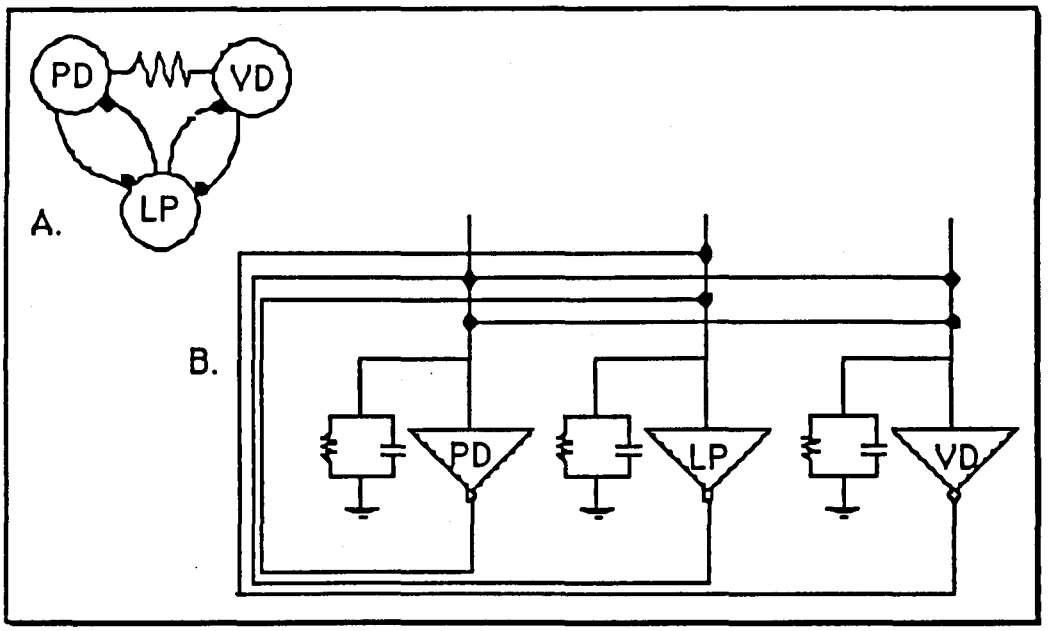

Figure 3.12 Lobster pyloric sub–network. A. is the connectivity of the network in the ganglion - black circles represent inhibitory synapses, the resistance component represents a gap junction connection. B is the electrically modelled version. (Adapted from Selverston and Mazzoni, 1989).

Larger Networks

Clark, Chen and Kurten (Clark, Chen and Kurten, 1989) modelled a sub–circuit of the olfactory cortex, based on the structure of the olfactory cortex, as described by Shepherd (Shepherd, 1979) . They use a coarse–grain model, which represents neuronal activity as mean spike frequencies in the analog domain. They found that this approximation technique gave good results when the model was compared with the output of the biological system. The input from the olfactory receptor cells is known to be a frequency coded function of stimulant concentration. For each neuron, the firing rate is determined as the sum of external and

internal (synaptic) stimuli, with any accommodation to firing frequency, minus a neuron threshold variable. The synaptic inputs are weighted, determining the sign of the synapse, and a fatigue factor scales the synaptic input. The output of the neuron is the sum of this function, which is sigmoidally scaled, with additional rate factors added in as the last step to account for synaptic delay and neuron component 'warm–up' factors.

The olfactory cortex for this model consists of three major cell types. The Mitral cell is the cell which receives the sensory input and transmits the output to other brain areas, while the periglomerular neurons and granule cells are interneurons, modulating the computations of the mitral cell. (see Figure 3.13) The 'tufted' cell, also present in the olfactory bulb, was not used in this model. The functional equivalent circuit is given in figure 3.13b, and consists of mitral cell models with input modulation performed by the periglomerular cell model, output being modulated by the granule cell. The connectivity allows a 'spreading activation' input function mediated by the periglomerular cell and a 'lateral inhibition' output function mediated by the granule cell.

The published results of this model show a sustained 20Hz oscillation in the circuit which is not dependent on the frequency of stimulation, as long as the stimulation frequency is within a certain range. This is said to correspond to the phenomenological measurement of EEG in animals.

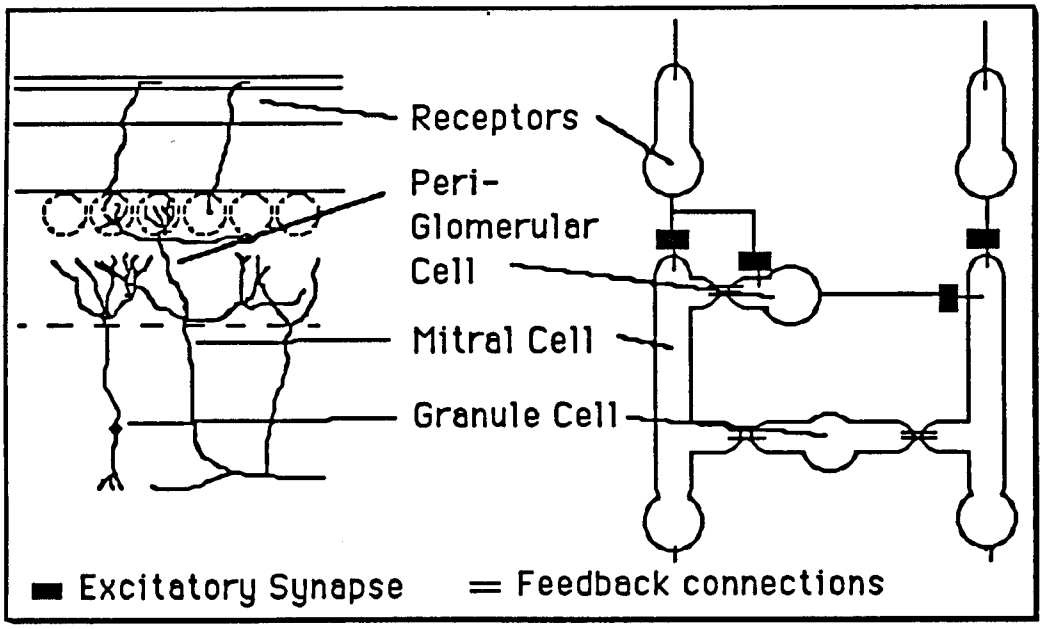

Figure 3.13 Olfactory cortex, A. representation of a local circuit and B. model circuit (Adapted from Clark, Chen and Kurten 1989).

Summary

Real neural networks are many orders of complexity above the representations that researchers can possibly create in reasonable computational spaces and time-scales, at present. The variation of membrane resistance along a cell process, alone, is computationally complex enough to require the fastest parallel processors to be anywhere near an accurate representation of the actual process of electrical propagation. Despite this, many researchers feel that a good enough model of real networks can be built with composite components, such that input/output mappings can be accurately followed from the measured characteristics of real neurons. This should allow neural circuits to be built which can imitate their real counterparts to a reasonable degree of

accuracy. The latter examples in this chapter show that this approach appears to work.

4

The Brain, Neuropeptides and noise:

Theory of signal transmission involving neuropeptide substances.

A great deal of work has been done on neurotransmitters within the last 30 years. Beginning with the 'classic' transmitters, such as Serotonin, Acetylcholine and Norepinephrine, and now heading towards the characterisation of so–called retrograde transmitters, candidates for which were Arachidonic acid and now Nitric Oxide, theoretically used in synaptic update mechanisms (Garthwaite, 1991). In the space between these two classes are a wealth of substances which have been shown to be neuro–active, some of which are simple molecules, fatty acids (Ordway et al., 1991), steroids (Joels & de Kloet, 1992) and others which are peptide–based and may be fairly large scale molecules. There are now well over 50 possible neurotransmitting substances to be found in the nervous system, and an increasing focus of research is the operational reason for the existence of so many possible transmitters.

Some researchers assume that the large number of putative transmitters is a relic of evolutionary history. Rossier (1985) gives the example of the neuropeptide, α-MSH (alpha-Melanocyte

Stimulating Hormone), which darkens the skin in amphibians, but which also exists in higher vertebrates without an apparent biological function. It has been suggested that neuropeptides may modulate the action of 'classical' transmitters, but the majority of researchers appear to feel that neuropeptides can be transmitters in their own right.

In general, it is expected that only a few of these possible transmitters will be operational in one specific area: But if all the transmitters are conveying information at the same time, it may be difficult to separate the effects of some of the channels, using the same transmitter, and 'noise' may occur as a result of signals crossing channels. This may, in fact, be an enhancement, as described later. The crossing of signals using different transmitters may be a method of signal propagation which allows multiple signals to act effectively in a limited space. This is the basis of concurrent propagation, which would allow multiple signalling activities to continue uninterrupted in the nervous system

Neuropeptides - The evidence for neuroactivity

The exposure of neuropeptides to the neuroscientific community occurred between the late 1970's and the present day, although some of the preliminary work on certain peptides goes back as far as the 1930's. The genesis of the idea of neuropeptides as transmitting molecules goes back to the work on hormones and neurohypophyseal peptides, and a particular peptide known as Substance P (studied by von Euler, see (von Euler, 1985)) In

general, the term 'neuropeptide' is applied to a peptide which has been shown, immunochemically, to be present in the brain, and as this work was not carried out until the late 1970's, the history of this field is not long.

The class of 'peptides' encompasses a variety of functional molecular structures, in fact, anything which is built of non–simple amino–acids. Thus, it is expected that a wide variety of such substances will occur in a living organism. If we pause for thought for a moment, one might find it rather surprising to find that peptides, given their numbers and distributions in living organisms, were not thought to take part in the signalling activity of the organism, even at only a low–level or slow time scale.

There are several classes of neuropeptides. The Opioids appear to have been most studied, and have been split into three groups, based on their precursor molecules – enkephalins, dynorphins and endorphins (Hughes, 1985). The Opioids appear to modulate sensory input. Other broad classes of neuropeptides are based on their somatic activities, of which the two main classes are gastro–active and vaso–active. Some peptides do not readily fall into neat categories, and tend to regulate specific functions (such as Oxytocin's regulation of uterine contraction and milk ejection), but within their regulatory function they can be defined as either releasing or inhibiting peptides. However, it is not intended to detail the somatic activity of particular peptides here, unless it is found to be particularly relevant to their neural activities. The purpose of this discussion is to elaborate on the occurrence and possible signalling activity of peptides in the nervous system, and as such, discussions will be limited to the

central nervous system and ganglia without too detailed a description of the somatic function of the ganglia, where possible.

Distribution of Peptides in the central nervous system

The occurrence of neuropeptides in the central nervous system is of greater importance in this treatment, because here, it can be argued, there will be a limited variety of functional importance attached to a particular species of molecule. A chemical may be an evolutionary residue as Rossier (1985) says, or it may take part in the functional life of the neural system. In external ganglia, the neuro–functional properties of the molecule may be confused with the somatic–functional properties, denying the assumption of a specific role. Unfortunately, difficulty in studying the central nervous system and the ease of study of peripheral ganglia, in situ, and with all connections intact, means that the central nervous system is rarely extensively studiable except in fixed preparations. Thus, most evidence comes from studies of peripheral ganglia. However, table 4.1 shows some peptides which have been localised to the mammalian cerebral cortex by immunochemical methods.

Three of the most widely studied peptides have also been localised to particular cell types. Vaso–active Intestinal Peptide (VIP) has been shown to exist in bipolar neurons with cell bodies in cortical layers II and III, which receive input from thalamocortical neurons and synapse onto the apical dendrites of pyramidal cells. (Morrison & Magistretti, 1985) Cholecystokinin (CCK) tends to occur over a wider range of cell types, as well as in bipolar

neurons, again in cortical layers II and III. Somatostatin (SS) appears in a heterogenous population of cells, mostly multipolar and modified pyramidal cells, with cell bodies distributed between cortical layers II–III, and V–VI.

Other peptides have not been localised as effectively as these major peptides but Substance P (SP) has been found in cortical cells, as well as in the widely documented peripheral unmyelinated sensory neurons (Kelly, 1985). Oxytocin and Vasopressin have been extensively studied, with indications that these peptides are widely distributed through basal nuclei and limbic system and are functional in a neuromodulatory capacity. (Sofroniew, 1985)

Neuropeptide Y (NPY) is one of a group of pancreatic peptides which can be found in the cerebral cortex, mostly in interneurons. This peptide also exists in the basal nuclei and limbic structures such as the hippocampus and amygdala (Emson & De Quidt, 1985). NPY can also be found in some central adrenergic neurons, in the locus coeruleus.

Angiotensin II
Avian pancreatic peptide (APP)
Bradykinin
Bombesin
Corticotropin releasing factor (CRF)
Cholecystokinin (CCK8 - refers to 8 base molecule)
Enkephalin
Molluscan cardio–excitatory peptide (FMRF–NH_2)
Neurotensin
Somatostatin (different versions)
Substance P
Vasoactive intestinal peptide (VIP)

Table 4.1 Peptide–like immunoreactivities identified in mammalian cerebral cortex. Many more can be found in peripheral tissues (see Schwartz, 1985) (table adapted from Morrison and Magistretti, 1985)

Co-existence of 'Classical' neurotransmitters with neuropeptides

Much of the work on Neuropeptides concentrates on the role of peptides as co-existent and co-released substances with conventional neurotransmitters. In some cases several peptides co-exist in the same terminal with a 'classic' neurotransmitter (See Table 4.2 for some known co-existence sites). Some researchers theorise that this co-existence acts to co-ordinate events in the nervous system (shown in Aplysia by Scheller et al, 1985). Other work shows that peptides can produce long lasting discharge activity which is interrupted by the release of 'classic' neurotransmitters (Horn & Dodd, 1985). An example of co-existence facilitating a response is the action of Acetylcholine (ACh) and VIP (Vasoactive Intestinal Peptide) in the sub-mandibular salivary gland of the cat. Here, VIP does not cause salivation on its own, but facilitates the action of Acetylcholine to produce salivation, possibly by increasing the affinity of Muscarinic receptors for Acetylcholine: ACh, on its own, produces a lesser salivation response (Lundberg & Hökfelt, 1985). VIP is not solely found in intestinally related areas, but is also present in cerebral cortex interneurones where it is thought to play a role in enhancing local blood flow (Morrison & Magistretti, 1985). Other neuropeptides seem to be even more important in the nervous system. Cholecystokinin (CCK), along with Neurotensin, is thought to play a role in regulating the release of dopamine in cortical dopaminergic neurons (which apparently have no pre-synaptic autoreceptors). The

release of CCK in this case, inhibits the release of dopamine, and is antagonistic to neurotensin which facilitates dopamine release (see Emson, 1985).

Classical transmitter	Peptide	Tissue/region (species)
Dopamine	Enkephalin	Carotid Body (Cat)
	CCK	Ventral tegmental area (Man)
Noradrenaline	Somatostatin	Sympathetic ganglion (G-pig)
		SIF cells (Cat)
	Enkephalin	Sympathetic ganglia (Cat)
		Adrenal Medulla (Several)
		SIF cells (G-pig)
		Locus Coeruleus (Cat)
	Neurotensin	Adrenal Medulla (Cat)
	APP/BPP/NPY	Sympathetic ganglia (Man)
		Medulla Oblongata (Man)
		Locus Coeruleus (Rat)
Adrenaline	Enkephalin	Adrenal Medulla (Several)
	APP/BPP/NPY	Medulla Oblongata (Rat)
5-HT (Serotonin)	Substance P	Medulla Oblongata (Rat)
	TRH	Medulla Oblongata (Rat)
	Substance P/TRH	Medulla Oblongata (Rat)
	Enkephalin	Medulla Oblongata (Rat)
ACh	VIP	Autonomic ganglia (Cat)
	Enkephalin	Preganglionic nerves (Cat)
		Cochlear nerves (G-pig)
	Neurotensin	Preganglionic nerves (Cat)
	LHRH	Sympathetic Ganglion (Frog)
	Somatostatin	Heart (Toad)
	Substance P and Enkephalin	Ciliary Ganglion (Avian)
GABA	Somatostatin	Thalamus (Cat)
	Motilin	Cerebellum (Rat)

Table 4.2. From Lundberg and Hökfelt (1985) showing co-existence of classic neurotransmitters with neuropeptides. Abbreviations are: 5-HT = 5-Hydroxytryptamine, ACh = Acetylcholine, GABA = γ-aminobutyric acid, CCK = Cholecystokinin, TRH = Thyrotropin Releasing Hormone, VIP = Vasoactive intestinal polypeptide, LHRH = Luteinizing hormone releasing hormone, NPY = Neuropeptide Y, APP = Avian Pancreatic Polypeptide, BPP = Bovine Pancreatic Polypeptide. Lundberg and Hökfelt make a number of reservation regarding the specificity of immunohistochemical techniques employed in the formation of this table.

Activity of Neuropeptides

Luteinizing–hormone–releasing–hormone (LHRH) has been studied in the Frog sympathetic ganglia by Jan and Jan. (Jan & Jan, 1985) The above table shows that LHRH is a co–transmitter with ACh in this ganglion. Jan and Jan studied the effect of blocking transmitter responses with specific chemical antagonists, on the excitatory post synaptic potential (EPSP). Their work was based on the presence of a long–term EPSP (termed the Late Slow EPSP) which was unaffected by ACh antagonists. They found that they could isolate the LHRH–like peptide in the terminals by immunohistochemistry and radioimmunoassays. They found greater concentrations of extracellular peptide after electrical stimulation of the peptide containing cells, and they found that they could mimic the late slow EPSP by applying a synthetic LHRH–like peptide to the sympathetic neurons connected to the peptide containing cells. Further, they found that LHRH antagonists applied directly before stimulation resulted in the absence of the late slow EPSP. Additionally, their evidence suggests that the activity of the LHRH–like peptide was extra–synaptic and effective over a distance of several tens of micrometers and a time–course of several minutes.

Scheller et al studied the bag–cell neurons of the abdominal ganglion of *Aplysia Californica*. Their work concentrates on three peptides which trigger egg–laying in the snail, of which the apparent hormone (due to its actual physiological spread and

slowness of action) is known as Egg–laying hormone (ELH), and the peptides were named α and β bag–cell peptide (α- and β- BCP). (Scheller,Rothman & Mayeri, 1985) The bag–cells of the abdominal ganglion are generally thought to be the trigger for egg–laying in *Aplysia*, as these cells are active before egg–laying occurs, and artificial stimulation of these cells induces egg–laying behaviour. The bag–cells produce ELH, α- and β- BCP, when stimulated by the application of peptides A and B, which originate from the Atrial gland some distance away. The activity of the bursting response from the bag–cells lasts for 20–30 minutes, allowing a considerable amount of peptide to be released into the abdominal ganglion. Specific target cells in the ganglion respond to the α- and β- BCP, augmenting the activity of some and inhibiting others, and each of these activities can be mimicked by the external application of collected peptides. The activity of α- and β- BCP is relatively short–lasting, being degraded within ten minutes, and these peptides have a locus of activity entirely within the abdominal ganglion, strengthening their roles as local transmitters, rather than global hormones.

Work on Substance P (SP) has concentrated on its action in the dorsal root ganglion of the spinal cord, where it apparently modulates the activity of sensory interneurons (Otsuka & Konishi, 1985). It has been localised in these areas by immunohistochemical and radioimmunoassay studies in Guinea–pigs, along with the primary transmitter Acetylcholine. The dorsal root ganglion connects the skin nociceptors and other peripheral ganglia to the CNS. Substance P is assumed to take part in a slow EPSP produced in inferior mesenteric ganglion cells, which is not blocked by

Acetylcholine antagonists, and can be mimicked by the external application of SP to the ganglion. The response to external application lasts from 20 seconds to 4 minutes, but repeated application within this time appears to saturate the receptor, such that the SP induced activity is dampened. Treatment of the ganglion with Capsaicin depletes the Substance P (Nagy, 1985) and abolishes the slow EPSP. There is also evidence that Substance P has some part to play in the transmission of painful external stimuli. Substance P antagonists appear to have analgesic effects when injected into an animal – injecting Substance P itself, elicits behaviour associated with painful stimuli.

The Opioid peptides are some of the most widely studied groups of peptides. They mediate action which is readily seen in behavioural change of the organism concerned. There are, in fact, three families of opioid peptides (Hollt, 1985), eighteen specific peptides, manufactured from three different precursors. These are Pro–opiomelanocortin (POMC), pro–enkephalin–A and pro–enkephalin–B. POMC–based peptides are known to be released from the pituitary, but also can be found in neurons of the arcuate nucleus, which project to the amygdala. Pro–enkephalin–A and –B derived peptides are found in the nucleus accumbens, hippocampus and spinal cord. It is known that there are three receptors with rather strange selectivities. (Weber et al, 1985) The receptors are classed as μ, κ and δ, and the strange receptor selectivity behaviour is seen as a switching of receptor preference, as intermediate peptides are processed into end products. This leads Weber et al. to suggest that the processing of the precursor molecules may also be modulated in some way to produce different

effects on the organism, by producing differences in the concentrations of particular intermediate peptides. A fourth receptor type (σ) is postulated, but is not accepted as an opiate receptor by all researchers, as it is apparently not affected by classic opiate blockers. (Zukin & Zukin, 1985) Opiates are most widely known with respect to their modulation of painful stimuli. This is a part of a proposed linkage with the substance P peptide in pain reception. Specifically, enkephalins are thought to inhibit Calcium influx into pre-synaptic terminals of substance P containing nociceptive neurons. The dearth of Calcium in the pre-synaptic terminal, inhibits the release of substance P (and other messengers), which decreases the reception of painful stimuli. (Kelly, 1985)

Neuropeptide Y (NPY) and Somatostatin (SS) appear to be closely linked, particularly in results of studies on rats. (Emson & De Quidt, 1985) These peptides are thought to be co-existent in non-pyramidal cells in cerebral cortex and basal ganglia, and make up 2%-3% of the neurons within the rat visual cortex. NPY has a profound, but Calcium dependent, effect on vascular smooth muscle, inducing a slow contraction of cat gut and arteries.

NPY also seems to have a linkage with the adrenergic system in the rat, which appears to be concerned with the control of blood pressure. This would seem to be consistent with the vaso-active properties of NPY. The linkage may be due to a similar action of the NPY to the catecholamine, but other studies have shown that NPY may increase the number of active adrenergic receptors on a prepared membrane (Emson & De Quidt, 1985).

Overview

The action of neuropeptides on the nervous system is unclear at the moment. It seems likely that the activity of neuropeptides is mediated through secondary intracellular messengers such as cyclic–AMP (see figure 3.7 and surrounding text) and certain lipids, or by altering the concentration of ionic species such as Calcium, thereby directly altering the exocytotic process in the pre–synaptic terminal. The evidence for the action of the opiates is in favour of the regulation of ionic species (Miller, 1985), but the activity of other peptides has not been sufficiently studied to give definitive actions for them. It does seem certain that peptides will have a slower time course than primary chemical messengers, in both release and de–activation. Many neuropeptide receptors are likely to be outside the synaptic cleft, which means that a slower diffusion process must take the peptide to the receptor (Jan & Jan, 1985). It does appear that peptides do have a de–activating peptidase system which will speed removal of the peptide from the extracellular space (Schwartz, 1985) rather than just allowing the concentration to fall to negligible levels by further diffusion. Such a de–activation system would appear to emphasise the action of neuropeptides as transmitters rather than irrelevant products of cellular metabolism, unless the de–activation mechanism is merely a metabolic waste removal process.

Other factors which may have to be considered are the site and rate of peptide production, the transportation of peptides to neural terminals and exocytotic fatigue, but these factors are

unlikely to play a large part in the availability of peptides to the normally stimulated neuron.

Information transmission in the nervous system

The evidence above implies that neurotransmitters can be classed as information transmitters: But what is the nature of the information that they transmit, and what constitutes a 'signal' in the nervous system? What is a meaningful grain size of information in the nervous system? Are there several levels at which information is meaningful? At the moment, we can only guess at the answer to some of these questions, but the following discussion attempts to propose some of the possibilities in this area.

Should we assume that the 'signal' in the nervous system is the separate depolarisation of a cell, which is then propagated along the axon; or is it the succession of depolarisations of an axon, bringing in the notion of frequency modulation as the 'signal'; or, perhaps, is it the concerted firing of a group of neurons which acts as the signal; or, in fact, are <u>all</u> of these characteristics inseparable parts of 'the signal'? The majority of current researchers appear to favour the frequency modulation schema for most signal propagation, but do not fully discuss the possible 'meaning' of such a signal type. This 'meaning of a signal' will not be discussed here either, as it is outside the scope of this functional analysis.

We can look at a few ways in which 'information' can be propagated, in order to clear the path for later analyses:

1. Information (in the true sense) - 'Figure' in the figure/ground dichotomy - a 'bit' of information is propagated to form an internal representation, and thence to the output processes or brain state. Propagation is assumed to be accompanied by a rise or fall in the firing frequency of neurons and is consistent with activation propagation theories.

2. 'Anti-information' (as in NOT('information'))- 'Ground' in the figure/ground dichotomy - can also be propagated, aiding the formation of the output/brain state. Propagation of 'Anti-information' is marked by a fall or rise of firing rate brought about by either excitation or inhibition propagation..

3. 'Constraint' propagation. - Information may be inverted at some stage in processing - The 'invert' is propagated, reaching an output state which defines the set of non-classes of responses to a stimulus. In an object identification context, this means that the output state defines the set of classes to which the object does *not* belong. Thus, by elimination, this method *defines* the object class..

Using these three possibilities we can construct some ideas about information processing and learning:

1. In the 'Informant' propagation model, the routing of neural units, and thus the representation of a symbol could be partially serial, and it may be possible that to reach a particular sub-state of the entire representational brain-state, routing must occur

through another sub-node of the network (as would, perhaps occur in analogical learning - before strengths of connections had enabled an easier route to be followed), thus lengthening the neural pathway required to be traversed before a decision can be made as to the classification of the object. In this instance a system which propagates constraints could be more efficient and faster than the ordinary information propagation approach, as the path of constraints would be more direct than those of 'informants'.

2. Psychologists have shown that it is possible to recognise 'what objects are not', before recognising 'what they are' in cognitive psychology experiments (from Neisser onwards (Neisser, 1967; Nickerson, 1972; Marcel, 1977)). In such a system, it is possible that parallel exploration of object attributes, as represented in memory, reject non–fitting cases before producing the fitting case by default. This strategy may be echoed by the brains' mechanism of propagation, whereby the propagation of constraints rules out entire classes of objects before settling into a state which recognises an object by the lack of activity in the 'memory trace' of the particular object.

3. Theories of template matching in cognitive psychology are supported by several reaction time experiments, but also confounded by others. (Again, see Neisser 1967 for early examples of the type of study discussed in this paragraph.) In these cases it is shown that where objects are drawn from only one of a number of congruent sets (numbers, friends' faces, geometric figures) reaction time is not affected, however, when presentations can be from many of these sets, reaction time increases proportionally

with the number of sets. Constraint propagation allows for these observations. Expectation of a particular set of symbols sets up a 'mode' of thinking, where the constraints are already set - recognition of the class of symbol is unnecessary, and processing requires only the recognition of the particular instance of the set. On the other hand, expectation can not work in a situation where multiple classes of objects are to be presented, and recognition of the instance of the object takes longer, as it also requires recognition of the class of objects. Constraint propagation would be a more efficient strategy in this case.

It is not known which, if any, of these propagation schemes are used in human information processing, but the existence of the large number of neurotransmitters found, seems to imply that there is a more complicated mechanism than simple informant propagation at work in this system. McBurney (McBurney, 1985) suggests that:

> *"If Different transmitter molecules generate different post-synaptic effects, not just in terms of excitation and inhibition but in terms of the duration of the post-synaptic conductance change per released quantum, they will contribute different characteristics to the input-output function of neuronal networks"*

McBurney uses this as the basis of allowing different 'weights' in a network, formed from differing signal sources, and hence, concurrent propagation of signals.

In the primary nociceptive system we may have a prime example of antagonistic peptide action between the enkephalins and substance P (see above reference to Kelly, 1985). This would be mediated by pre–synaptic Calcium concentration (which could also be considered as a signal messenger in this case, as it determines the amount of transmitter released into the synaptic space: Any other substance which modulates the concentration of Calcium within the pre–synaptic terminal must also be recognised as a signal transmitter). In this system, therefore, we can count at least four substances, without going into too much detail, which may have an effect on the degree of pain felt as a result of noxious stimuli - these are Substance P, enkephalins (eg. dopamine), Calcium concentration and the primary messengers of nociceptive neurons (5–HT). This process can be likened to an active biasing of the key components in the pain receptor mechanism, and is not a simple signal propagation.

Jan & Jan (Jan & Jan, 1985) hypothesise that the emergence of peptides as neurotransmitters falls between 'classic' neuro–transmitters and paracrine hormones - to give a continuous range of secretory products, which act as fast transmitters in the case of 'classic' neurotransmitters, or slower more diffusible and widely–acting transmitters further into the more recently discovered large molecule transmitters. Their theory is that:

> *"...in situations where the speed of action is not crucial, or perhaps a slower and more prolonged influence is more desirable, conceivably the presynaptic neurons may terminate*

in the vicinity of the postsynaptic neurons without making synaptic contacts with them."

"...Selective communication may be achieved if different neurons in a given region express different subsets of receptors, so that a transmitter released from a presynaptic neuron influences only those neurons nearby which have the right receptors on their surface. ... For this type of interneuronal communication to be used extensively in the nervous system without cross-talk between parallel pathways, a necessary requirement is that many different molecules are used as transmitters. Perhaps this is one reason for the multiplicity of peptides that are implicated as transmitters."

This selective communication would allow complex signalling methods to be continuous in the nervous system, giving a mode of communication with facets of each of the three methods mentioned above.

A word about noise

The term 'Noise' can be used in any signalling system to refer to that part of a communication medium which arrives at a reception point, without having been an intended characteristic of the signal at the source point. In many cases noise must be removed, at least partially, in order to make the original message

intelligible. Noise is a fact in most of our electronic communication systems, but does the term have any meaning in reference to a nervous system? A nervous system must be an inherently noisy environment. Thermal, chemical and electromagnetic 'noise' must be a part of any nervous system, but is it regarded as noise by the nervous system? In many cases, what may be referred to as noise can be a necessary part of the signal, supplying a context for the proper decoding of the signal. Silence may also be a part of a signal, for example, in Morse code.

In a nervous system, we assume that signal propagation must take place, and we ascribe signal–like properties to what appear to be obvious communication processes, such as the faster electrical and chemical translocations. Generally, at some level, we place a cut–off and expect any translocation process beneath a certain chemical concentration or electrical voltage value or time–scale to be irrelevant to the 'meaning' of the 'signal', and it will, therefore, become 'noise' in our terms: But is this a valid strategy for the nervous system? The field of artificial neural networks operates upon the previous assumptions, and yet evidence has been found that randomly injected activation of a low order, which corresponds to what would be called 'noise', actually improves the performance of a basic networks' learning cycle (Von Lehman et al, 1987) (Docking, 1989). The phenomenological conclusion is that 'noise' aids the learning process; But the action of the 'noise' is a statistical process. Injected noise may aid the learning cycle by essentially broadening the number of activation points in a finite point system, which allows the gradient descent strategy to run more efficiently (See chapter 2 for a description of gradient descent strategy); alternatively, it may appear to allow faster

learning, through a process of random search through the solution space.

In the living nervous system neuropeptides may be considered as 'noise', because they may appear to act slowly, and in diminished concentrations - but if noise helps the system to converge more quickly, then this form of noise may actually be part of the 'signal', and the definitions of signal and noise become blurred. In this instance it is possible for the actions of the noise to become indistinguishable from the actions of the signal, because the signal requires the noise to speed the formation of a desirable 'brain state'.

This 'noise yet signal' problem can be illustrated by a simple, roughly similar human example. A 'wink' is a signal to a human observer that information provided by a spoken message is meant to be interpreted in a special way by the observer. In the absence of a 'wink', the message may be interpreted in a normal manner. In another setting, the 'wink' might be considered to be a nervous 'tic' and be ignored by the observer, leaving the meaning of the spoken message unchanged. In a final setting, the 'wink' may be received by an observer who was not the intended recipient, and the message will be interpreted in another manner. In this way, the same overt signal has acquired three different meanings in the presence or absence of a physical movement which can be considered as either 'signal' or 'noise'. For the purposes of this study we can call a single 'noise' modulation event of this type a 'Modulum'. (Plural – Modulae)

Neuropeptide operators

Neuropeptides might act as relatively long-term operators, perhaps with graded on- and off-set. The action of a peptide may be to modulate the excitability of a neuron, without necessarily producing a threshold event, unless the cell is already in a highly excited state. The source of the excitatory (or inhibitory) neuropeptide need not be in contact with the target cell, and the effect of the neuropeptide would diminish with distance from the source cell, through a process of diffusion and enzymatic degradation. This would act as a variable 'biasing' element in an electronic circuit, the value of which would be dependent on the excitatory parameters of the source neuron and transmitter concentration, and therefore the distance from the biasing source. (See figure 4.1) In multiple neurotransmitter biasing situations, such as the proposed antagonistic effects of Substance P and enkephalins in the primary afferent C-fibres of the nociceptive system, the operator would conform to the parameters of an electronic differential active biasing system. This allows the concentration of all peptide-biasing elements to control the bias of the target cell.

It may be possible that different cells within a local neural cluster could express different neuropeptide receptor types on the cell membrane (Koch & Poggio, 1987), or this may occur even on different parts of the same cell. Such a system would indicate that differential activation biases could be applied to the cells of a cluster, or to different sections of dendrite, by diffusion of different peptides into the cluster from internal or external

sources. In a situation where many neuropeptides are in use, this differential 'addressing' ability can result in the formation of extremely complex computational circuits.

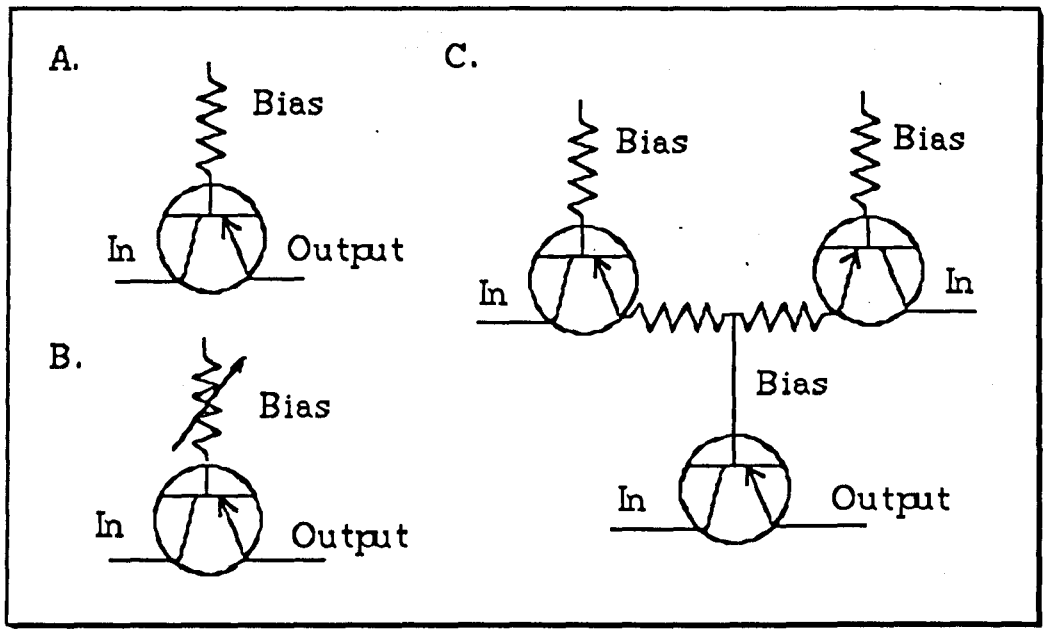

Figure 4.1 Electronic circuit equivalences of neuropeptide transmission systems. A. a normal non-variable biased transistor circuit representing the input and output characteristics of a simple neural cell. B. Variable bias produced by a one peptide system. C. Variable bias produced by a two peptide differential amplifier system.

Why neuropeptides?

What is the role of neuropeptides in the nervous system? Peptides certainly exist in the nervous system, and the evidence seems to show that they possess the ability to modulate the operation of some types of neurons.

What is the level of operation of these peptides? At the lowest possible functional level, peptides may modulate only very few neurons, and have so little effect that they cause no noticeable difference in the real operation of the network. In this case, they can be ignored and treated as either a developing concurrent propagation strategy, or the relic of a past transmission system which has been replaced by the current system.

At a higher level, peptides may modulate networks by greater or lesser degrees of 'noise' production. Noise has been shown to increase the learning efficiency of some limited neural network models, so it is possible that 'noise' will produce similar effects in real nervous systems. In this case, the use of noise in a real nervous system would imply that the signal to noise differential is fairly high, as coherent thought is usually maintained. In addition, the degree of accuracy required in the achieved excitation levels of individual neurons may be minimal, because of the presence of the 'noise'. A further implication is that the lowest meaningful 'informational atom' of a nervous system will be a number of neurons, in a certain configuration, rather than a single neuron. In this case, with the addition of noise at this level, the degree of accuracy of a single neurons' excitatory activity may not be sufficient to reliably represent a particular atom.

The use of peptides at this level ambiguates the use of the term 'noise', as the action of peptides need not be global phenomenon. Directional release of peptides from specific source cells or their processes, may produce 'directed noise' in a local cluster of cells around the release site. This directional use of 'noise' implies that the release of the peptide is a 'signalling'

process, but that the effect of this process is to add 'noise' to a specific part of the system.

At the next level, peptides may act as limited true information carriers, serving a local population of neurons, immediately adjacent to the source cell or axonal process. Peptides would produce an excitatory or inhibitory deflection in the excitability level of a target cell, allowing the target cell to be triggered by 'classical' transmitters, in lesser quantities than is usual for a 'classical' transmitter 'threshold' event. Peptides would act as a variable gain control in the mechanism of the target cells, allowing sensitivity adjustments, of particular circuits, to be made by other neuronal circuits. This level of operation increases the perceived complexity of the nervous system by many degrees. It could be argued that the number of peptides present within a given area, corresponds to the number of variable gain controls operating within that area. The connectivities of these controls determine the number of operational circuits in this area, and the concentrations at specific release times determine the gain.

The next level of possible peptide action, sees peptides as true neurotransmitters. These precipitate 'threshold' events in specific neurons, or groups of neurons, if the concentrations at distant sites allow. In the single neuron case, this would bring the possible grain size of the 'informational atom' back down to the single neuron level. All other details would be as for 'classical' transmitters, except for the vagaries of diffusion operation and the removal of the peptide from the active sites. There is little doubt that the modulation due to peptide transmitters would have a longer time course.

At this level, there is a possibility that peptides could be a 'sure' method of firing a neuron. This would occur by the release of a peptide within the dendritic tree of a specific neuron. The diffuse nature of the subsequent excitation would not allow veto operations, as apparently occurs for the 'classical' transmitters. Such a method ensures a binary response to stimulation.

At any of these levels, differential receptor placement becomes an important issue. If different peptide receptors are located on the same portion of a dendritic tree, antagonistic effects of different peptides are possible, similar to those of 'classical' neurotransmitters. If different regions of a dendritic tree manifest receptors for different peptides, different response properties can be elicited from the cell, because of the individual dynamics of the specific dendritic branches, imposed by this method of peptidergic modulation.

Experimental rationale

The purpose of the following study is to test the viability of some of the possible mechanisms of action of neuropeptides. This will be done by computational modelling of two different types of network. The first is a popular coarse grain model known as a back-propagation network, as defined in chapter 2. The second network is a medium grain model designed on the standard network simulator 'GENESIS', from the California Institute of Technology. These network models are being used to ensure repeatability of results and a standard 'form' for the experiments.

The coarse grain back–propagation scheme will be the major focus of the experimental section, as the low level of representation gives the model a more computationally amenable nature. This allows the model network to perform as a learning machine in a computationally feasible time–scale. Incorporating a learning system in the medium grain model would lengthen the project considerably. The back–propagation model can be executed many times in the time taken for the execution of a single contextually relevant medium grain model, so a greater range of results for the back–propagation model can be produced. In the case of the medium grain model, only a qualitative description can be produced, due to the small number of data points that can be generated in a reasonable time. Additionally, the back–propagation model is easily understood by most researchers, and is readily available in most artificial intelligence laboratories for repetition of these results.

The models will be executed under different conditions to simulate the diffusion of neuropeptides in different 'strengths', in an attempt to cover the different levels of possible action, as indicated above. This will not require structural modifications, as differing 'strengths' can be simulated by varying the efficacy of the peptidergic signal on the target site. The models must first be executed with no extensions to their structure, and then with progressively greater 'strength' values to the neuropeptide parameters. In addition, a model must also be executed with completely random noise characteristics in order to compare the action of random noise and the neuropeptide simulation results. The efficiency of these networks will be measured according to the length of learning time, in the case of the back-propagation

network, and the characteristic of the neuron output stage in the case of the GENESIS simulation.

It is hoped that these tests will enable a tentative proposition of the role of peptides (within current knowledge limitations) to be founded. It may be possible to differentiate between the effects of random noise on the network, and the effects of a messenger system based on the diffusion of messengers (a local, broadcast mechanism) between neurons. It is unlikely that these tests will be able to distinguish the level of the action of peptides in a nervous system, until a reference level can be found in the nervous system with which to compare the 'strengths' used in these tests. It may be possible to place a crude measure on the basis of relative degrees of activation by classical transmitters and neuropeptides, but this approach also suffers from a lack of biological reference levels.

PAGE HAS NO CONTENT

5

Experiments in Back–Propagation.

This section reports on the design, execution and results of the Back–Propagation model used in this study. The rationale for these tests is described in the previous chapter. The back propagation model used here is based on the original paper by Rumelhart et al. (Rumelhart, Hinton and Williams, 1986), and, to re–iterate, this type of network is used in preference to any other model because of its accessibility as a standard network model, rather than for any other reason.

This back–propagation model is described in detail in chapter 2, which eliminates the need for a detailed treatment here. The major focus of the design section is to present the differences in design required by the nature of the addition of a 'peptidergic' modulum characteristic to the back–propagation network. This characteristic will be referred to as 'Peptidergic leak' or just 'Leak' activation from this point onwards.

Model Design

The design of the back–propagation model to be used has to be reviewed from the original design, for the purposes of this study. The classical architectures of these models are usually fully

interconnected between layers, but unconnected within layers. This makes the virtual structure of a network two–dimensional at an abstract level. The model required for this study must be three dimensional, as the additional 'leak' level must be added with a representation of distance between all nodes of the network. This adds connections between each node, producing an entirely fully connected network with the original connection structure maintained at the original level, plus the 'leak' connections which interconnect each node, albeit, at a weaker connection strength.

Architecture and input/output behaviour

The architectural design for the model to be used, came from Colvin (Colvin, 1989). This treats each layer of 'synapses' as a separate entity. Since each unit of each layer is fully interconnected, it is only necessary to provide a set of inputs and outputs to each layer of 'synapses', so that it is not necessary to equate a particular input with a particular input cell. The inputs are only recognised as such by implication. The same is true of the weight matrix. All entities are treated as vectors with an implicit positioning only. The calculation of the sigmoid activation function, gradient and weight updates were taken from Rumelhart, Hinton and Williams (1986) and amended as suggested in McClelland and Rumelhart (1988) to operate within the confines of a limited bit precision computer system.

After some experimentation, it was decided for speed of operation to use a network with only 38 nodes in total. The network to be used is a three layer system with 15 input nodes, a

hidden layer of 15 nodes and an output layer of 8 nodes. The input vector is arranged as a 5x3 matrix, representing a letter of the alphabet. In total only the first three letters of the alphabet are to be used. These are coded as in figure 5.1.

Each layer of the network is fully interconnected with the previous and subsequent layers. There are no connections in the plane of the layer. The output units are to be trained to represent each of the input patterns in terms of its ASCII code. "A" will output a value of (decimal) 65 in binary code, "B" will output 66, and "C" will output a value of 67. These are also shown in figure 5.1.

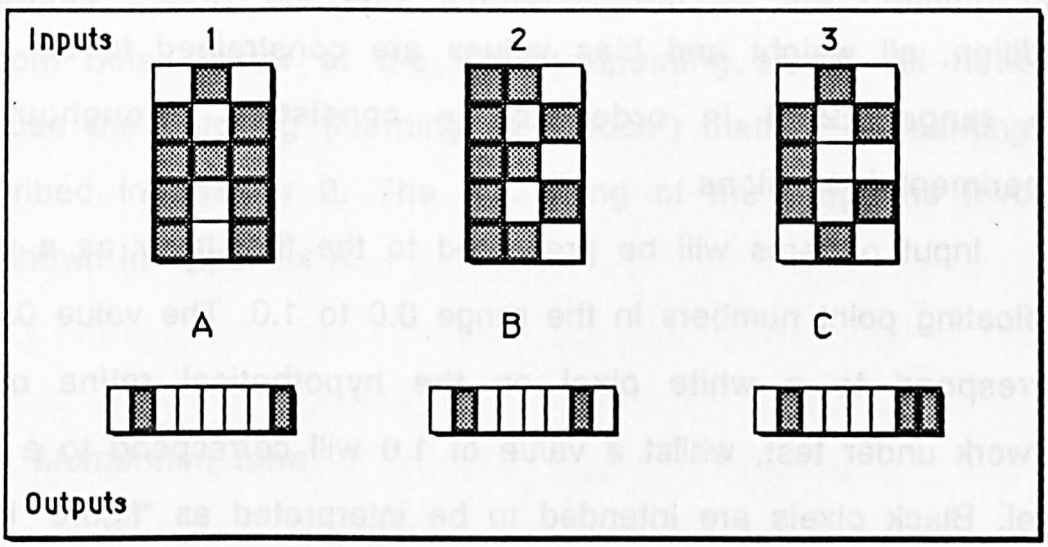

Figure 5.1 Network input and output patterns. The input vector represents a character, while the output vector represents the ASCII code for the character. Black squares represent activations of near 1.0, white squares represent near 0.0 activation values.

All networks will use the same physical architecture as described above and in chapter 2. Each model will differ,

functionally, only in the weight updating formulae, the assignment of random number starting values, and with the addition of some extra code for the latency effect of diffusion.

Initialisation of the network is by pseudo-random number. The pseudo-random number generator was taken from Colvin (1989). Both the weight matrices and unit biases are supplied by this method. The range of number generation for both biases and weights is ±3.0. In the case of the noise addition back propagation model, the "random" noise is also generated from this source, with a range of ±0.3, developed as a standard value from pilot studies. The pseudo-random number generator is seeded with the same value at the beginning of each different model test, thus each model begins with the same initial weight and bias values. In addition, all weight and bias values are constrained to be within the range ±20.0 in order to be consistent throughout the experimental sessions.

Input patterns will be presented to the first layer as a vector of floating point numbers in the range 0.0 to 1.0. The value 0.0 will correspond to a white pixel on the hypothetical retina of the network under test, whilst a value of 1.0 will correspond to a black pixel. Black pixels are intended to be interpreted as "figure" in the figure/ground dichotomy. Target patterns for the network will also be in the 0.0 to 1.0 range. Each target will be initialised to a value of 0.0 or 1.0, with the same interpretations as for input patterns. Output nodes of the network will be artificially thresholded for the purpose of presenting a binary output to a hypothetical post-processor. The threshold values will be ±1% of the range limits - 0.01 and 0.99 - representing a binary 0 and 1 respectively.

Training of the ordinary network will consist of presenting the input pattern, and its associated output pattern, to the input and output layers of the network, respectively. The network will then feed forward the activations relevant to the activation patterns and weight matrices. The feed back phase will calculate the errors found between the output pattern and the target pattern, and the previous layers will have their error coefficients calculated. The gradient will be calculated from the error coefficients, and the weights updated according to the gradient terms (one update for every "batch"). This procedure is exactly as stated in Rumelhart, Hinton and Williams (1986).

In the case of the noise–enhanced back propagation model, as described below, the only difference will be the addition of a random noise factor at the weight updating stage. All networks will use the batching (learning by "epoch") method of learning, as described in chapter 2. The full listing of the programs involved are shown in Appendix A.

Momentum term

The Momentum factor was introduced in the original paper of Rumelhart, Hinton and Williams (1986). Algorithmically, it consists of the addition of a proportion of the immediately preceding calculated gradient, requiring that each calculated gradient is saved in temporary storage until the next iteration, where it will be used. Momentum is added at the weight update stage, as in the equation:

$$w_{ij(k+1)} = w_{ij(k)} + \eta \Delta w_{ij(k+1)} + \alpha \Delta w_{ij(k)} \quad \text{(equation 5.1)}$$

(Where $w_{ij(k+1)}$ is the weight for the ij^{th} interconnection at iteration k+1, $w_{ij(k)}$ is the weight for the same interconnection at the k^{th} iteration, η is the learning rate parameter, and $\Delta w_{ij(k+1)}$ is the adjustment parameter, or gradient, derived from the error functions calculated in iteration k+1. α is the momentum rate, and $\Delta w_{ij(k)}$ is the gradient saved from the previous iteration.)

The momentum rate, α is generally set in the original literature to a value of 0.9, such that a large proportion of the previous gradient term is introduced into the current weight update calculation. Watrous (1987), points out that the momentum term acts to force the movement of the system downwards in the weight/error space, as the effect of adding in momentum tends to average oscillations in gradient calculations, leading to a stabler gradient direction indicator. He also states that the momentum term is not very efficient in aiding the crossing of plateau regions in the weight/error space, as, when the gradient is small, little momentum is added to the update equation and many more steps are needed to traverse the plateau, whereas in steep gradient regions, the momentum term effectively increases the step size, allowing a faster descent in the weight/error space.

In this study, the momentum term will be used as an added variable in the study of the model, such that models will be run with and without the addition of the momentum term (at a level of 0.9) in an attempt to view the effect of the momentum term on the results of both random noise and 'directed' or 'peptidergic' noise.

It could be argued that the momentum term acts as a form of potentiation in the back-propagation network, as an analogue of

long term potentiation in real neuron circuits. It is not the aim of this study to go into the details of this relationship.

Addition of noise

This extension sounds rather like a continuation of the principle of the Boltzmann machine. The reference to the addition of noise, in the paper by Von Lehmen et al (1987), is regarded as an analogue to the temperature of the noise system. This is a concept which was used by Hinton and Sejnowski (1986) in their exposition of the Boltzmann machine. However, the major difference is that in the Von Lehmen et al. system, temperature is a fixed constant which does not decrease over time. This means that this is not a simulated annealing strategy, but merely a way of decreasing the likelihood of the system finding a local minimum and staying within that minimum. It could be argued that with simulated annealing, there is a greater probability that a local minimum will be found as the temperature of the noise added to the system is decreased depending on the gradient of the descent, rather than over time, allowing the search to fall into, and remain inside, local minima. In the Von Lehmen et al extension, this is not a problem. The apparent behaviour of a network using this extension will be to find a global minimum and then oscillate endlessly with a mean error proportional to the temperature of the noise constant being used. This point would be computationally easy to find. In addition such a procedure would appear to be implementable in a network requiring less human intervention (such as the setting of time or gradient based annealing parameters).

5. Experiments in Back Propagation

The method of noise addition described in Von Lehmen et al, requires that the noise is added to the network weights at the time of weight updating, after the errors and activations have been calculated. The noise component is, therefore, easy to implement as it requires just one change to the formulae required for the back propagation algorithm. In the ordinary case for basic back propagation the weight update equation is:

$$w_{ij(k+1)} = w_{ij(k)} + \eta \Delta w_{ij(k+1)} \qquad \text{(equation 5.2)}$$

(Where $w_{ij(k+1)}$ is the weight for the ij^{th} interconnection at iteration k+1, $w_{ij(k)}$ is the weight for the same interconnection at the k^{th} iteration, η is the learning rate parameter, and $\Delta w_{ij(k+1)}$ is the adjustment parameter, or gradient, derived from the error functions calculated in iteration k+1.)

In the Von Lehmen et al. case the weight update function becomes:

$$w_{ij(k+1)} = w_{ij(k)} + \eta \Delta w_{ij(k+1)} + \text{random}(\pm n_{max}) \qquad \text{(equation 5.3)}$$

(Using the same notation as equation 5.2, with the addition of the term random($\pm n_{max}$), which introduces a random number with the range $\pm n_{max}$, where n_{max} is set to a constant before execution, usually at values between 0.0 and 1.0.)

Von Lehmen et al experimented with several values of n_{max} in both analog and discrete network activation functions, and with several different ranges for the initialisation of the network weights. They found that, using a network to test the learning of

the XOR problem, the addition of noise had little effect on the probability of convergence, whereas the initial network weight assignments had a significant effect (up to 10% in certain cases). However, they found that in weight limited networks. (where weight values are limited to a certain range) the addition of noise improved the probability of convergence to 100% over a wide range of n_{max} values (0.2 - 1.0). They explain these results as follows:

> "...clamping the maximum weight value essentially limits the weight space that can be explored by the network; the presence of noise encourages the system to thoroughly explore the restricted weight space and reduces the chance of trapping in local minima." (Von Lehman et al, 1987)

It seems, then, that the addition of noise within the 100% convergence range of Von Lehmen et al, should help to produce networks which are very stable under different initial weight conditions. Unfortunately they do not present figures for the *convergence rate* of these added noise networks. This will be a sub-task of this study.

Leak activation and the spread of activation in a network

The focus of this study is the addition of a 'peptidergic' activation factor, which is responsive to source cell activation,

diffusion distance and the activating nature (or strength) of the transmission itself. This is a theoretical study, in the sense that neuropeptides have not been proved, in a living system, to diffuse locally in nervous tissue. The existence and release of neuropeptides has been shown (see chapter 4) from source synapses, and they have been shown to modulate the responses of target cells. The nature of the linkage between these two points has not been explicitly formed, so the foundations of this study require several assumptions as to the nature and parameters of 'peptidergic' transmission.

If we begin with an assumption that a transmitter derives from synapses related to the main axon of a source cell, and that a large proportion of this transmitter is of a classic type, which is taken up in the synaptic cleft, then there is a continuing possibility that some transmission occurs outside the cleft, which will relate to the generally activity related 'noise' or 'Leakage'.

This means that, at a target cell (j), activity will be defined as:

$$a_j = \Sigma_i . w_{ij} \, a_i \qquad \text{(equation 5.4)}$$

(where a_j is 'classic' activation of a node, w_{ij} is the weight between nodes i and j and a_i is the activation of the input node)

$$l_j = \beta \, a_j \qquad \text{(equation 5.5)}$$

(where l_j is the petidergic leakage from synapses local to node j and β is a constant peptide strength factor)

$$a_{sj} = (1 + \beta) \, a_j \qquad \text{(equation 5.6)}$$

(where a_{sj} is activation from synaptic messengers)

$$a_{Ij} = \Sigma_k \; a_{dk(t-d)} \cdot f(d) \qquad \text{(equation 5.7)}$$

(where a_{Ij} is the total activation of node j from extrasynaptic sources, a_{dk} is the activation diffusion from node k at time t minus the distance, d, of node k from node j, and f(d) is a function of distance from node k. k = 1 .. n, where n = the total number of nodes in the network)

In this model it is assumed that 'synaptic' transmission is complete within a very short period of time, too small to be modelled precisely in a model of this coarseness. Diffusible transmitters, though, will involve some form of lag, which should be distance related, but again in a fashion relevant to the grain size of the model. Latency in diffusion can be modelled by using a round, linked list, which points to at least 3 vectors (with the number of nodes in the network as elements) so that, at each iteration of the network, the diffused activation is added in from further elements. Thus, Intra-layer activation reaches the target node after one iteration, activation from the next layer reaches the target node after a delay of two iterations, and so on. We simulate this delay by offsetting the calculated activation according to the layer in question. ie. the activation to a particular node is calculated separately and stored in a vector, based on its distance from the current layer, so that this vector is only used to simulate the activation on a node when its turn comes around.

At each iteration, the activation reaching a target node is calculated for each cell in the network, but values calculated for different layers are stored in different vectors; such that by the time a vector comes to be used in peptidergic activation calculation it has an inherent delay. There is some difficulty in this, in that

peptidergic activation comes from both the synaptic termination on a target cell and (roughly) the soma of the source cell. Therefore the same level of activation has to be used in calculating the activation values for 2 different layers. It can be argued that in a completely connected network (like this one), the peptidergic activation reaching the target cells is completely correlated with the activation from classic transmitters over the entire layer, and can therefore be ignored as a factor of activation of the cell itself. But it must be taken into account that this peptidergic signal leaks into the surrounding space and contributes to the activation appearing to come from the *target* cell — Therefore the peptidergic activation must be calculated at the peptide strength rate from the source cell, and must also be added to the target cells' diffusible peptidergic activation value, as a function of the remaining peptides after proteolysis – we can probably make a rough guess at about 30% of the peptidergic activation leaking from synaptic junctions – and this makes a calculation of input activation of a target cell:

$$a_{tj} = a_{sj} + a_{lj} \qquad \text{(equation 5.8)}$$

(where a is the total activation of node j, from all sources.)

and the diffusible activation from this cell will be:

$$d_j = 0.3(\ l_j + 0.3\ a_{lj}) \qquad \text{(equation 5.9)}$$

(where d_j is the total peptidergic activation appearing to emanate from node j. This factor is the single node component of the activation component (a_{dk}) in equation 5.7)

So, the calculation of local and diffusion transmitter activation is a largely linked interdependent process, which is difficult to model.

In the case of this model, it means that we must calculate the outgoing diffusion activation (D-activation) on the the basis of a normal 30% somatic secretion, plus 30% of the incoming synaptic activation from 'Peptidergic' transmission.

Representing three dimensions

Adding a three–dimensional mapping to the network is a relatively easy process. Classical back–propagation networks can be thought of, in abstract terms, as dimensionless entities, because the 'distance between elements' is not a concern. The network is usually presented as having a flat two–dimensional topology (as in figure 2.4 of chapter 2) with equal distances between the nodes of the network. Input and output are taken as vectors to be applied and read in a linear fashion at the two ends of the network. In effect, the dimensionality of the network is usually imposed by the researcher and their ideas of segmentation in these vectors. Figure 5.1 is an example of this. The characters are represented on a two–dimensional plane as recognisable letters of the alphabet in a 3x5 matrix, but are fed into a two–dimensional network as a 15 element vector. This study requires that we arrange the input nodes to represent a 3x5 matrix as in figure 5.1. Subsequent layers of the matrix may take any form, but it was decided to arrange these nodes in a similar manner. The final network was given a three–dimensional structure as represented in figure 5.2.

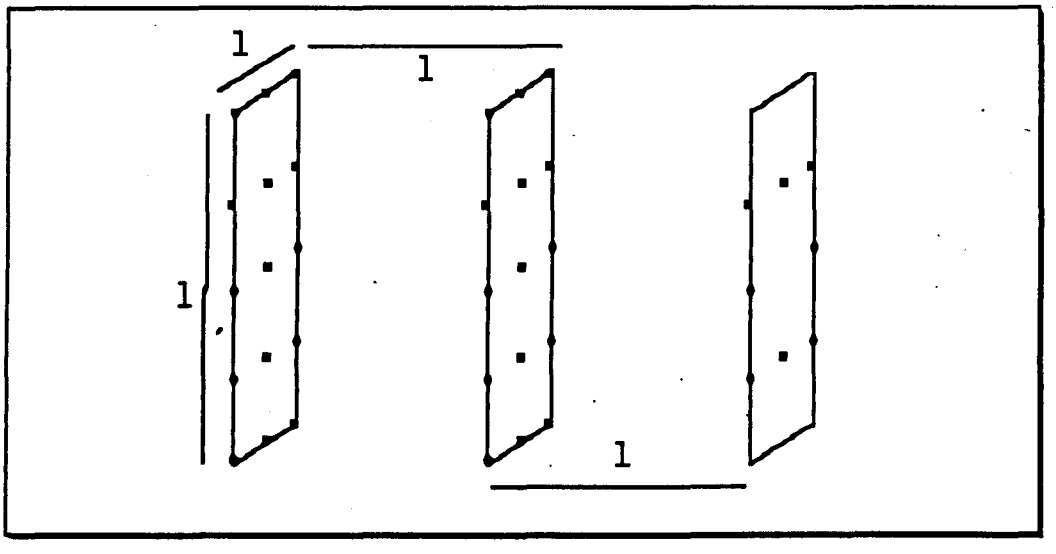

Figure 5.2. Network design for a three–dimensional model. All single lines are of unit length. Black dots represent nodes. The left side corresponds to the input plane, the right side to output. Vertical node displacement is 0.25 units, horizontal displacement is 0.5 units.

The distances between the nodes can be calculated by the two–dimensional Pythagorean equation, as inter–plane distances are of 1 unit:

$$d = (X_2 - X_1)^2 + (Y_2 - Y_1)^2 + 1 \qquad \text{(equation 5.10)}$$

As these values will be invariant, they are stored as a set of matrices encoded in diffusion constant/distance format. This also eliminates the need to recalculate distances and diffusion constant/distances during execution and speeds the execution of the network. These matrices can be examined in Appendix A.

The diffusion constant was chosen as a standard inverse exponential function:

The diffusion constant was chosen as a standard inverse exponential function:

$$c = \frac{1}{e^d} \qquad \text{(equation 5.11)}$$

(where c is the diffusion constant and d is the distance between nodes)

This gives a maximum output at unit distance of .368, roughly corresponding to the most effective 'noise' levels in random noise experiments. The maximum achievable level is .779 for the smallest inter–node distance (0.25u) and the minimum level is .086 at the largest inter–node distance (2.45u). The diffusion constant/distance measure is scaled with a 'peptide strength' factor and the output activation of the source node to determine the activation value to be applied to a target node, after the delay imposed on it by the node–node distance:

$$At_{i+1} = At_i + As_i * c * P \qquad \text{(equation 5.12)}$$

(where At_i is the activation of the target node, As_i is the activation at the source node, c is the constant calculated in equation 5.11 above and P is the 'peptidergic' strength factor)

In practice, the delay is simulated by an array of additive vectors, one element of the vector for each node in the network, and one element of the array for each time–delay period simulated, which is updated during each iteration of the network to correspond to the activation reaching a particular node during the

foregoing time period. The array is 'looped' to maintain the continuity of the time delay.

Measuring network efficiency

It was decided that the most important measure of efficiency would be the convergence rate – this is a measure of how many iterations the model requires to reach a stable output state, with a mean square error of 0.0001. This translates to an error of 0.01 per output unit, or 1% of the activation range.

Model Execution

Testing the model

Choosing standard settings for several of the parameters involved in running the back–propagation network required several preliminary studies. Many of the parameters are taken as a standard in the back–propagation methodology, while others are varied to suit the requirements of the study. The parameters available are often too numerous for any one study to produce detailed descriptions of the network behaviour whilst varying all parameter conditions, so it was decided to use three levels of parameter settings for this study. Some variables were to remain at 'standard' settings, as defined in the first back–propagation

research papers. Other settings would be optimised, based on investigations of varying parameters in the pilot studies. The final parameters which were deemed more important for the understanding of transmission within the system, were varied in the final models. In 'classic' network models, the most variable of all parameters is taken to be the 'Learning Rate' parameter, a precedent which this study has adopted.

Fixed parameters in the study models are:

- Use of Unit Bias value
- Activation Momentum value

Unit bias is a parameter which represents a symmetry breaking variable. This is set to a random value before the network is executed, with an update mechanism which allows the bias to vary in conjunction with the activation of the node. This is not a requirement for a back–propagation network, but usually leads to more stable network behaviour.

The momentum value in models with added momentum is generally set at a value of 0.9 times the previous value of a particular weight (Rumelhart, Hinton & Williams, 1986). This value of the momentum term has not been varied in these tests. The *use* of the momentum term has been varied, as this parameter can produce wide variation in the efficacy of networks

Setting optimised parameters

In the first experiments, which were executed in order to set the optimised parameters, a total of 4,800 models were run under different conditions, of which 2,215 converged within 20,000 iterations, although not all remained in a stable state. The optimised parameters are those which are set to a fixed value, for the entire length of the final set of experiments.

Optimised parameters are:

- Noise Range
- 'Leakage' Auto–Activation
- 'Leakage' Plane–Activation
- 'Leakage' scaling factor

Von Lehman et al (Von Lehman et al, 1987) found that random 'noise', introduced into the node update mechanism is effective over a wide range of maximum values. As 'noise' values will be fixed at a standard level in the final set of experiments, tests were performed to determine the optimal setting for the random noise variable over 5 Learning Rate intervals of 0.5 to 1.5, in 0.25 unit intervals. A summary of the results of some of these tests is shown in figure 5.3. The data pertaining to this figure, and further results can be found in Appendix B. On the basis of these results, it was decided to set the random noise value to a maximum value of

±0.3; a point at which the network appears to reach stability at a faster rate.

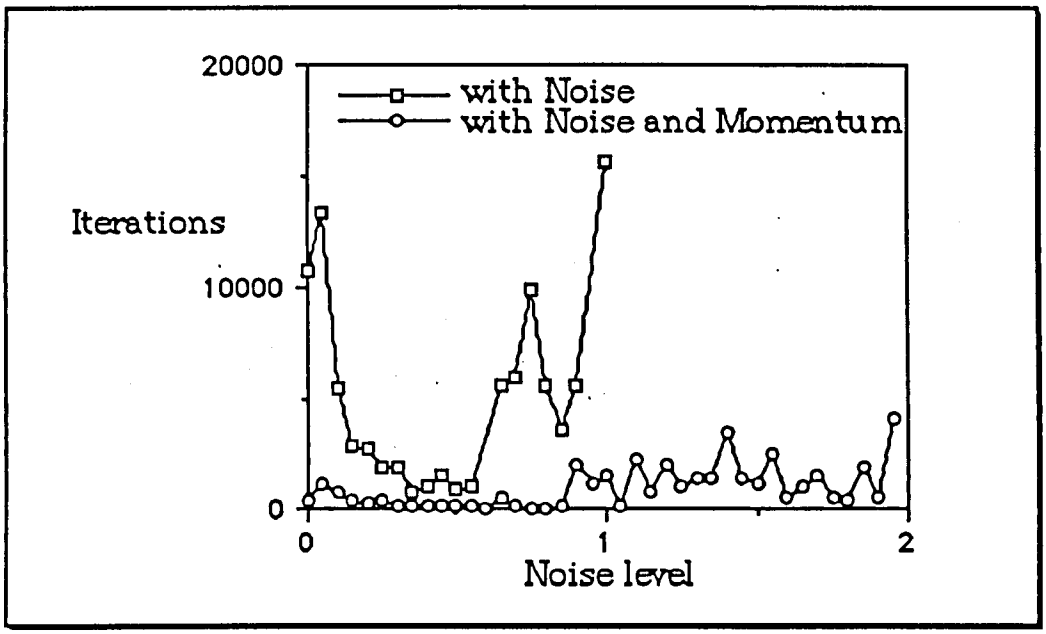

Figure 5.3. Random noise plotted against the number of iterations to convergence.

The 'Leakage' model in this study presents a fully connected sub–channel for signal transmission for each of the nodes. The model allows direct transmission (in the leakage domain) from a node to every other node, including nodes with a separation of more than one layer, and also to those in the same layer. 'Leakage' *auto–activation* refers to the phenomenon of allowing a feedback loop to exist between the activation function of a target node and the 'peptidergic' leakage function at the same node. If this is allowed, 'leakage' activation from a node will be added to the activation level of the same node. A pilot test was undertaken to discover if auto–activation had any effect on network behaviour. In practice, allowing or disallowing auto–activation produces no detectable effect, as shown in figure 5.4. This method of assessment was

abandoned for the remainder of the programme of study. All subsequent models studied have auto-activation enabled.

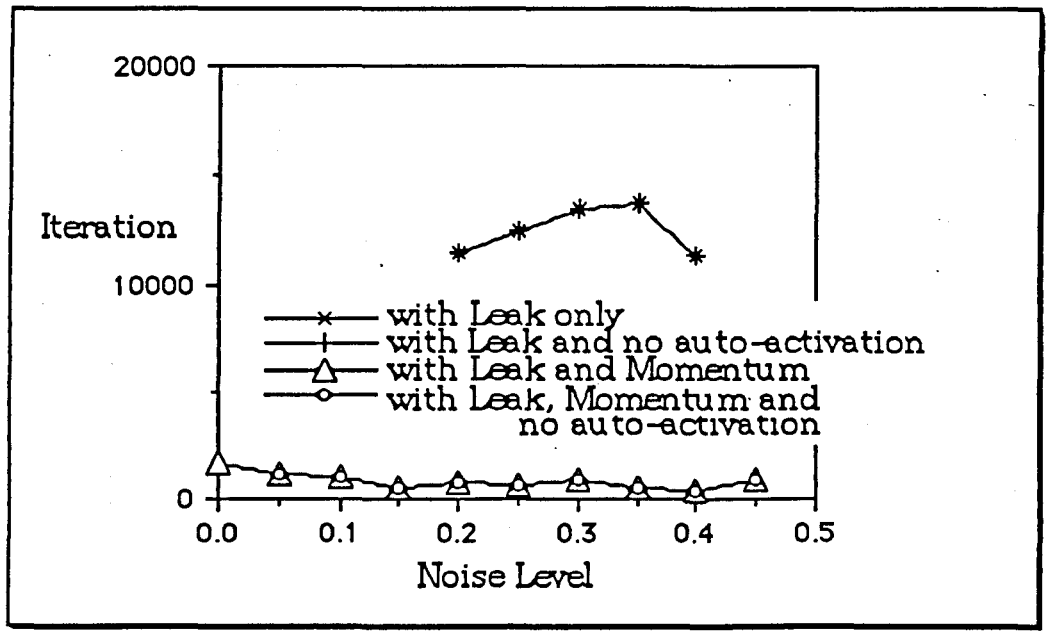

Figure 5.4. Number of iterations to convergence over noise level. .Auto-activation of a back-propagation network under the 'leakage' activation model. No detectable effect can be found.

'Leakage' *plane-activation* is a variant of the model to design which switches off intra-layer 'leakage' activation calculation. In this model, 'peptidergic' activation reaches only nodes in layers not belonging to the normal plane of the source node. This represents a parallel to the normal working of the back-propagation algorithm, which connects only along inter-layer arcs. Testing the switching of intra-layer 'leakage' activations gives contradictory results, appearing to depend on the initial settings of the weight values. This is show in figure 5.5. As the setting of initial weight values is a random process, it was decided to continue with a fully connected (with intra-layer activation calculation) network for the remaining models.

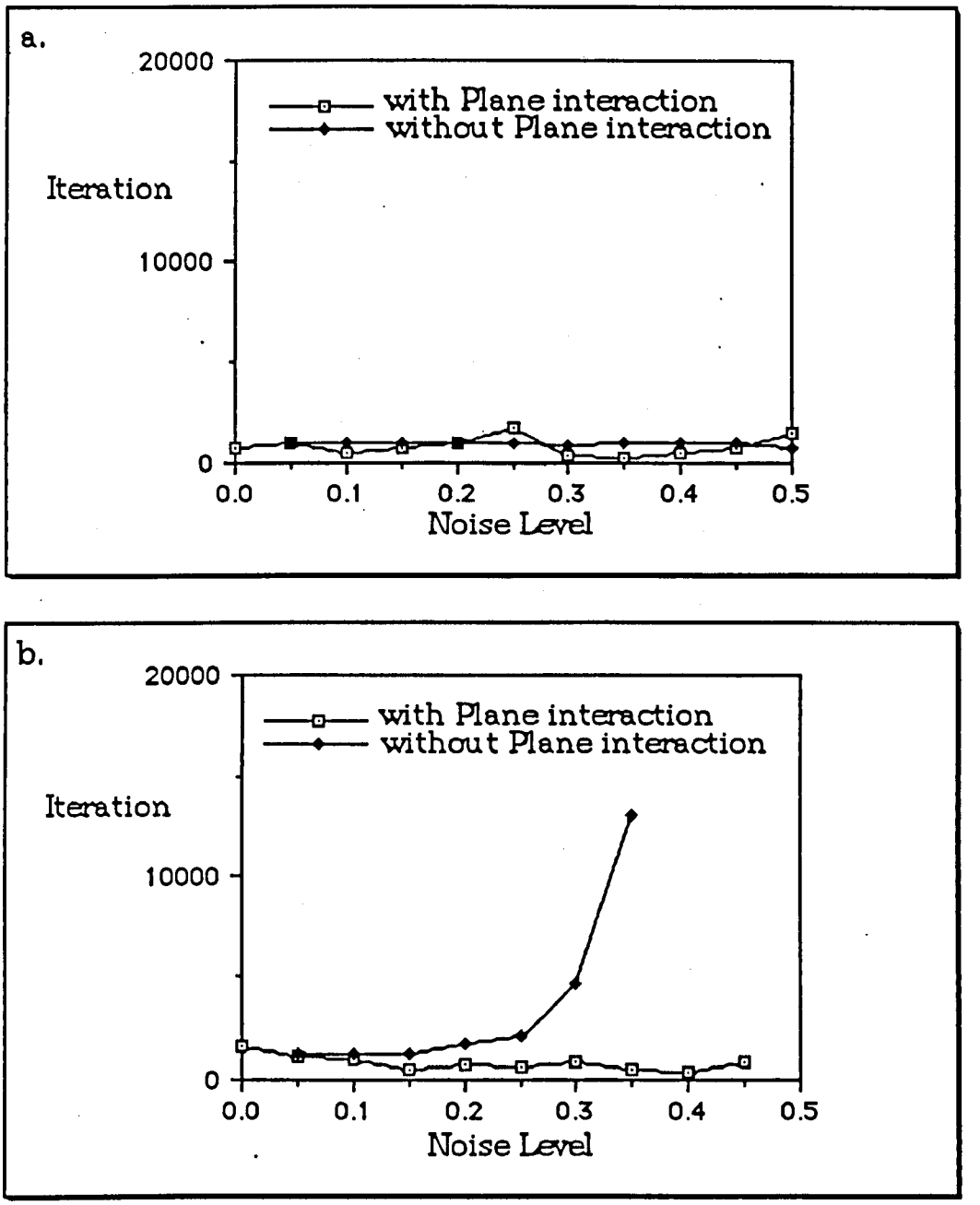

Figure 5.5. Two graphs showing the contradictory nature of intra-layer activation switching, with the number of iterations to convergence over noise level. Graph (a) shows little change in the rate of convergence with intra-layer activation turned off; while graph (b) shows a lack of efficiency with intra-layer activation turned off, for the same network under different initial conditions.

The 'Leakage' scaling factor is described earlier as a variable which determines the perceived 'strength' of 'peptidergic' activation to each node of the network. In this study, such a factor must be represented by an arbitrary value, as there is no way of selecting a realistic setting from a 'live' network - particularly as the back–propagation network is nothing like a 'live' network. It was decided to perform a series of tests, similar to the random noise range setting tests, in order to find an optimal setting for the 'leakage' factor. The summary results (figure 5.6) show that the efficacy of 'leakage' activation increases over a range of 0.2 to 0.4. It was decided to select a value of 0.35 for the final experiments, as an arbitrary value within this range.

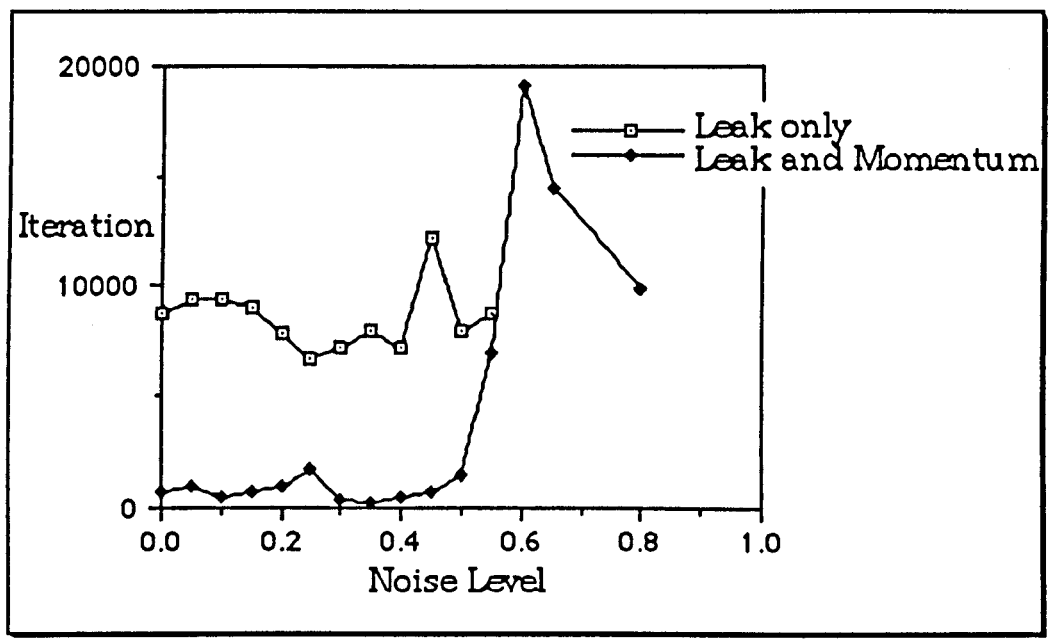

Figure 5.6. Number of iterations to convergence over 'leakage' scaling factor. Lower levels of scaling factor appear to give better network performance.

Running the final model

Having set the optimised parameters for the test, it was decided to compare the execution of the model over various run–time parameters, These were, in order of increasing importance:

- Learning Rate
- Type of 'Noise' (including 'peptidergic' leakage)
- Effect of the use of Momentum
- Effect of Initial weight settings

A total of 3,600 models were executed, of which 2510 converged within 20,000 iterations. Each model was tested with a range of Learning Rates, from 0.05 to 10.00, with intervals of 0.05 units. Two types of 'Noise' function were used. 'Peptidergic Leakage' activation was applied to one third of the models executed, and 'Random Noise' functions were applied to another third of the models. Momentum was added as a binary function over the range of 'Noise–type' tests. Other parameters were fixed as stated above, with Random noise at a level of ±0.3, and 'Leakage' activation strength as a factor of 0.35.

There were three initial weight settings, designated by the random 'seed' used in the model. Seed settings of '43', '67' and '89' were used. Producing more models based on initial 'seed' settings would have allowed a 'mean' behaviour to be calculated, but this would have required many more model executions than was feasible in the time allowed. Results are, therefore, shown for all initial

settings, and conclusions must be drawn based on all three behaviours.

A note on Learning Rates

It must be noted that, in these experiments, learning rates of up to 10 have been used. This extended range of learning rate is used as a means of covering an entire range of possible learning rate values, and is not intended to represent a set of realistically reasonable learning rates. In practice, it has been found that learning rates are reasonably set to a value of about 0.5. (Rumelhart, Hinton & Williams,1986) Such a setting increases the learning time, as can be seen from these data, but it also increases the stability of the resulting networks.

It is a normal practice for a network to be allowed continual learning, in the expectation that learning can be continued if new objects are added to the stimulus/response set. In the case of the lower learning rates, this is feasible, as stimulus/response items are stabilised upon having converged. Correct responses to stimuli occur indefinitely once the stimulus/response associations have been learned. This is not the case in higher learning rates, however. There is no obvious cut–off point of learning rate values, but in general, as the learning rate approaches a value of 1.0, the resulting network fails to 'remember' a stimulus/response pair, even though it appears to have 'remembered' on one or more occasions. This is not 'convergent' behaviour, where the stimulus/response items are correctly matched in perpetuity, but may be a 'one–off' occurrence, which demonstrates a 'lucky' finding

of the correct weighting scheme to achieve pairing. A consequent change of the weighting may, and usually does, alter the associations, so that the correct responses are no longer obtained.

In this study, one occurrence of a correct stimulus/response pairing, for all pairs, has been used as a trigger for recording the number of iterations to the correct response. This has been done to create a result set which covers a broad spectrum of behaviour. In all cases, actual convergence was recorded, but has not been displayed in the results. In general, spurious convergences can be implied by the increasingly erratic behaviour of the results, as learning rate is increased. This can be extended to the higher random noise and 'peptidergic activation' values, each of which become more erratic as values are increased. Interpretation of the following results, then, is a matter of assessing the gradient of the graph section, before assessing the relationship between the different operational conditions.

Results

All results shown in graph form in this section are sub–sets of the full data. This represents the authors' view of a 'reasonable' range of data for display, based on the above information about 'convergence' over higher ranges of Learning Rate parameters. The full data is available in numerical form in Appendix C, and in graph form in Appendix D.

Base models with no extensions

Executing the base models under the three initial conditions used in this study shows that the initial conditions imposed by the random seeds differ markedly, particularly in the seed(89) condition. This condition fails to converge in under 20,000 iterations under 'normal conditions', in the terms of back propagation models this is a learning rate of between 0.0 and 1.0. The behaviour of the seed(43) and seed(69) are very similar except when the learning rate reaches a value of over 10 times the normal values.

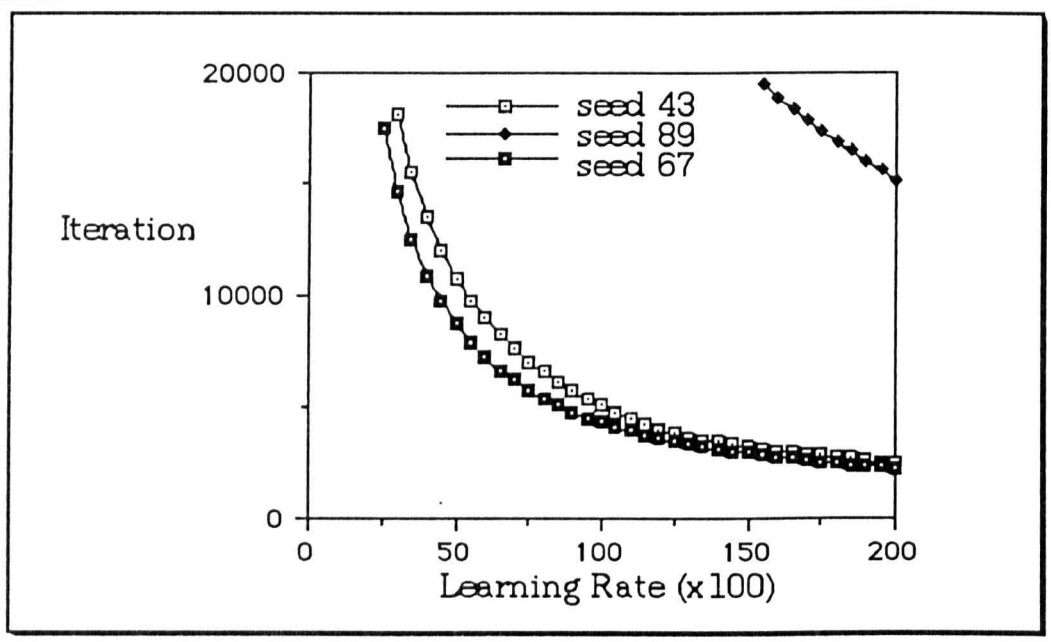

Figure 5.7. Back Propagation model using three different initial random seed settings. Number of iterations to convergence over learning rate.

Base models with Random noise

Random noise models show a remarkable stability across the range of initial seed settings, and also across the range of learning

rates. In this case, the seed(67) model is the least efficient at low noise learning rates, but then drops to remain similar to the seed(89) model. The seed(43) model shows an apparent improvement on the behaviour of both seed(89) and seed(67) models.

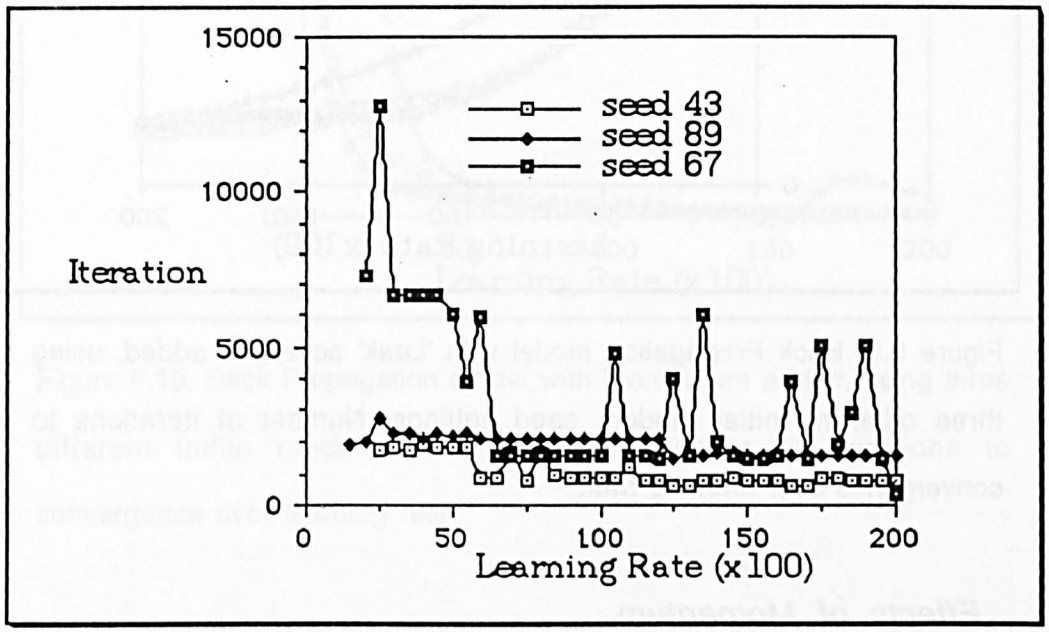

Figure 5.8. Back Propagation model with random noise added, using three different initial random seed settings. Number of iterations to convergence over learning rate.

Base models with Leak activation

Leak activation models show, as before, the similarity imposed on the models by the initial conditions. Seed(43) and seed(67) models follow almost the same trajectory in figure 5.9, leaving the seed(89) model displaying abnormal behaviour and instability after a learning rate of approximately 5.0 (shown in Appendix D).

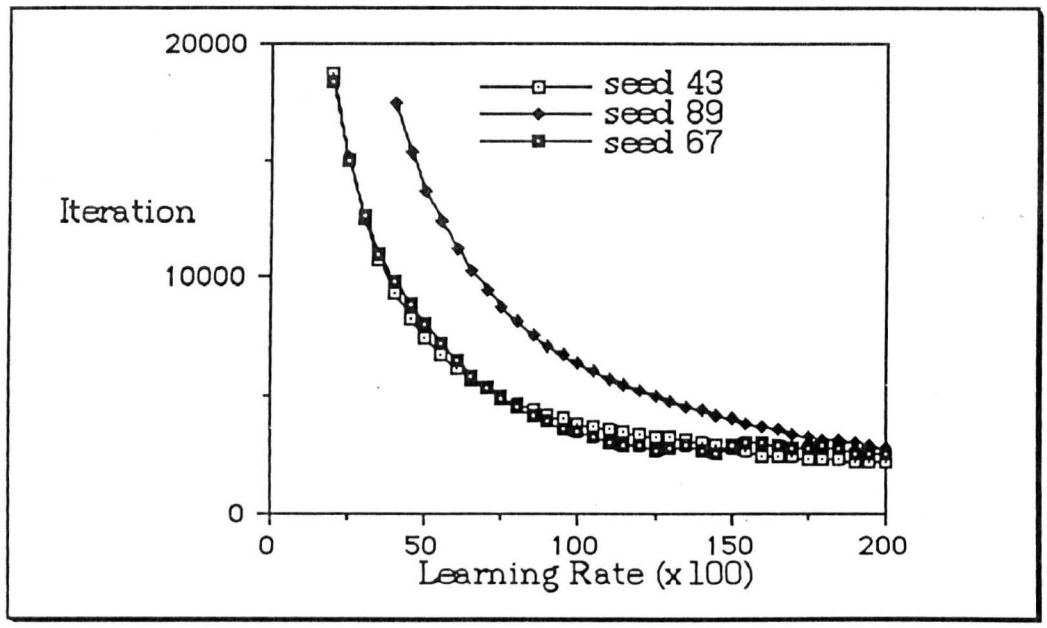

Figure 5.9. Back Propagation model with 'Leak' activation added, using three different initial random seed settings. Number of iterations to convergence over learning rate.

Effects of Momentum

Momentum imparts a high degree of stability on the back propagation model only in the lower ranges of the learning rate parameter settings. This is one reason for the reference to the range of 0.0 to 1.0 as being 'normal conditions' for the back propagation model. At learning rates of 0.5 to 1.5 all networks converge in a very short time, and the traces of the graph are largely indistinguishable in figure 5.10

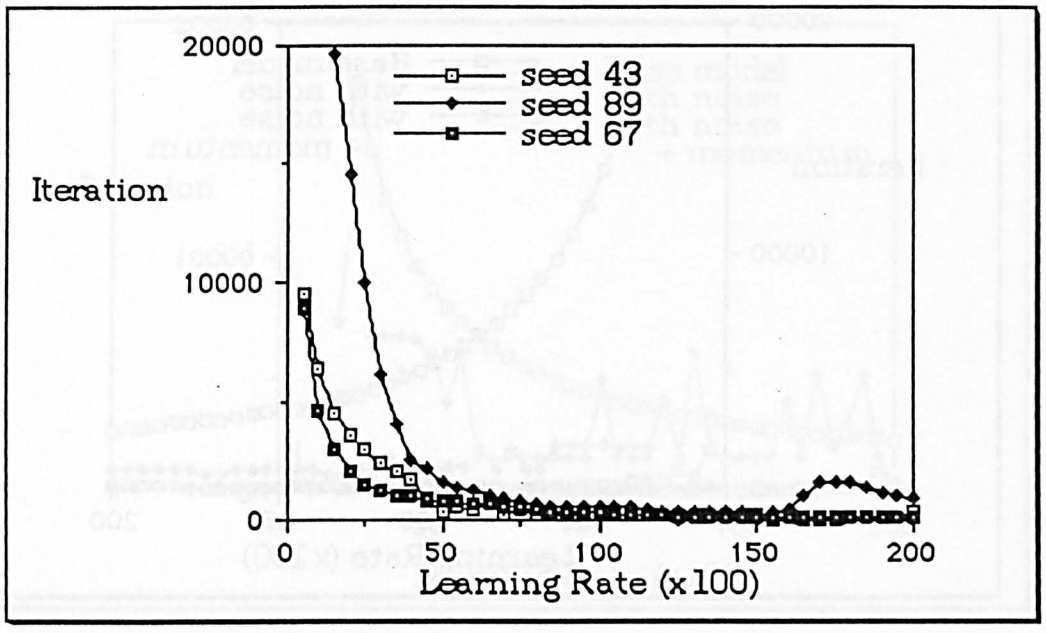

Figure 5.10. Back Propagation model with Momentum added, using three different initial random seed settings. Number of iterations to convergence over learning rate.

Comparison of Base models with added Noise and Momentum

Noise adds a greater degree of convergence speed to the ordinary back propagation model. Coupled with momentum, noise adds even more speed and stability to the behaviour of the network. This occurs over a large range of learning rates, except in the seed(67) initial condition of figure 5.13. The most stable ranges of noise and momentum are in the ranges of 0.0 to 2.0, with the exception of the seed(89) model, showing a great stability across the entire range tested.

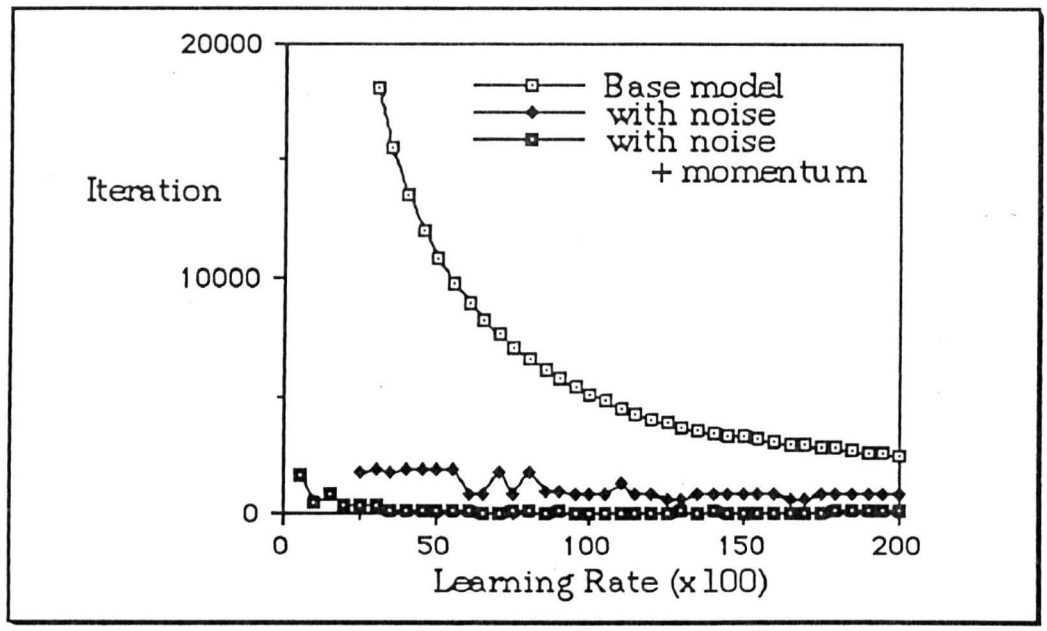

Figure 5.11. Back Propagation model with Random noise and Momentum added, using a single initial random seed setting (43). Number of iterations to convergence over learning rate.

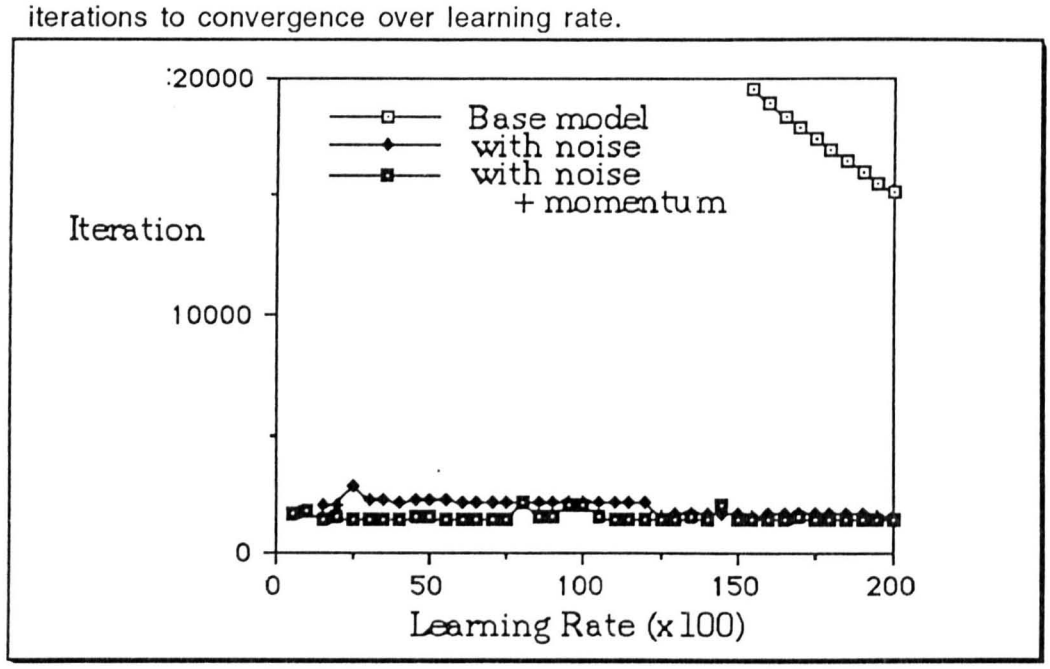

Figure 5.12. Back Propagation model with Random noise and Momentum added, using a single initial random seed setting (89). Number of iterations to convergence over learning rate.

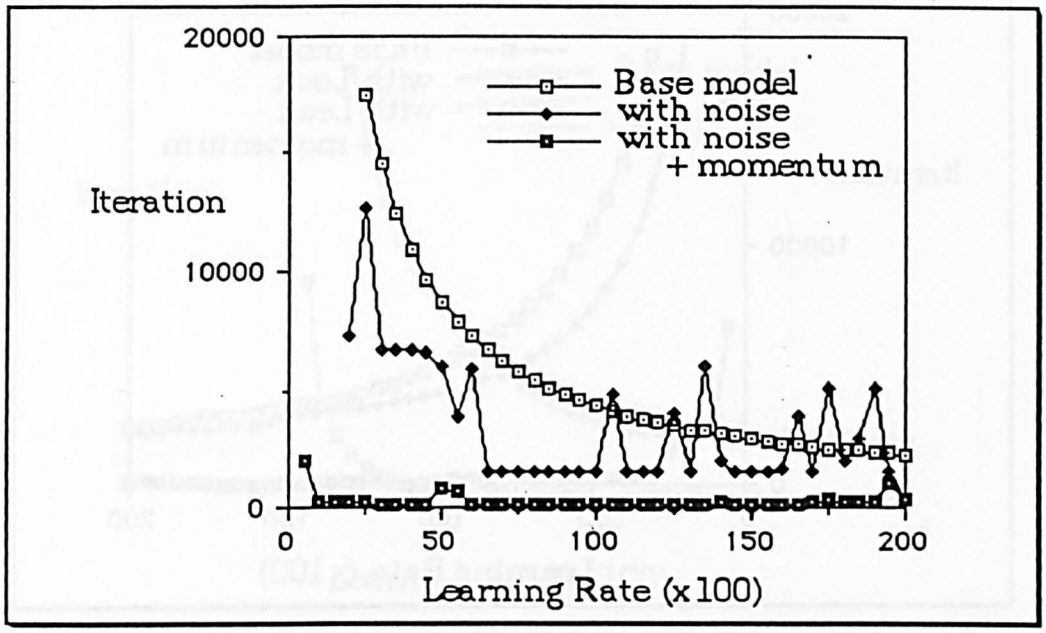

Figure 5.13. Back Propagation model with Random noise and Momentum added, using a single initial random seed setting (67). Number of iterations to convergence over learning rate.

Comparison of Base models with Leak activation and Momentum

Leak models show a small improvement on the ordinary back propagation model, but the greatest improvement is shown when momentum is added. Leak activation is unstable at higher learning rates in the seed(67) and seed(89) conditions, but shows great stability in the seed(43) conditions. Adding momentum improves the convergence speed and stability over a short range of learning rates in all models from approximately 0.25 to 2.0, where the behaviour of the model shows as an almost flat trace in figures 5.14 to 5.16.

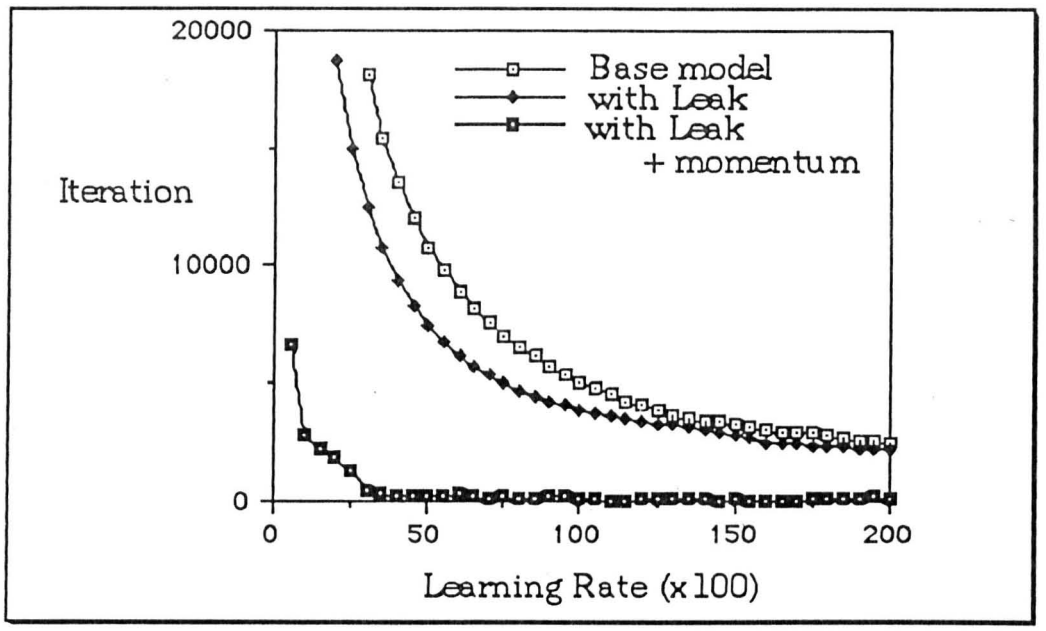

Figure 5.14. Back Propagation model with 'Leak' activation and Momentum added, using a single initial random seed setting (43). Number of iterations to convergence over learning rate.

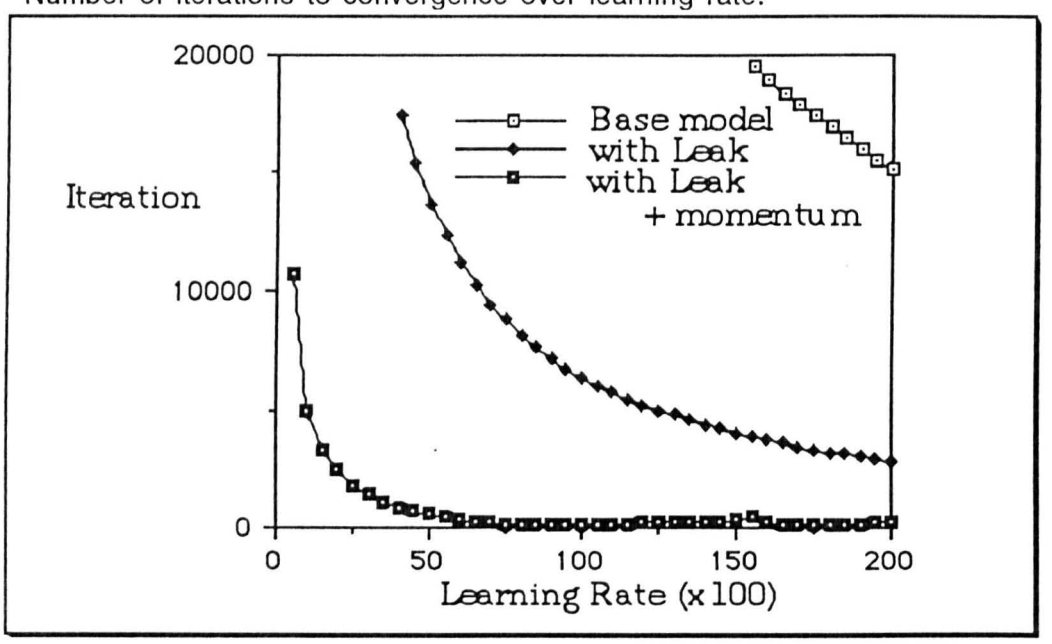

Figure 5.15. Back Propagation model with 'Leak' activation and Momentum added, using a single initial random seed setting (89). Number of iterations to convergence over learning rate.

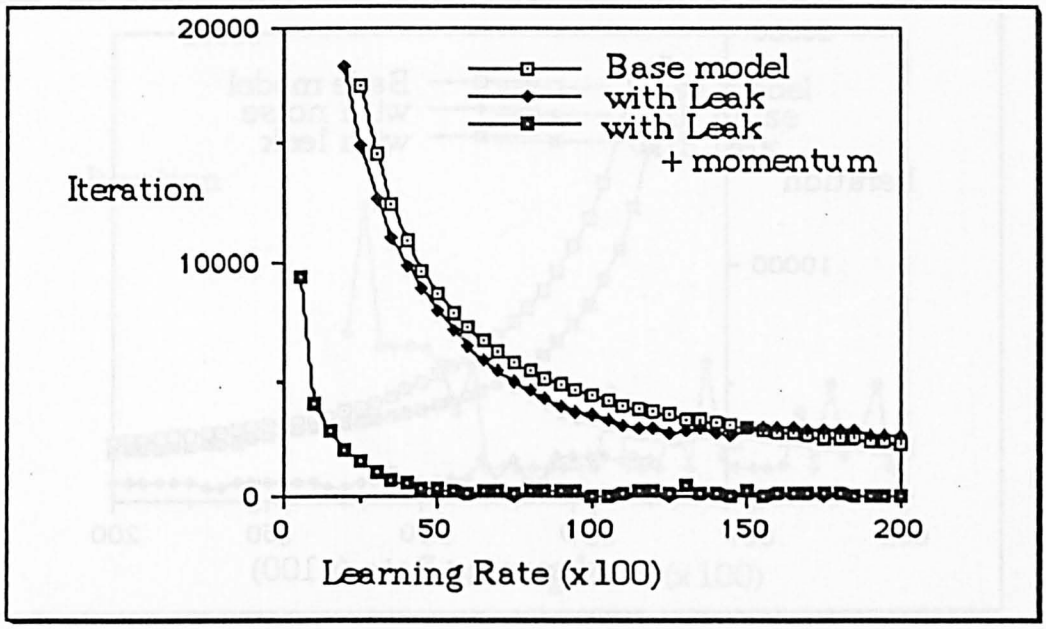

Figure 5.16. Back Propagation model with 'Leak' activation and Momentum added, using a single initial random seed setting (67). Number of iterations to convergence over learning rate.

Comparison of Base models with Noise and Leak activation

The following figures (figures 5.17 to 5.19) show that random noise is a more effective method of improving convergence speed and stability than using a bare 'Leakage' activation function, in networks without momentum functions. This result is shown in all initial condition settings. To reiterate, stability is good in the seed(43) and seed(89) cases, with a more haphazard 'stability' in the case of the seed(67) initial setting.

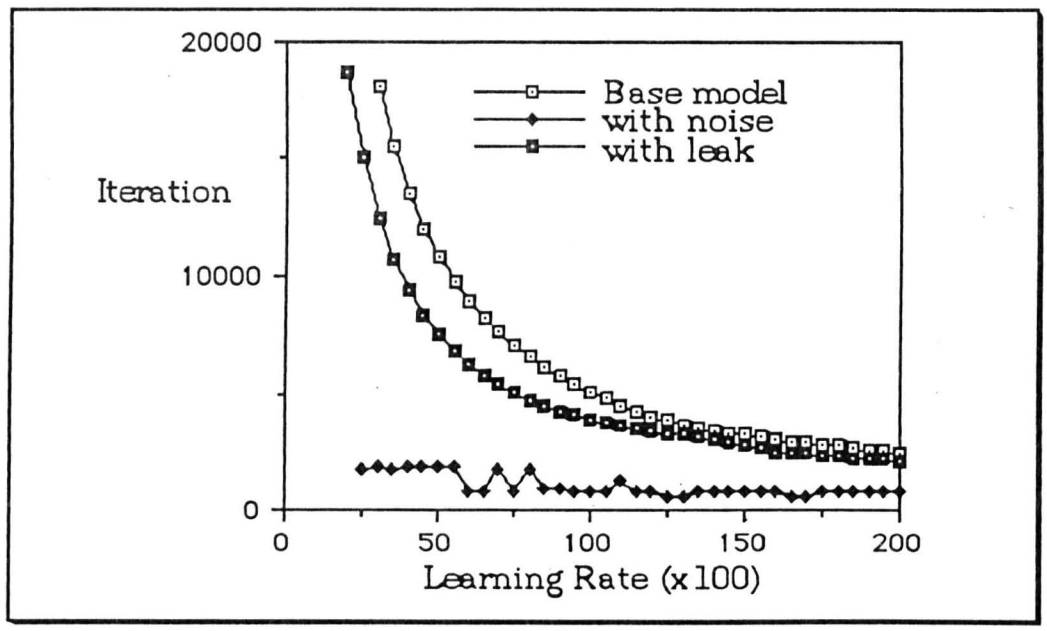

Figure 5.17. Back Propagation model comparing random noise and 'Leak' activation, using a single initial random seed setting (43). Number of iterations to convergence over learning rate.

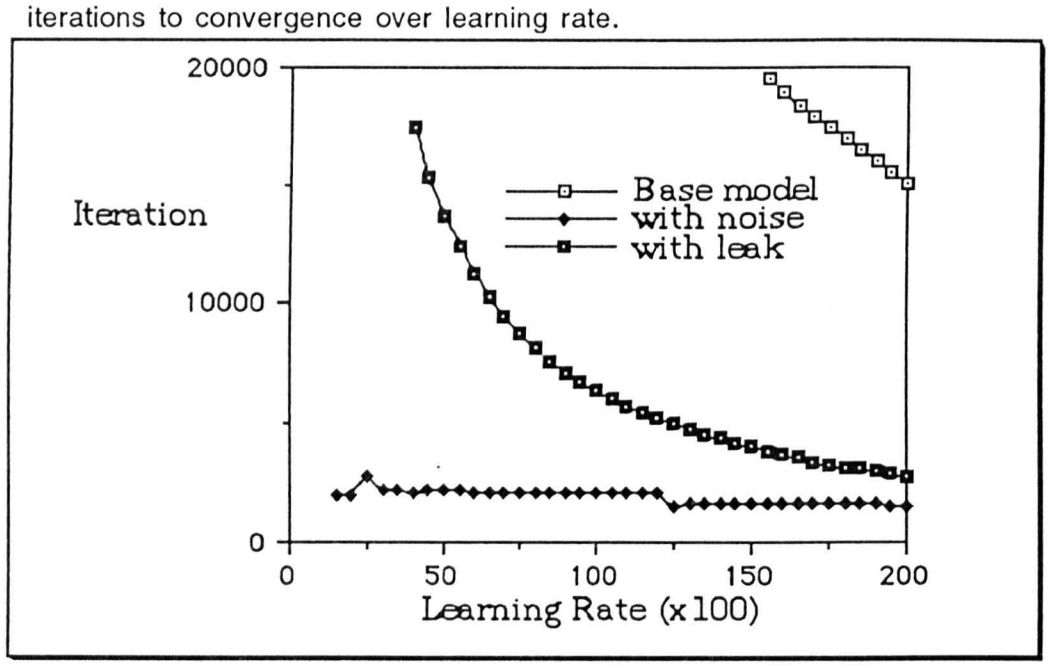

Figure 5.18. Back Propagation model comparing random noise and 'Leak' activation, using a single initial random seed setting (89). Number of iterations to convergence over learning rate.

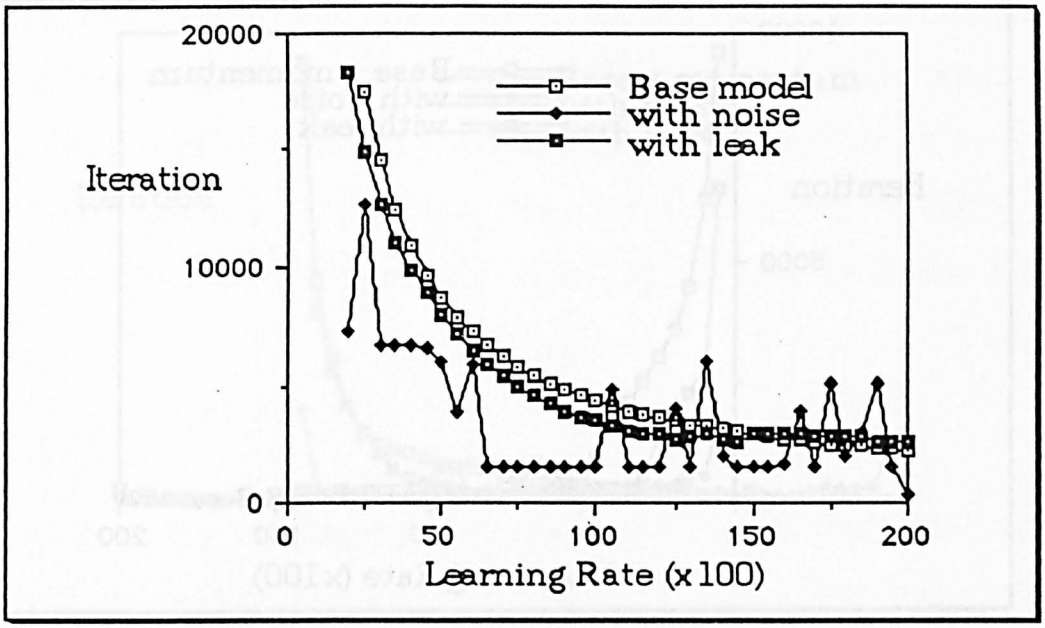

Figure 5.19. Back Propagation model comparing random noise and 'Leak' activation, using a single initial random seed setting (67). Number of iterations to convergence over learning rate.

Comparison of Momentum models with Noise and Leak activation

The results of this category are harder to distinguish on a gross level. In the seed(89) case, 'Leakage' activation seems to give a better account of itself than the base model with either momentum or momentum with noise. This is contradicted by the seed(43) model, but only by a small factor, giving a convergence disparity of approximately 150 iterations between the random noise and leak models, with random noise showing the better convergence rate in 'normal conditions'. In the seed(67) condition, the leakage and noise model results cross each other under 'normal conditions', but with a small advantage to the noise model.

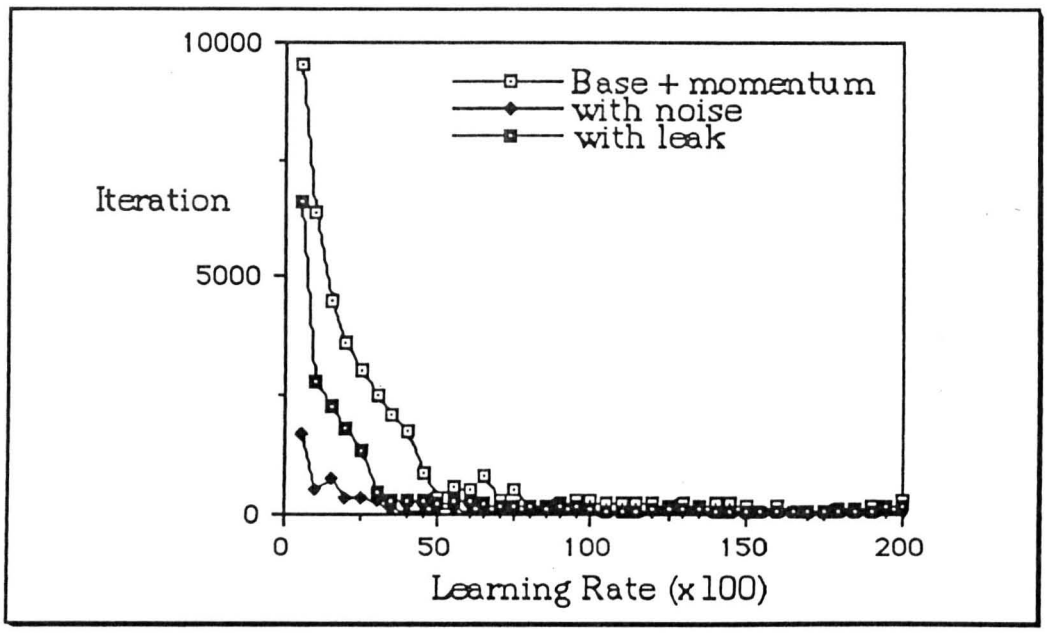

Figure 5.20. Back Propagation model comparing random noise and 'Leak' activation with added momentum, using a single initial random seed setting (43). Number of iterations to convergence over learning rate.

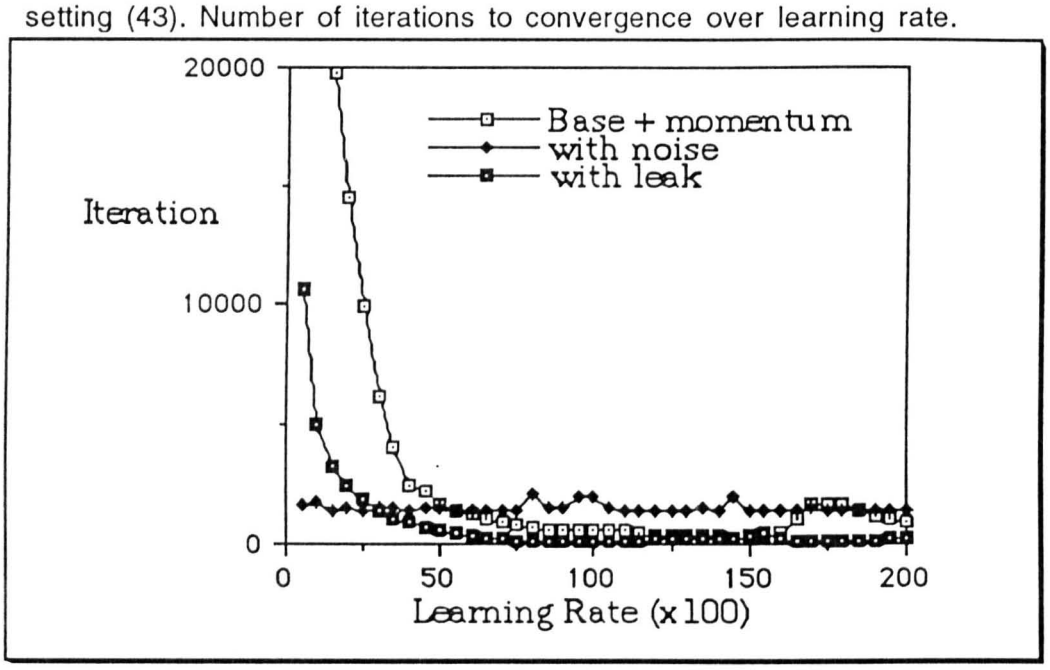

Figure 5.21. Back Propagation model comparing random noise and 'Leak' activation with added momentum, using a single initial random seed setting (89). Number of iterations to convergence over learning rate.

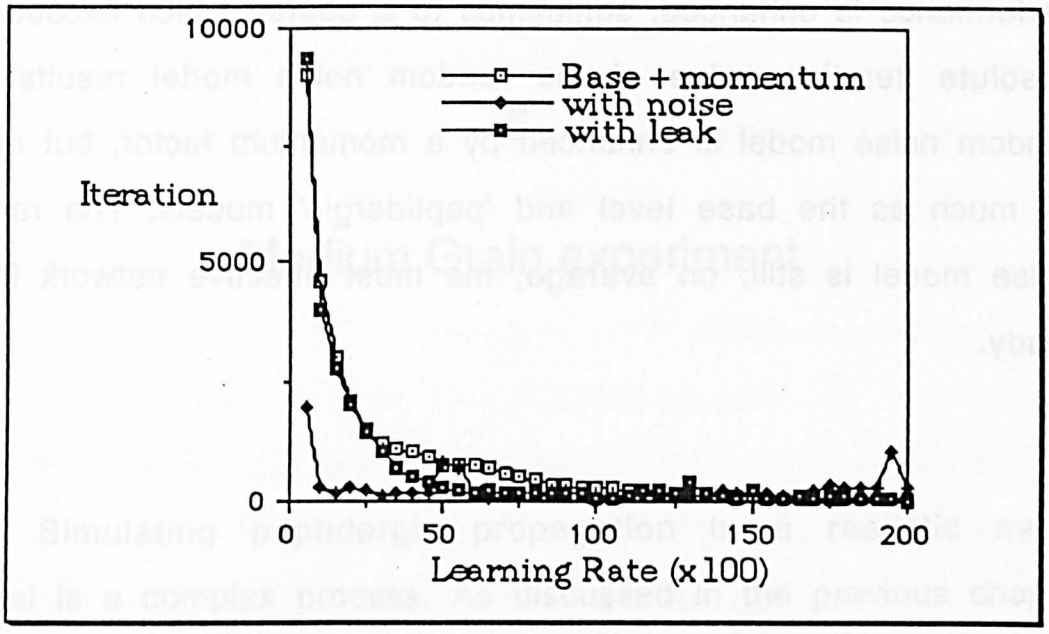

Figure 5.22. Back Propagation model comparing random noise and 'Leak' activation with added momentum, using a single initial random seed setting (67). Number of iterations to convergence over learning rate.

Summary

The implications of these results are discussed in detail in chapter 7, where they will be compared to the results of the following chapter. In summary, the results of the Back Propagation model given in this chapter, show that under conditions of random noise, the network learns more effectively (ie. at a faster rate) than in the base level model. In a further modified network model incorporating a possible analogue of peptidergic activation, learning rates are marginally better than the unmodified network, but not as good as the network under random noise conditions.

Adding momentum to each of the networks makes a great difference to their behaviour. Base level networks perform at a much improved level, while the 'peptidergic activation' model

performance is enhanced, sometimes to a degree which exceeds the absolute iteration value of the random noise model results. The random noise model is enhanced by a momentum factor, but not by as much as the base level and 'peptidergic' models. The random noise model is still, on average, the most effective network in the study.

6

Medium Grain experiment

Simulating peptidergic propagation in a realistic neural model is a complex process. As discussed in the previous chapter, the inclusion of a peptidergic factor in neural models requires that the network be placed in three dimensions, with the means available for calculating the peptidergic component reaching a particular point in three dimensional space from moment to moment. The complexity of this operation grows with the size of the network. While this task is not so daunting in a simplistic 'nodal' representation, a proper rendition of a peptidergic network, at even a medium grain size, requires that the peptidergic factor be calculated at *all points* on a cell surface, or at least, at all ion channel positions likely to be affected by peptides. This is a ridiculously complex expectation for any model, which no modelling package has yet provided, even in ordinary transmitter conditions. (But the case for ordinary transmitters can be much simpler than for neuropeptides, a concept which will not be discussed here.) A more practical method involves the averaging of peptidergic concentrations over a reasonably small area of cell surface, which should give a fair degree of accuracy in obtained results.

In this study, these ideals have been heavily compromised. In order to reduce complexity, and computational time, the model used has only two 'neurons'. Using only two neurons allows the peptidergic delay and concentration factors to be 'hard-wired' into the model at the outset of the experiment. Such 'hard-wiring' means that, in fact, only one point *per cell* is used in the calculation of the peptidergic concentration. Even so, the complexity of the modelled system requires several tens of seconds to simulate half a second of 'neural time'.

This study makes no attempt to investigate the possibilities of inhibitory peptidergic transmission, or of the properties of randomly injected noise. The main reason for the omission of a treatment of inhibitory transmission is that it would be expected, in a 'positive–signal' circuit, that inhibitory peptidergic transmission would merely delete events in the target cell, if the peptide strength and delay parameters were set at an effective value. While this would be a valuable finding, it is largely a complementary process to the use of peptidergic excitation in a 'positive–signal' circuit, which is what this experiment is intended to study. In the case of random noise injection, it is not expected that random injection will produce patterned responses in the simulation. Again, while spurious spiking events would be important in a nervous system, the results of noise injection in this model are entirely predictable as an event somewhere along the time–line, which may or may not effect the behaviour of the connected cell, depending on the time of onset of the event. It is therefore deemed an unnecessary experiment.

6. Medium Grain Experiment

Model Design

The medium grain model used in the following experiments is an adaptation of a model supplied with the GENESIS neural simulator from the California Institute of Technology (Wilson & Bower, 1989). The GENESIS system is an electrical equivalent computational modelling system, which models at the level of membrane and ion channel conductance and capacitance, with current sources acting as ion reservoirs, incorporating analogues of neurotransmitter time constants. Appendix E gives a description of the software package and its specifics of use in the following experiments.

The model used, known as the "MultiCell" simulation, consists of a pair of neurons represented at a medium level of detail as in (Wilson, 1989 - a paper provided with the GENESIS modelling package). Each cell has only two compartments, which represent the soma and dendritic arbour. The cells are connected in a feedback arrangement (Figure 6.1). The first cell provides an excitatory connection to the second cell, which feeds back via an inhibitory connection. All connections are simulated by an axonic delay with realistic conductance parameters and a weighting system.

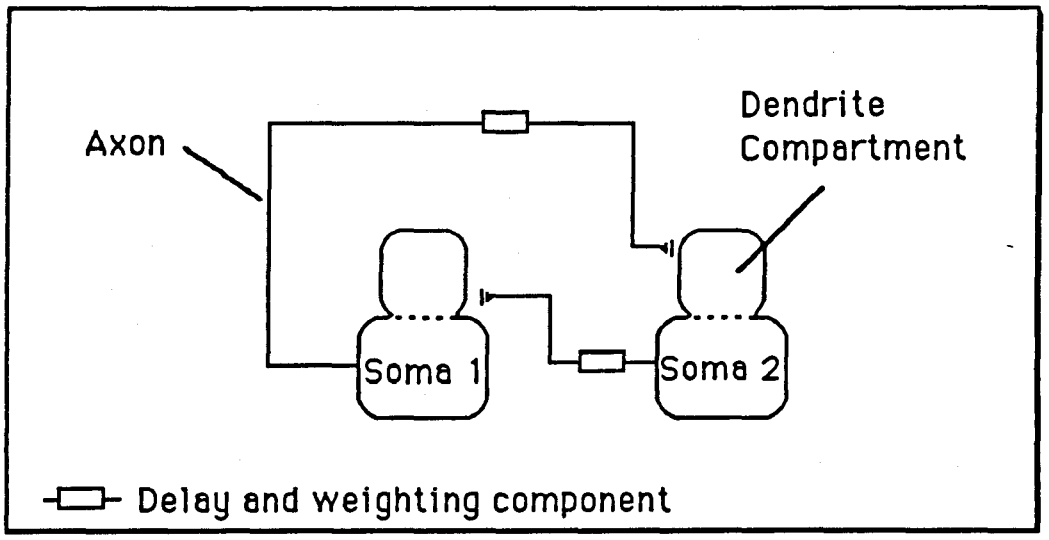

Figure 6.1 Two neuron feedback system supplied with the GENESIS simulation package.

Architecture and input/output behaviour of the model

Each cell of the model is represented as two compartments, shown in figure 6.1. The dendritic compartment carries the active Sodium and Potassium channels which are activated by changes in synaptic conditions, as well as incorporating conductance channels for resting potential and membrane leakage. The somatic compartment contains voltage controlled (Hodgkin–Huxley) Sodium and Potassium channels, which are triggered by changes in potential entering from the dendritic compartment. (see figure 6.2 and 6.3 for electrical characteristics) The apparent voltage in this compartment triggers the attached axonal element, which is set to produce a spike event at a particular potential. The spike event reaches the target compartment after a preset delay, and after being weighted by a preset weight. The weighting of the synaptic connection is a fixed parameter which is scaled by a maximum

conductance parameter to produce the final output of the synapse. The output is tempered by a function controlling the time course of the conductance changes at the synapse. This is composed of two fixed parameters which impose an exponential rise and fall on the output.

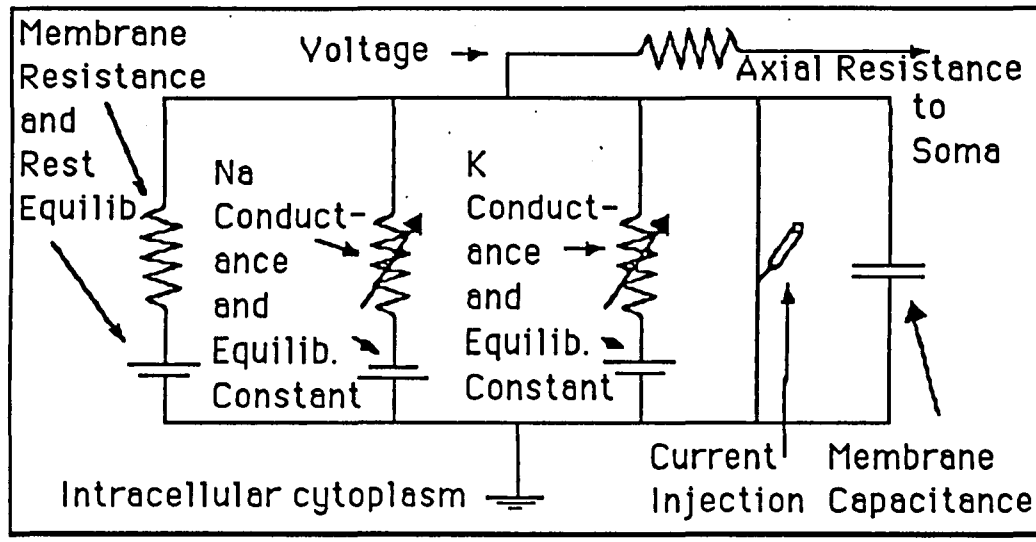

Figure 6.2. Electrical characteristics of the dendritic compartment in the two neuron model.

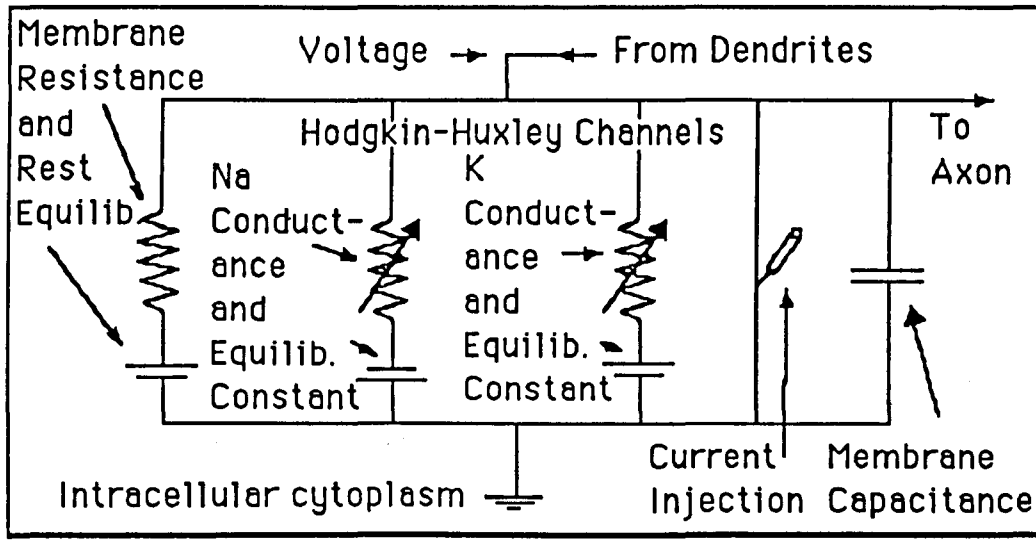

Figure 6.3. Electrical characteristics of the somatic compartment in the two neuron model.

The cells are arranged such that the first cell – the cell with an outgoing excitatory axon and an incoming inhibitory axon (known

as cell 1, hereafter) – is excited by a constant current injection. After a summation delay, this produces a spike event in the axon which is synaptically connected to cell 2. (The mechanics of these events follows the procedure outlined for real neurons in chapter 3.) Cell 2 is excited by the event and produces a spike event in its own axon, which is connected via inhibitory channels to cell 1. This may inhibit cell 1 if the timing of the spike event is coincident with the summation of current injection in cell 1. (The normal behaviour of both cells is shown in figure 6.4 and 6.5)

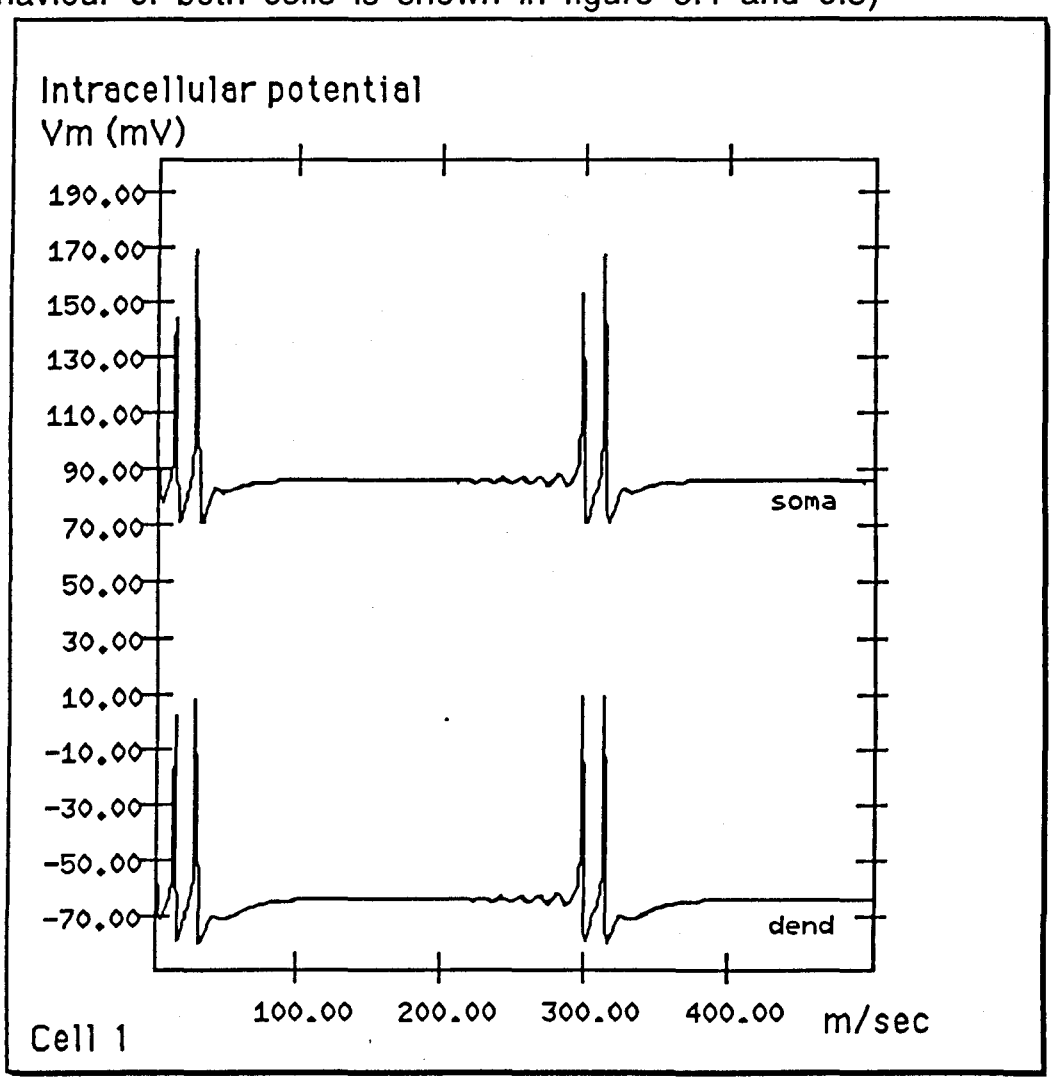

Figure 6.4. Normal behaviour of cell 1 in the model, as supplied with the GENESIS software.

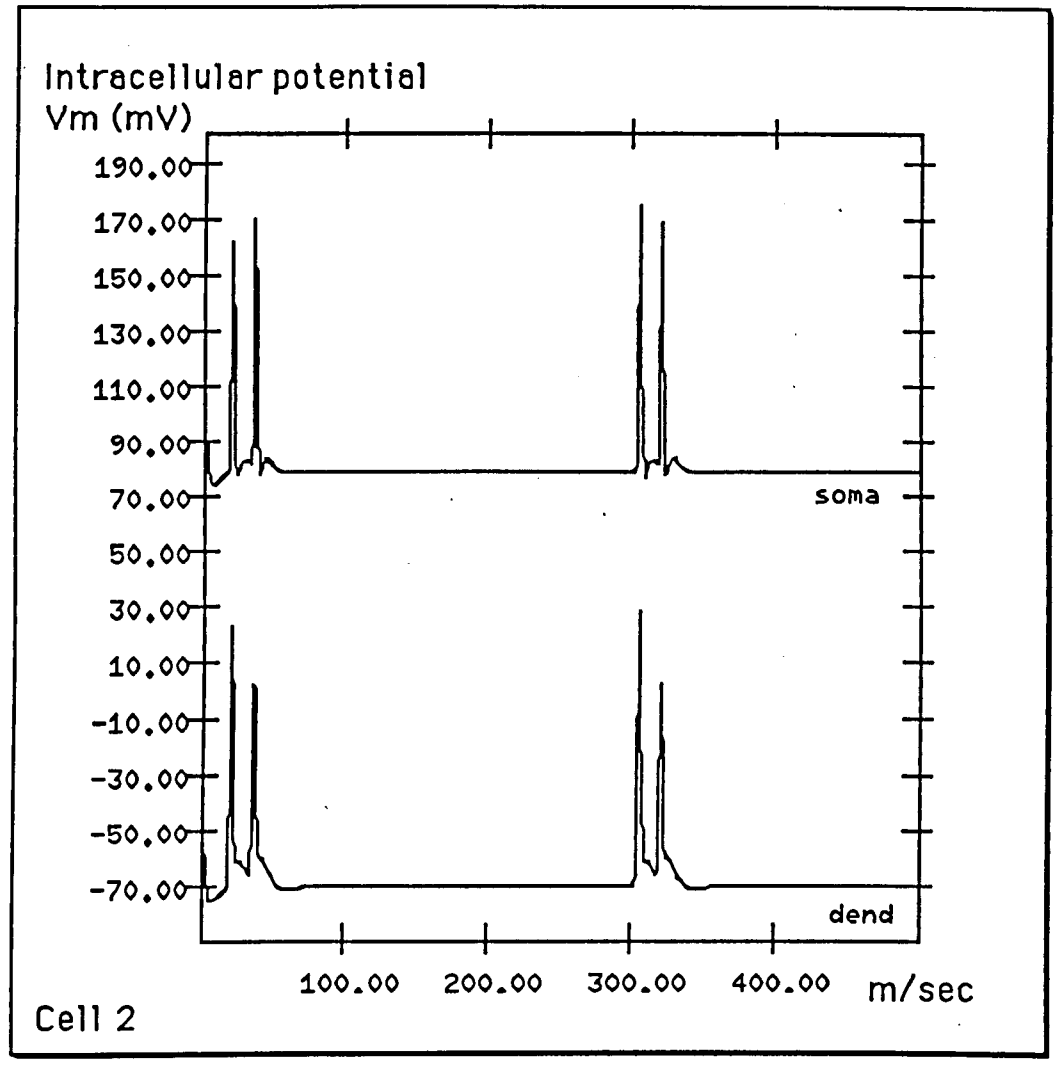

Figure 6.5. Normal behaviour of cell 2 in the model, as supplied with the GENESIS software.

The parameters used for the electrical characteristics of the cell membranes and ion channels are fixed in the parameter input to the GENESIS program. All the electrical parameters are taken from experimental papers which measure the conductances of specific ion channels, resistance and capacitance of cell membranes and ion channel densities. The sources for these data are catalogued in other simulations provided with the GENESIS package. Some of the model parameters are arbitrary. These are generally variables such as synaptic weight, axonal delay and spike generation thresholds.

6. Medium Grain Experiment

Parameters relevant to this study are the synaptic weights, axonal delays and synaptic time constants, as these can be varied in a modified model to simulate a simplistic version of peptidergic transmission. Table 6.1 shows the values used in the simulations of figures 6.4 and 6.5.

	Excitatory connection (Cell 1 -> Cell 2)	Inhibitory connection (Cell 2 -> Cell 1)
Axonal delay	5 msec	5 msec
Synaptic weight	30	300
synaptic time constants (tau1/tau2) (msec)	3/3	10/10

Table 6.1 Parameters relevant to this study, as used in the unmodified MultiCell simulation.

The Experiment

The experiment modifies the existing two neuron system to incorporate a simplistic peptidergic component. This is represented by a further pair of axons with modified delay and weighting parameters, connecting each of the two cells as in figure 6.6. (See Appendix E for the GENESIS script modifications required.) This is only just feasible in a two neuron model, as complex three dimensional calculations of time and diffusion/concentration coefficients can be avoided by pre-setting these characteristics; assuming that the ion channels responsible for neuropeptidergic activity cover a small area of the dendritic membrane. This is not a particularly useful assumption, but it does

negate the need for complex run-time calculation of these variables.

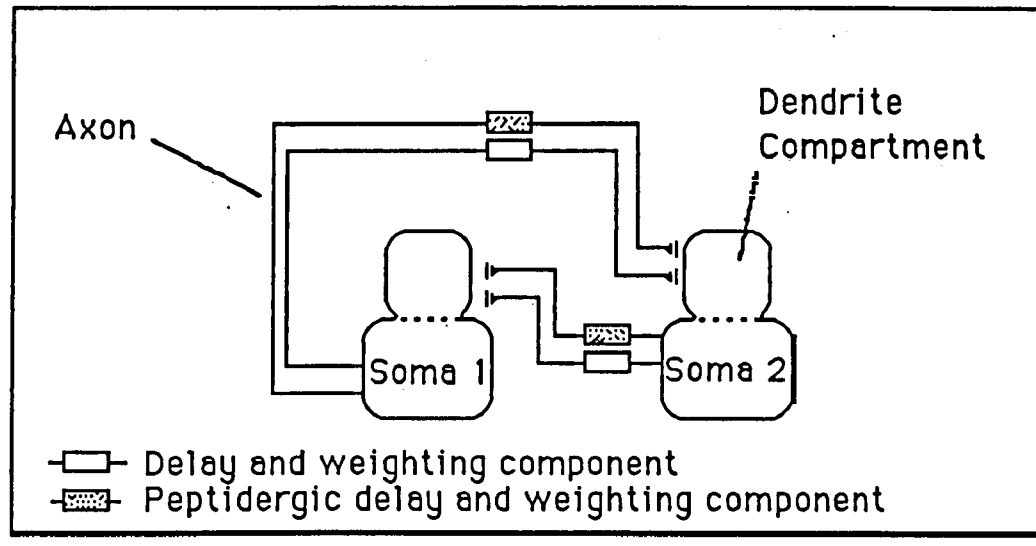

Figure 6.6. Modified MultiCell simulation. The additional axons simulate a peptidergic input to the dendritic compartment of both cells.

The modified connections should attempt to simulate a possible peptidergic signal arriving at a neighbouring neuron. Such an effect can be produced by lowering the synaptic weight to a fraction of that for normal transmission, increasing the transmission delay between signal source and reception, and increasing the time course of the effect on the target cell, allowing for a longer effect caused by slower removal of peptide transmitters. These factors are represented by the parameters displayed in table 6.1.

The major concern in this modification is the application of realistic values to these parameters. Representative data was not available, so it was decided to use a set of values relative to the existing values used in the normal transmission simulation, and to modify these values during different simulations to test the behaviour of the model.

6. Medium Grain Experiment

Initially, it was decided to set the value of the synaptic weight to a tenth of the lowest value used in the existing model, and then to vary the weight to assess the effects of weight on the model. The axonal delay was set to 6 times longer than the existing axonal delay and varied in a similar manner. The synaptic time course parameters were set to twice the value of the longest time course in the existing model and then varied. These values are shown in table 6.2.

	Existing Excitatory connection (Cell 1 -> Cell 2)	Existing Inhibitory connection. (Cell 2 -> Cell 1)	New Peptidergic connection (Cell 1 = Cell 2)
Axonal delay	5 msec	5 msec	30 msec
Synaptic weight	30	300	3
synaptic time constants (tau1/tau2) (msec)	3/3	10/10	20/20

Table 6.2 Modified MultiCell simulation using modified axonal connections to represent peptidergic transmission.

Results

Base model

The Base model is defined as the model with the settings shown in table 6.2. Running this simulation has no apparent effect on the behaviour of cell 1, but a marked effect on the behaviour of cell 2. (as shown in figure 6.7) Cell 2 produces the normal spikes associated with stimulation by cell 1, but then produces three more spikes, as the peptidergic emulation produces an enhanced excitation state. Theoretically, the enhanced excitation of cell 2 should produce a consequent inhibition of cell 1, but as the summated current injection into cell 1 is insufficient to fire cell 1 again, after the first spike train, no reduction of activity is seen in cell 1. Note that the addition of the second part of axons has increased the number of spiking events from two in the unmodified model, to three in the modified model.

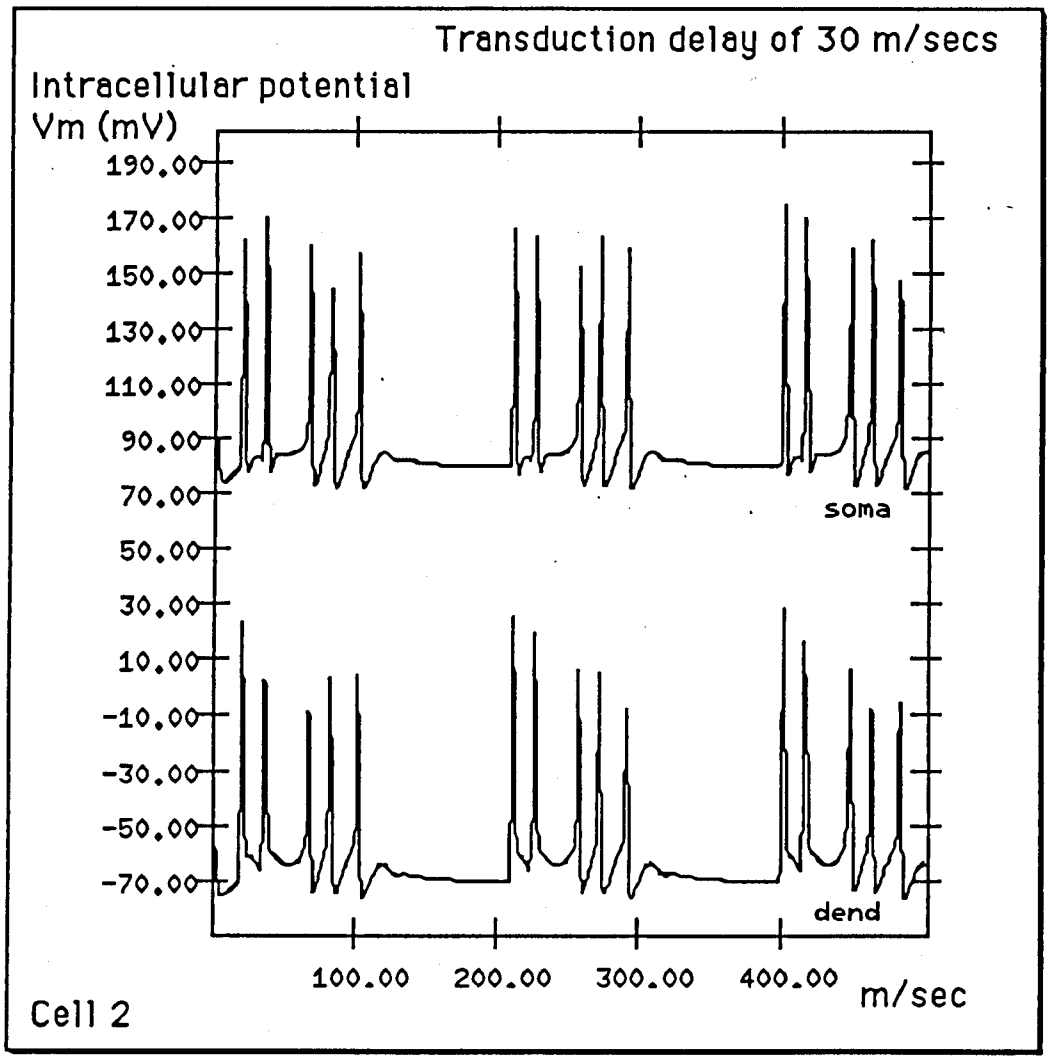

Figure 6.7. Using the values in table 6.2 for the modified axonal connections produces an additional spike train in the peptidergic transmission model of cell 2.

Variations in axonal delay

Increasing the axonal delay setting has the effect of lengthening the time between the primary spike duo and the following spike train. This occurs for all delays upwards of 30 msecs. Below 30 msecs the secondary spike train is not produced, resulting in a primary spike train with a potentiation of excited activity within cell 2. (Figure 6.8)

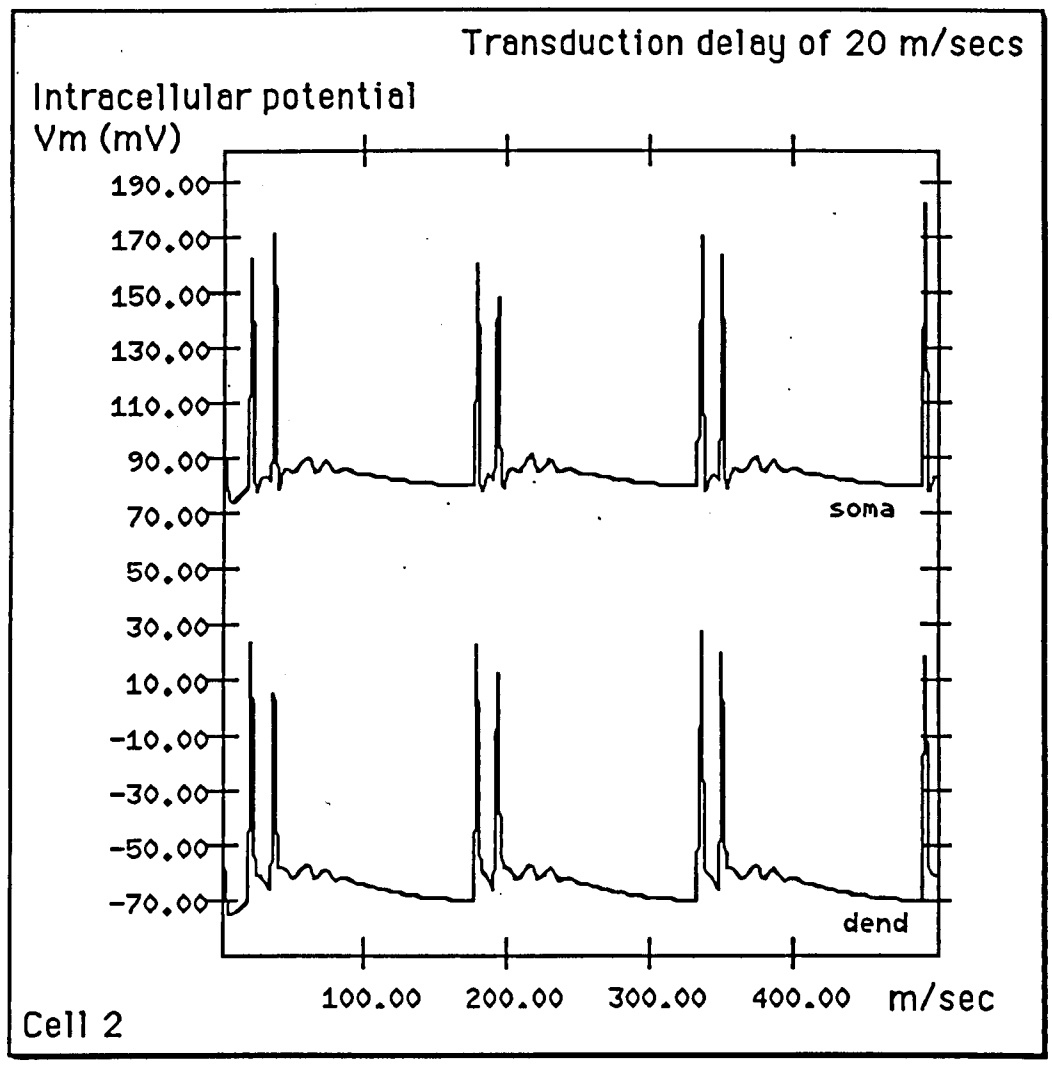

Figure 6.8. Reducing the axonal delay to 20 msecs for the modified axonal connections produces an additional excitation, but no spike train in the peptidergic transmission model of cell 2.

As the delay in the axonal link becomes longer, approaching the onset of a new spiking event, the inhibition from cell 2 begins to show an effect on the behaviour of cell 1. Finally, at a delay corresponding to the onset of spiking events in cell 1, the inhibition from cell 2 cuts off the response from cell 1, producing a different pattern of firing from both cells 1 and 2 (Figures 6.9 and 6.10)

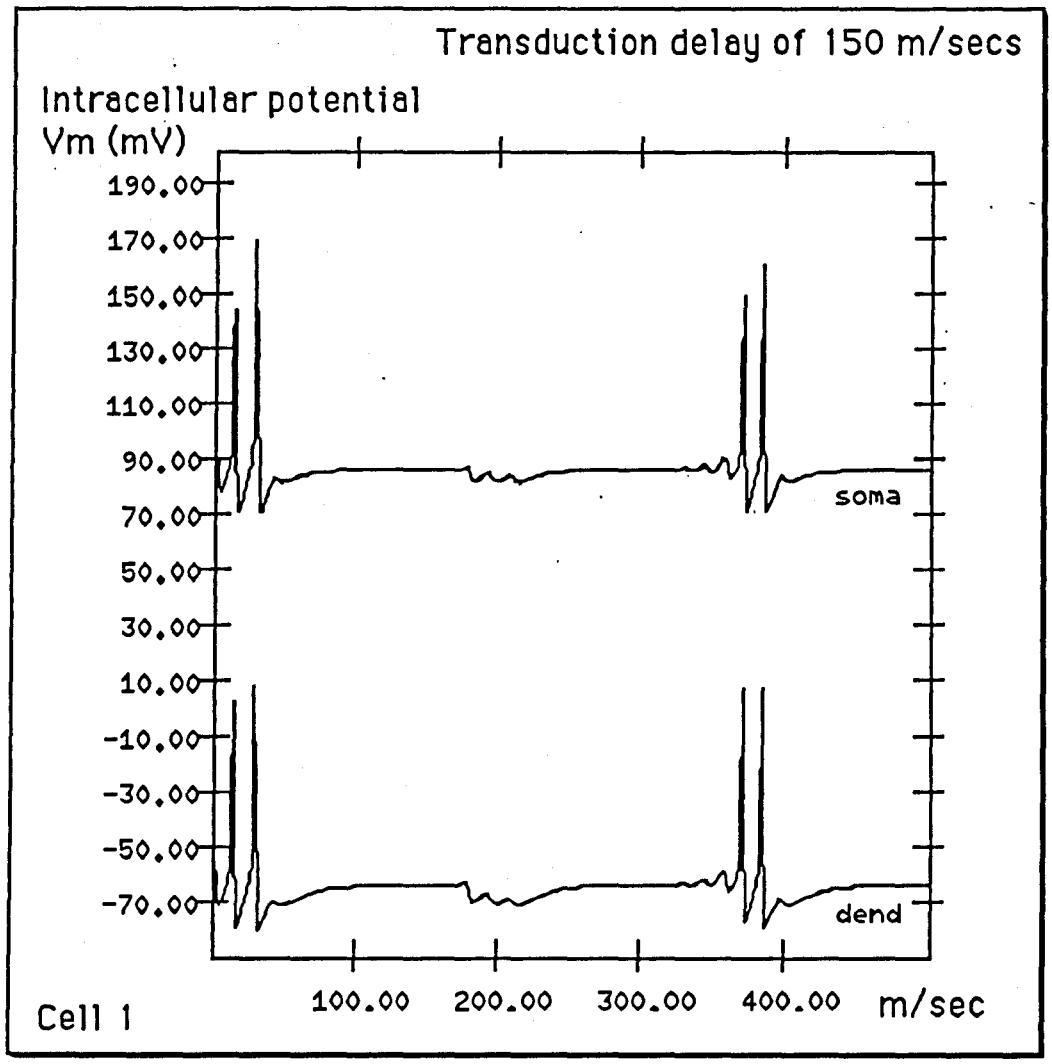

Figure 6.9. A delay of 150 msecs imposes inhibition on cell 1, when a spiking event is imminent.

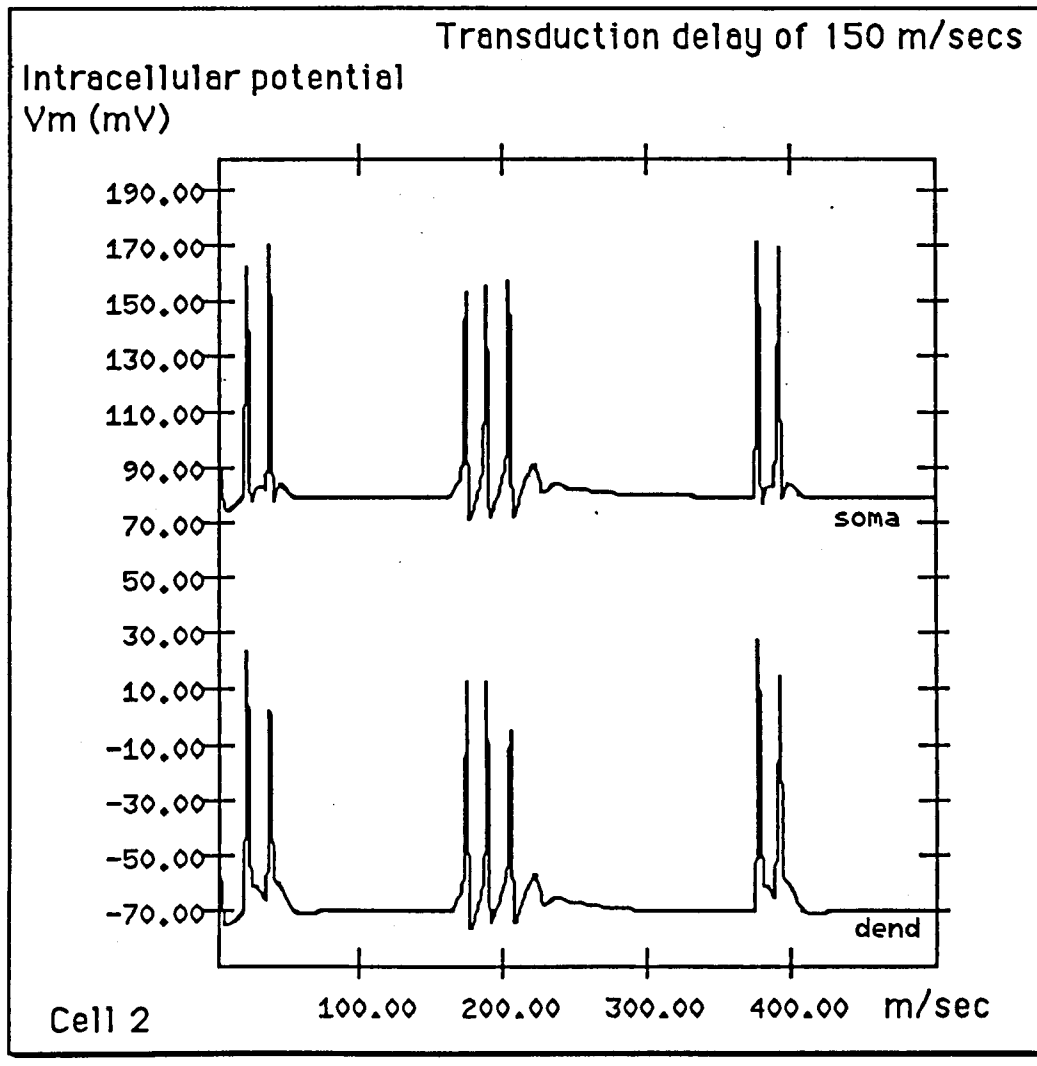

Figure 6.10. A delay of 150 msecs causes inhibition on cell 1 as a result of the excitation in cell 2.

Variations in synaptic weight

Variations in synaptic weight have a pronounced effect on the output of the peptidergic transmission system, due to the length of the time constant associated with the peptidergic stimulation. Setting the synaptic weight to values of less than 3 produces no secondary spiking events in either cell. At a value of 2, a small potentiation effect appears to take place in cell 2, without a corresponding spiking event. Setting the weight to values greater

than 3 produces a larger number of spikes in the secondary spike train, until, at a value of about 10, the secondary spike train in cell 2 appears to go into tetanus, giving a continual spike train in cell 2 and the extinction of any responses in cell 1.

Variations in synaptic time constant

Variations in the synaptic time constant have a similar effect to the changes in synaptic weight shown above. Setting the time constant to values lower than 20 produce a smaller number of spikes in the secondary spike train of cell 2. This has little effect on cell 1. Setting the time constant to higher values increases the number of spikes in the secondary spike train, until at much higher values the response of cell 2 goes into tetanic behaviour with a corresponding extinction of responses in cell 1.

Summary

The results of the simple compartmental model in this chapter, show that a peptidergic link may have a neurally reasonable effect on signal transmission quality, if the parameters of peptidergic strength, delay and time constants are within the bounds specified below. In general, varying the weighting strength parameter varies the number of spikes in a 'peptidergic' spike train. Varying the time constant of the synaptic 'peptidergic' effect produces similar results to the variation of weighting strength. Varying the transmission delay of the 'peptidergic' effect

produces a change in the timing of the response of the target cell to the 'peptidergic' stimulation.

Changes in transmission quality, from the characteristics of the base model, require that the lower bounds of signal strength be no less than 10% of the classical transmission strength, if spiking activity is to be invoked by the peptidergic component of cell activation. The lowest delay which produces spiking under the same strength conditions is 30 milliseconds, which allows the refractory period (10 milliseconds) of the spiking module to elapse, and the build up of charge within the soma to be unaffected by the latent hyperpolarisation of the Potassium channels of the target cell. The time constant lower bounds are defined by similar criteria when delays are low, but are also determined by the length of time required for a critical charge to build up within the soma of the target cell. When the time constant is low, fewer 'peptidergic' spikes are seen in the output of the cell.

The upper bounds of the above constants are imposed by qualitative changes in the output of the target cell. Obviously a stronger 'peptidergic' signal and/or a greater time constant invokes an enhanced spiking behaviour. The upper limits for these parameters are interdependent, and could be fixed at the point where the target cell exhibits tetanic behaviour. Varying the reception delay of the 'peptidergic' signal imposes a different pattern of behaviour than the strength or time constant parameters. There are no obvious upper bounds on this parameter, except those imposed by diffusion calculation limits, as the effect of varying this parameter merely time-shifts the 'peptidergic' response spiking behaviour.

PAGE HAS NO CONTENT

7

Discussion

The results in the previous two chapters reflect a positive difference in outcome of the operation of two types of neural network, under the conditions of random noise and what can be termed a weak, directed signal (the 'peptidergic' signal) or modulum, as proposed in chapter four. The direction and quality of the difference in the operation of the models under these conditions, is the subject of this chapter.

It is important to attempt a detailed analysis of the mechanisms underlying the operation of each category of signal propagation used in the preceding chapters, although the results of the experiments do not give an indication of the actual mechanisms, but rather suggest that the mechanisms may be different. The questions which will be attempted in this chapter are those relating to the theoretical operation of both the random and 'peptidergic' functions. The most interesting of these questions are those concerning the possible differences and parallels between the two types of network modulation; whether they act along similar lines or are part of a different mechanism.

The primary aim of this work is to assess the contributions of random noise and 'peptidergic' signal to network performance, with a view to extending neural computation theory into these

domains. The results of the previous chapters have shown that these two factors behaviourally affect two different types of network, and should therefore be incorporated into network theory.

Findings of Back-Propagation modelling

The Back–Propagation models show that a 'peptidergic leakage' activation inclusion produces a learning enhancement on a similar scale to that of random noise. The characteristic graphs of this enhancement are different, however, in that the random noise models tend to produce a less incremental change in convergence rates than the other categories of model. Random noise produces plateau–like regions (see figures' 5.11 onwards) with abrupt changes in gradient between learning rate settings. Base models and Peptidergic leak models, on the other hand, produce low second order gradient changes between learning rate increments. These shallow gradients indicate that the network is more stable than the random noise model.

In a comparison of the methods used in this study, the random noise model presents little correlation between initial setting parameters, indicating that initial 'synaptic' weights are extremely important in the subsequent behaviour of the network. In contrast, the 'peptidergic' leak model produces a greater correlation across initial setting modes than the base models. (See Table 7.1) This indicates that the result obtained is less dependent on the initial state of the network than even the ordinary back–propagation algorithm. This implies that the 'peptidergic' leak model is a slightly stabler model than the base model.

	seed 43	seed 89	seed 67
seed 43	1.000		
seed 89	0.992	1.000	
seed 67	0.980	0.994	1.000

Back Propagation only

	seed 43	seed 89	seed 67
seed 43	1.000		
seed 89	0.947	1.000	
seed 67	0.953	0.923	1.000

Back Propagation with momentum

	seed 43	seed 89	seed 67
seed 43	1.000		
seed 89	0.086	1.000	
seed 67	0.720	-0.017	1.000

Back Propagation with Noise & Momentum

	seed 43	seed 89	seed 67
seed 43	1.000		
seed 89	0.992	1.000	
seed 67	0.995	0.997	1.000

Back Propagation with 'Peptidergic' Leak & Momentum

Table 7.1 A comparison of the correlation within conditions across random seeding categories. (Learning rates 0.05 to 2.00)

The high correlation shown for the 'peptidergic' leakage function in the above table implies that perhaps a stronger activation cross-correlation method is being used in the 'peptidergic' leakage model, which is absent in the random noise

model. From this data, we can conclude that the methods of convergence occurring in the random noise model and the leakage model are quite different.

The results of the random noise model give the appearance that the random noise model uses a "lucky" paradigm, which works in this study particularly because of the limited range of activation and weight spaces allowed in the computational model of the network. Clamping the weight values of the network to values of ±20.0, means that the range of noise allowed is a considerable proportion (up to 6%) of the weight space. In a network with a larger range of activation and weight space, or requiring a more complex transformation, it is likely that the random noise model would produce less useful results. However, the injection of random noise may also serve to 'broaden' the activation range of the network, by allowing access to particular activation states within one iteration, which would otherwise take several iterations to achieve, thus cutting down on computational time.

The 'peptidergic' leakage model appears to use a method of activation correlation within the network, which may be similar to methods used in Kohonen's self–organising map models (see chapter 2). In this case a weak spreading activation function is occurring within a local area, with an even weaker spreading activation function occurring on a global scale. The overriding method in use, however, is still the back–propagation algorithm that must, in some way, benefit from the additional activation obtained from the 'peptidergic' leakage function. This may be due to the extension of the nodal activation range, with the added activation provided by the leakage function.

A further difference existing between the two methods is the input characteristics of each method. The injection of random noise is theoretically a stochastic process with a final net input of zero, given that the noise parameter is set up as an even distribution about the zero point. This contrasts with the 'peptidergic' leakage model, which has a definite net input into the network over its operation cycle, which in this study was usually positive, with an approximate scaling over the random noise input of 10^4. This additional input strengthens the case for the theory that leak activation extends the range of activation of the network.

The effect of momentum on the operation of these models would be to keep the gradient descent strategy on course, as it was originally intended to do. This is achieved through the 'history' mechanism of the momentum function which adds 90% of the previous error function to the upcoming weighting event. In the case of random noise, this would enhance the effect of continuously polar series, and negate the effect of opposite random assignments. Peptidergic leakage on the other hand, will produce a smoothly varying activation addition, generally at much lower absolute values (ie. less extreme values) than that produced by random noise. The action of momentum will not enhance the operation of the network to such a degree as that achieved in random noise models. In addition, changes of activation polarity (where they occur) would be highly smoothed by the momentum term.

The contextually relevant findings are that the peptidergic leakage activation appears to act as cross-association function. This leakage appears to *extend* the activation range of already activated nodes, allowing the back-propagation algorithm to

converge at a faster rate. Noise appears to *broaden* the activation scale, by allowing a larger number of available activation points within a network. Noise models give a plateau-like characteristic curve, whereas base-level and leak curves look more 'inverse-exponential' in nature. Leak curves (of all seeds) appear close together in graphs, with a high correlation across all initial conditions, indicating that initial conditions are less important than in models using noise, leading to stabler networks.

Findings of Medium Grain modelling

The GENESIS model uses only a very limited 'peptidergic' simulation, which accounts for only point–to–point peptidergic propagation from the synapse to the dendritic processes of the target cell. The viability of this model rests on the fact that it is only a two–neuron system, so that inter–cell distances are not complex, and peptidergic diffusion can be modelled as a lengthened period of contact with the target cell. The exact concentrations of peptides across the cell cannot be simulated at present, without much additional programming work.

The results of this study show that the 'peptidergic' effect (as defined in this study) produces a significant behavioural change in the output of the two cells. At the lowest level of operation, the peptidergic component produces a potentiation effect on the output of the second cell, whilst at the highest level it can produce sustained oscillations. At (for want of a better description) reasonable biological levels, the peptidergic component lengthens

the spike train of a stimulus from the source cell, and can cancel spikes in a source cell via feedback connections, if the onset delay is correlated with the normal spiking behaviour.

It appears that the peptidergic component in this study produces a lengthening of the apparent activation of a target cell, at least to the point where potentiation occurs (see figure 6.8). In more complex systems this would be available to modulate signals arriving from other components of the system, producing an effect greater than that with simple classical transmission events. Extending the potentiation effect to a cluster of cells, would allow peptidergic activation to maintain a general level of excitation in the cluster, for a specific time after the onset of a peptidergic signal. This may act as a cluster 'biasing' system – a 'priming' or 'expectation' system.

In a neural system with a greater peptidergic component, to a degree which prolongs the activation of a cell (see figure 6.7), we can expect the peptidergic component to modify the behaviour of the network to a considerable extent; single activation signals will lengthen and the length of the propagation chain will, unless modified by other events, also lengthen.

Discussion of both models

One of the aims of this study was to decide whether random noise and peptidergic activation, which was originally described as a form of 'directed noise' has any ability to influence the outcome of neural network operation. In Back–Propagation networks, noise and 'peptidergic' leakage activation seem to have a

positive effect – and it seems that the mechanism of operation is different for the two activation methods tested. In the medium grain model, again, peptidergic operation produces an additional effect on the excitation of the cell; but can we say that this mechanism is functionally similar to the mechanism involved in the back–propagation model?

It was stated that the mechanism of operation of the peptidergic factor in the back–propagation model was thought to be akin to an extension of the activation range of each node, which allows a larger signal to be propagated, correlated with the signals of the surrounding nodes. In the medium–grain model, the effect of a peptidergic factor is an extension of the signal length. This can be construed as an extension of the activation range of a node; but only if it is possible to propagate a signal without the associated peptidergic signal, or with an attenuated peptidergic component. This was not tested in this study, but is a result of several studies of neuropeptides (Lundberg & Hökfelt, 1985; Horn and Dodd, 1985). We can conclude that the mechanism of action under both network models is very similar, given that a low–level model can be constructed with a variable peptidergic component.

This study has not only shown that excitatory peptidergic factors can lengthen the response of target cells, but has also shown that an inhibitory feedback connection of a peptidergic nature, can veto the firing (and thus signalling) of a source cell (see figure 6.9). This type of signalling falls into the category of an antagonistic neuropeptide (see Emson, 1985, and chapter 4) The veto action of a peptidergic connection is highly dependent on the distance between nodes, the period of peptidergic stimulation and the concentration of peptides at the receptors.

Implications for Signal/Noise dichotomy

The results of this study seem to show that neuropeptides may affect the components of a nervous system, if the assumptions made about the nature of a peptidergic factor are close to biological levels. In this case, it cannot be possible to assign definite boundaries to signal and noise in the nervous system. It is unlikely that such assignments are the current practice, but perhaps the subtler modulation brought about by low–level signals may now be studied in greater depth.

There are still some questions to ask about the role of noise in the nervous system. For example, noise may still be a factor in neural circuits, but when does 'noise' become a signal, and when does 'signal' become noise? It is unlikely that these events occur at exactly the same point along a chemical transmitter concentration gradient, for example.

It has been proposed earlier that it is possible that random noise broadens the number of activation points in a finite point system in artificial neural networks. This would mean that noise would become less effective as the number of activation points available grew; ie. a growth in either the size of the network or the number of digits in the mantissa, and moving towards an analogue implementation. This could mean that noise may not be useful as an operational enhancer (as in the Back–Propagation study) in the human brain, leading to the proposition that perhaps the human brain exhibits its degree of intelligence because we have exceeded some critical size for the reduction of the influence of noise on the

content of signal transmission within the brain. Does the complexity of the human nervous system mean that it is the first to overcome the *problem* of noise; ie. Is it possible that the Homo species are the first species with a large enough nervous system to ward off the adverse effects of an inherently noisy environment?

Concurrent propagation?

In chapter 4, it was argued that the action of neuropeptides could produce the mechanism for a system of concurrent propagation within a neural system. This proposal is still valid, as it has been demonstrated that neuropeptides may have some effects within a neural system. In chapter six, it was shown that the neuropeptidergic signal, at various parameter settings, was able to induce time-shifted spiking activity in a target neuron. Such a finding is one of the requirements for a concurrent propagation scheme. The results shown are ambiguous, however. The temporal difference of the spiking event in the medium-grain model is related to the setting of the peptidergic activity time-constant. If the time-constant is very low, no extra activity appears. If the time-constant is a little higher, the activity that appears may be seen to be coupled to the primary activation of the neuron – producing a mere *extension* of the excitation; which is not independent. The independence required for true concurrent propagation requires a parallel peptidergic and classical signal.

The major problems in interpreting the results as showing that concurrent propagation exists, are firstly, that many of the

experimental parameter settings were fabricated, due to lack of data, and secondly, that the neural model used was small, fabricated, and still a fairly coarse-grain representation, not subject to many of the processes which are present in the neural environment. One of the major factors required for demonstrating concurrent propagation is the independent release of both peptidergic and classical transmitters, a process which could not be studied in this model. A resolution of this question lies largely in future research within the biological sphere.

Future Research

The best possible direction for future research in this area, lies in the establishment of a multi-disciplinary project focussing on the aspects of neuropeptides which could be studied from two angles; biological and computational.

Computational modelling of peptidergic systems can be preferable to neurobiological investigation for a number of reasons. Investigating neuropeptidergic transmission requires that the relative distances between neural elements remain fixed, which is not easy to maintain in biological preparations. *In vivo* tests are difficult to execute in conditions requiring no cell displacement and continuity of the extracellular medium, and repeatable impalement of a single cell is difficult in *in vivo* conditions, across preparations. Recording from many cells at the same time is also difficult. Damage is also caused by impaling neural tissue, which can allow current leakage through the membrane, and leads to inaccurate results. Computational

modelling suffers from none of these drawbacks, and has many advantages. The major difficulty lies in obtaining accurate parameters for the model from neuroscience research. The establishment of a multi-disciplinary project should eliminate this limitation.

The execution of such a project would require the characterisation of a particular real neural system in a neurobiological preparation, with an attempt to derive concentration, weighting and periodic behaviour parameters. These could be installed in a computational model, built at as fine a grain size as possible, which should behave in a similar manner to the biological network. The parameters of the computational network could then be varied to assess the effects of removing any peptidergic components from the system, and comparing the behaviour obtained with the real neural system.

A project such as this should be a valuable contribution to knowledge in this area, providing us with fairly reliable proof of the contribution of neuropeptides to the specific neural circuit.

Conclusion

This study has pursued the theory that neuropeptides can be a complementary system of signal propagation in a nervous system. At the highest level, the proposal is that neuropeptides can act as additional signal channels in a neural circuit, providing a parallel and concurrent system of signal propagation. At a lower level, neuropeptides may act as signal modulators, providing an extension of the activation produced by classical transmitters. At an even

lower level, neuropeptides can be considered as noise, which may, or may not influence signal propagation. Finally, the possibility exists that neuropeptides serve no useful purpose in the nervous system.

This study has begun to address the question of which level neuropeptides inhabit by testing the behaviour of two artificial neural networks, at very different levels of neural representation. The Back–propagation model used in this study, showed that random noise injected into the network enhanced the rate of learning to a greater extent than a fabricated 'peptidergic' leakage signal injection. However, the 'peptidergic' leakage signal injection did show an enhanced rate of learning over the basic back–propagation algorithm. The medium grain model, at a lower level of neural representation, showed that the addition of a 'peptidergic' factor extended the normal activation of the target by a factor of 2.5.

The assumptions made about these results are that the 'peptidergic' factor *extends* the activation range of the network nodes in both models, whilst the 'noise' factor *broadens* the activation range in the back–propagation model, as well as adding a chance element to the convergence process.

The assignment of peptidergic activity level depends largely on the independence of the 'peptidergic' signal. If the signal can be classed as independent, then concurrent propagation can be said to occur. If the signal always occurs at, or around, the normal excitation time of the cell, then the peptidergic component is a mere extension of the ordinary activation of the cell. The activation extensions produced, in the back–propagation model with the 'peptidergic' component, cannot be classified according to their dependence, because the activations produced in this model are

largely interdependent. The activation extension produced by the medium–grain model depends, primarily, on the setting of the 'peptidergic' activity strengths and time–constants, which was not based on biological data, as none was available. This model does seem to show depolarisations produced by the peptidergic factor acting as independent of the primary 'classical' depolarisations, but only to the extent that they are time–shifted by variations in 'diffusion' delays. For these reasons, it is impossible to extend the results of this study into an assumption about the dependence level of peptidergic activity.

It is possible to state that it is unlikely that the 'peptidergic' factor in this study, was acting as a "noise" phenomenon. This result may be due to incorrect assumptions about the levels at which peptides operate in the nervous system, but invalidation of these results lies in a closer, and hopefully multi–disciplinary, examination of the parameters of peptidergic function in a neural system. It is therefore, possible to conclude that 'peptidergic' factors in artificial neural networks appear to enhance the operation of the network through a process of signal modulation, which appears to extend the activation range of network nodes. Extending these results to a biological nervous system is inappropriate, at the moment, due to the lack of verité in the parameters of the model systems; but it is hoped that this study provides a reasonable basis for the further examination of peptidergic functions in real neural systems.

Index

List of Figures and Tables

Chapter 2

Figures

Chapter 3

Figures

Tables

Chapter 4

Figures

Tables

Chapter 5

Figures

Chapter 6

Figures

Tables

Chapter 7

Tables

References

Aleksander I & Morton H. An Introduction to Neural Computing: North Oxford Academic Press; 1991.

Aleksander I. (ed). Neural Computing architectures: The design of brain like machines: North Oxford Academic; 1989b.

Bernstein J. Profiles: AI, Marvin Minsky. The New Yorker; 1981; December 14.: 50-126.

Bliss T.V.P., Lomo T. Long lasting Potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path. Journal of Physiology (London); 1973; 232: p331-356.

Cajal S. The structure and connections of neurons. In Nobel Lectures: Physiology or Medicine 1901-1921.: Elsevier; 1967.

Carpenter G.A., & Grossberg S. ART3 hierarchical search: Chemical transmitters in self–organizing pattern recognition architectures. International Joint Conference on Neural Networks (Washington DC): Erlbaum; 1990; II: 30-33.

Changeux J-P. Neuronal Man: Pantheon Books; 1985.

Clark J.W., Chen J-W., Kurten K.E. Analog simulation of circuits in the olfactory bulb.. Cotterill R.M.J. Models of Brain Function: Cambridge University Press; 1989: 327-347.

Colvin G. Synapsis: A Neural Network. The C users Journal; April 1989.

Crick F. Do dendritic spines twitch? Trends in Neurosciences; 1982; 5: 44-46.

Dahl E. Accelerated learning using the generalised delta rule. Proc IEEE First International conference on Neural Networks.; 1987.

Docking P. Back propagation in neural networks: A comparative test of some recent extensions: MSc Dissertation Kingston Polytechnic, Surrey; 1989.

Emson. Neurotransmitter systems. Bousfield D. Neurotransmitters in Action.: Elsevier; 1985: 6-9.

Emson P C, De Quidt M E. NPY - a new member of the pancreatic polypeptide family. Bousfield D. Neurotransmitters in action: Elsevier; 1985: 338-346.

Feldman J.A. & Ballard D.H. Connectionist models and their properties. Cognitive science; 1982; Vol 6: 205-254.

Feldman J.A. Connectionist models and their applications: Introduction. Cognitive science; 1985; Vol 9: 1-2.

Fifkova E., Markham J.A., Delay R. Calcium in the spine apparatus of dendritic spines in the dentate molecular layer. Brain Research; 1983; 266: 163-168.

Fukushima K., Miyake S. & Ito T. Neocognitron: A Neural Network Model for a Mechanism of Visual Pattern Recognition. IEEE Transactions on Systems, Man and Cubernetics.; 1983; Vol SMC-13(No. 5).

Furshpan E.J., Potter D.D. Transmissionat the giant motor synapses of crayfish. Journal of Physiology (London); 1959; 145: 289-325.

Garthwaite J. Glutamate, Nitric Oxide and cell-cell signalling in the nervous system. Trends in Neurosciences; 1991; 14(2): 60-67.

Gershon M. D., Schwartz J. H. & Kandel E. R. Morphology of Chemical Synapses and Patterns of Interconnection. Kandel E. R. & Schwartz J. H. (eds). Principles of Neural Science (2nd Ed): Elsevier; 1985: Pp 132-147.

Grossberg S. Adaptive pattern classificationand universal recoding: Part 1, parallel development and coding of neural feature detectors. Biological Cybernetics; 1976; 23: 121-134.

Hebb D.O. Organization of Behavior: Wiley; 1949.

Hecht-Nielsen R. Counterpropagation Networks. Proc IEEE First International conference on Neural Networks; 1987; Vol 2.

Hinton G.E. & Sejnowski T.J. Learning and Relearning in Boltzmann Machines.. Rumelhart D.E. & McClelland J.L. Parallel Distributed processing Volume 1.; 1986.

Hodgkin A L, Huxley A F. A quantitative description of membrane current and its application to conduction and excitation in nerve. Journal of Physiology, London; 1952; 117: 500-544.

Hollt V. Multiple endogenous opioid peptides. Bousfield D. Neurotransmitters in action: Elsevier; 1985: 188-193.

Hopfield J.J. & Tank D.W. Computing with neural circuits: A model. Science; 1986; 233(4764): 625-633.

Hopfield J.J. Neural Networks and physical systems with emergent collective computational abilities. Poceedings of the National Academy of Sciences, USA; 1982; 79: p 2553-2558.

Horn J.P., Dodd J. Inhibitory cholinergic synapses in autonomic ganglia. Bousfield D. Neurotransmitters in Action.: Elsevier; 1985.

Hubel D H, Wiesel T N. Binocular interaction in striate cortex of kittens reared with artificial squint. Journal of Neurophysiology; 1965; 28: 1041-1059.

Hubel D.H., Wiesel T.N. Receptive fields of single neurons in the cat's striate cortex. Journal of Physiology. (London); 1959; 148: 574-591.

Hughes J. Opioid peptides - families of receptors and neurotransmitters. Bousfield D. Neurotransmitters in action: Elsevier; 1985: 13-16.

Jan Y N, Jan L Y. A LHRH-like peptidergic neurotransmitter capable of "action at a distance" in autonomic ganglia. Bousfield D. Neurotransmitters in Action.: Elsevier; 1985: 94-103.

Joels M, de Kloet E R. Control of neuronal excitability by corticosteriod hormones. Trends in Neurosciences; 1992; 15(1): 25-29.

Kandel E R. Factors controlling transmitter release.. Kandel E R, Schwartz J H. Principles of neural science (2nd ed.): Elsevier; 1985: 120-131.

Katz B., Miledi R. A study of synaptic transmission in the absense of nerve impulses. Journal of Physiology (London); 1967; 192: 407-436.

Katz B., Miledi R. The role of Calcium in Neuromuscular facilitation. Journal of Physiology (London); 1968; 195: p481-492.

Kelly D D. Central representations of pain and analgesia. Kandel E R, Schwartz J H. Principles of neural science (2nd ed.): Elsevier; 1985: 331-341.

Koch C, Poggio T. Biophysics of Computation: Neurons, Synapses, and Membranes. Feldman G M, Ball W E, Cowan W M (eds). Synaptic Function. New York: Wiley; 1987: 637-697.

Koch C. Cable theory in neurons with active, linearized membranes. Biological Cybernetics; 1984; 50: 15-33.

Koch C., Poggio T. A simple algorithm for solving the cable equation in dendritic trees of arbitrary geometry. Journal of Neuroscience Methods; 1985; 12: 303-315.

Koch C., Poggio T., Torre V. Nonlinear interaction in a dendritic tree: Localization, timing and role in information processing. Proceedings of the National Academy of Sciences (USA); 1983; 80: 2799-2802.

Kohonen T. Speech recognition based on topology-preserving neural maps. Aleksander I. Neural Computing Architectures: The design of brain-like machines: North Oxford Academic; 1989: 26-40.

Kosko B. Competitive Adaptive Bidirectional Associative Memories. Proc IEEE First International conference on Neural Networks.; 1987; Vol 2: 759-766.

Kung I, Hwang U. An algebraic projection analysis for optimal hidden units size and learning rates in back propagation learning. Proc IEEE International conference on Neural Networks.; 1988.

Landahl H D, McCulloch W S, Pitts W. A statistical consequence of the logical calculus of nervous nets. Bull of Mathematical Biophysics; 1943; 5: 135-137.

Lashley K.S. In search of the engram. Soc. of Experimental Biology Symposium No. 4: Psy. Mechanism in animal behav.: Cambridge University Press; 1950: 478-505.

Levy W.B., Seward O. Synapses as associative memory elements in the Hippocampal formation. Brain Research; 1979; 175: p233-245.

Llinas R. Biological Neural Networks Seminar. Kings College, London; 1991: (April 11).

Lorente de No. Anatomy of the eighth nerve. Laryngoscope; 1933; 43: 327-350.

Lundberg J.M., Hökfelt T. Coexistence of Peptides and Classical neurotransmitters. Bousfield D. Neurotransmitters in Action.: Elsevier; 1985.

Marcel A. Negative set effects in character classification: A response retrieval view of Reaction Time. Q.J Experimental Psychology; 1977; Vol 29: 31-48.

Martin A.R., Ringham G.L. Synaptic tranfer at a vertebrate central nervous system synapse. Journal of Physiology (London); 1975; 251: 409-426.

Matsumoto G. Neurocomputing - Neurons as Microcomputers. Future Generations of Computer Systems; 1988; Vol 4: 39-51.

McBurney R.N. New approaches to the study of rapid events underlying neurotransmitter release.. Bousfield D. Neurotransmitters in Action.: Elsevier Biomedical Press; 1985.

McClelland J.L. & Rumelhart D.E. Explorations in Parallel Distributed Processing: Bradford - MIT Press; 1988.

McCulloch W.S. & Pitts W. A logical calculus of the ideas immanent in nervous activity. Bull. of Mathematical Biophysics; 1943; Vol 5: 115-133.

Miller R. How do opiates act?. Bousfield D. Neurotransmitters in action: Elsevier; 1985: 216-219.

Minsky M.L. & Papert S.A. Perceptrons: Expanded Edition: MIT Press; 1969,1988.

Morrison J H, Magistretti P J. Monoamines and peptides in cerebral cortex: contrasting principles of cortical organisation. Bousfield D. Neurotransmitters in Action.: Elsevier; 1985: 319-328.

Mountcastle V. B. An organising principle for cerebral function: The unit module and the distributed system. Edelman G. M. & Mountcastle V. B. The Mindful Brain: MIT Press; 1978: Pp 7–50.

Nagy J I. Capsaicin's action on the nervous system.. Bousfield D. Neurotransmitters in action: Elsevier; 1985.

Neisser U. Cognitive Psychology: Prentice Hall; 1967.

Nickerson R.S. Binary classification Reaction Time: A review of some studies of human information processing capabilities. Psychonomic Monographs; 1972; Vol 4((17)): 275-318.

Ordway R W, Singer J J, Walsh J V. Direct regulation of ion channels by fatty acids. Trends in Neurosciences; 1991; 14(3): 96-100.

Otsuka M, Konishi S. Substance P - the first peptide neurotransmitter?. Bousfield D. Neurotransmitters in Action.: Elsevier; 1985: 163-169.

Peters A. & Kara D.A. Dendritic bundles in the cortex. J. of Comparitive Neurology; 1987; Vol 260(No. 4).

Pollack J. B. Connectionism: Past, Present and Future. Artificial Intelligence Review; 1989; Vol 3: 3-20.

Rall W. Core conductor theory and cable properties of neurons. Kandel E.R. Handbook of Physiology: American Physiological Society; 1977; Volume 1: Section 1.

Rall W. Theoretical significance of dendritic tree for input-output relation. Reiss R F. Neural theory and modeling: Stanford university press; 1964: 73-97.

Rall w. Cable Theory. Koch C,. Segev I. Methods in Neuronal Modelling: MIT Press; 1989.

Rossier J. The peptide explosion and the new genetics. Bousfield D. Neurotransmitters in Action.: Elsevier; 1985.

Rumelhart D, McClelland J. Interactive processing through spreading activation.. Lesgold A, Perfetti C. Interactive processes in reading: Erlbaum; 1981.

Rumelhart D.E., Hinton G.E. & McClelland J.L. A General Framework for Parallel Distributed Processing. Rumelhart D.E. & McClelland J.L. Parallel Distributed processing Volume 1: MIT Press; 1986.

Rumelhart D.E., Hinton G.E. & Williams R.J. Learning Internal Representations by Error Propagation. Rumelhart D.E. & McClelland J.L. Parallel Distributed processing Volume 1: MIT Press; 1986.

Scheller R.H., Rothman B.S., Mayeri E. A single gene encodes multiple peptide transmitter candidates involved in a stereotyped behaviour. Bousfield D. Neurotransmitters in Action..: Elsevier; 1985.

Schwartz J H. Molecular steps in synaptic transmission.. Kandel E R, Schwartz J H. Principles of neural science (2nd ed.): Elsevier; 1985: 169-175.

Schwindt P.C., Crill W.E. Factors influencing motoneuron rhythmic firing: Results from a voltage clamp study. Journal of Neurophysiology; 1982; 48: 875-890.

Segev I, Fleshman J, Burke R. Compartmental Models of Complex Neurons. Koch C, Segev I. Methods in Neuronal Modeling: MIT Press; 1989.

Sejnowski T, Koch C, Churchland P. Computational Neuroscience. Science; 1988; 241: 1299-1306.

Sejnowski T. Chattarji S. & Stanton P. Induction of Synaptic Plasticity by Hebbian Covariance in the Hippocampus. Durbin R, Miall C and Mitchison G. The Computing Neuron.: Addison Wesley; 1989.

Selverston A., Mazzoni P. Flexibility of computational units in invertebrate CPGs.. Durbin R., Miall C., Mitchison G. The Computing Neuron: Addison Wesley; 1989: 205-228.

Shepherd G.M. Synaptic and impulse loci in olfactory buld dendritic circuits.. Roberts A., Bush B.M.B. Neurones without impulses.: Cambridge University Press; 1979: 255-267.

Siegelbaum S.A., Tsien R.W. Modulation of gated ion channels as a mode of transmitter action. Trends in Neurosciences; 1983; 6: 307-313.

Sofroniew M V. Vasopressin and oxytocin in the mammalian brain and spinal cord. Bousfield D. Neurotransmitters in action: Elsevier; 1985: 329-337.

Stratford K, Mason A, Larkman A, Major G, Jack J. The modelling of pyramidal neorones in the visual cortex. Durbin R, Miall C, Mitchison G. The Computing Neuron: Addison Wesley; 1989: 296-321.

Swindale N. V. Is the Cerebral Cortex modular? Trends in Neurosciences; 1990; 13(12): 487-492.

Von Lehman A., Paek E.G., Liao P.F., Marrakchi A., Patel J.S. Factors Influencing learning by Back Propagation. Proc IEEE First International conference on Neural Networks.; 1987.

von Euler U S. The history of substance P. Bousfield D. Neurotransmitters in action: Elsevier; 1985: 143-150.

Watrous R. Learning algorithms for connectionist networks: Applied gradient methods of nonlinear optimization. Proc IEEE First International conference on Neural Networks.; 1987.

Weber E, Evans C J, Barchas J D. Multiple endogenous ligands for opioid receptors.. Bousfield D. Neurotransmitters in action: Elsevier; 1985: 194-200.

Widrow G, Hoff M E. Adaptive switching circuits. Institute of Radio Engineers. Western Electronic Show and Convention, Convention Record; 1960; 4: 96-104.

Wilson M A, Bower J M. The simulation of large scale networks. Koch C, Segev I. Methods in Neuronal Modeling: Bradford/MIT; 1989: 291-334.

Yamada W M, Koch C, Adams P R. Multiple channels and Calcium dynamics. Koch C, Segev I. Methods in neuronal modeling: MIT Press; 1989: 97-135.

Zukin R S, Zukin S R. The case for multiple opiate receptors. Bousfield D. Neurotransmitters in action: Elsevier; 1985: 201-208.

Appendix A: Program Listings

```
/*  ***************  Program  alpha.c
**********************/

#define YES 1
#define NO 0
#define ONLINE NO

#define DEBUG NO
#define D_DEBUG NO
#define N_DEBUG NO
#define BIG_DEBUG NO
#define OUTPUT NO
#define RESULTS YES

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <CursorCtl.h>
#include <OSUtils.h>
#include "matrix.h"
#include "NNf.h"
#include "alpha.h"

/* global variable - display converged message only once */
int onceOnly = 0;
int lastone = 0;   /* and write a check of the outputs  */
float  TotFact;


main()

{
      char  line[81],fname[81];
      float  *op_char,*ip_char,real_op[8],error;
      int  i,j,n_of_layers,neurons_ip[3],neurons_op[3],
      rate[3],crit,batch_size,finish;
      int  binary_op[8],q,c,re_use_rate;
      long  n,n_of_inputs,n_of_outputs;
      float  mean_err,batch_err,old_batch_err,
      crit_error,m_crit_error,frateSave;
      NETWORK *network;
      NEURON *inputs, *outputs;
      FILE  *fp,*fr;
```

```
 extern int update,useMomentum,useBias,useNoise;
 extern float PepLeak,PepStrength,PepFactor,TotFact;
 extern DelayLinePtr DPtr;

 TotFact = 0;

 /* Assign  initial parameters */
 /* Number of layers refers to 'layers of modifible
  connections', and is therefore
'number of layers of nodes' - 1  */

 n_of_layers=2;

 /*if (ONLINE)
      printf("enter No. of iterations/batches:\n");
 gets(line);
 n=atoi(line);  */
 n = 20000; /* We're going for convergence here! */

 if (ONLINE)
      printf("enter Log filename:\n");
 gets(fname);



 /* assign batch size */
 batch_size = 3;

 /* input connections */
 n_of_inputs=15;
 neurons_ip[0]=15;
 neurons_ip[1]=15;

 /* output connections */
 neurons_op[0]=15;
 neurons_op[1]=8;
 n_of_outputs=8;


 /* Learning rate */
 if (ONLINE)
      printf("enter learning rate ( 0.0 to 1.0 ):\n");
 gets(line);
 frate =  atof(line);
 frate /= 100;
 frateSave = frate;
```

```
        fprintf(stdout,"Learning Rate = %-1.8g\n", frate);
        fflush(stdout);

        /* Learning momentum */
        if (ONLINE)
              printf("Use momentum? ( 0 or 1 ):\n");
        useMomentum = atoi(gets(line));


        /* Unit bias */
        if (ONLINE)
              printf("Use bias? ( 0 or 1 ):\n");
        useBias =  atoi(gets(line));

        if (ONLINE)  /* NOISE factor gathered here; I'm lazy */
              printf("Enter Noise Factor:\n");
        PepFactor =  atof(gets(line));
        PepFactor /= 100;

        /*if (ONLINE)
        printf("Enter Peptide Strength:\n");
        PepStrength = 0.0;     /*PepStrength =  atof(gets(line));  */

        /*if (ONLINE)
        printf("Enter Peptide Leak Factor:\n");
        PepLeak =  atof(gets(line));
        PepLeak /= 100;*/

        /* Pre-Factor for calculation */
/* PepFactor = PepStrength * PepLeak; */
        /*if (ONLINE)
        {     * /
        fprintf(stdout,"Noise Factor = %-1.8g\n", PepFactor);
        fflush(stdout);
        /*}  * /

        /* Use Noise addition */
/* printf("Use Noise? ( 0 or 1 ):\n"); */
        useNoise =        1;    /* always in this case */
                    /*    atoi(gets(line));   */



        /* set ctitical error for convergence indicator */
        crit_error = 0.01;
```

```
    /* Init finish counter - net will exit after convergence + 200
iterations  */
    finish = 0;

    /* create the network, randomise synapses */

    network =
new_network(&n_of_layers,neurons_ip,neurons_op);
    if  (!network)
        printf("out  of  memory\n"),  exit(0);

    /* Create the Delay List for p-activation */
    if  (useNoise)
        DPtr = DCreate(DELAYNO);          /*  create
DelayLineNO DelayLine lists  */

    /* Set pointers to first and last layers of the network */
    inputs=network->first_layer->inputs;
    outputs=network->last_layer->outputs;


    /* Randomise the synapse weights: specify the random
number generator seed */
    randomise(network, 300.0, 89L);


    /*  Init  network  */

    for  ( j=0; j<n_of_inputs; j++)
        inputs[j].activation  =  0.0;
    feedforward(network);    /* a null feedforward cycle */


    /* Open log file if output is specified */
    if (OUTPUT)
    {
        if((fp = fopen(fname,"w")) == NULL)
        {
            fprintf(stderr,"Cant  open  the  logfile\n");
            exit(1);
        }
    }

    if (RESULTS)
    {
```

```
        if((fr = fopen("Results","a")) == NULL)
        {
            fprintf(stderr,"Cant open the Results file\n");
            exit(1);
        }
    }


    InitCursorCtl;


    /* For each iteration ..... */
    for (i=0; i<n; i++)
    {

        /* for each input pattern .....*/
        for(batch_err = 0, c=0 ,crit=0; c < batch_size ; c++ )
        {

            /* Don't update weight values inside this loop */
            update = 0;

            /* Assign input and output vectors */
            ip_char = ip_array[c];
            op_char = op_array[c];

            if(BIG_DEBUG)
                printf("c = %d\n",c);


            if(BIG_DEBUG)
            {
                printf("ip_char = \n");
                for ( j=0; j < n_of_inputs; j++)

    printf("%f%c",ip_char[j],((j+1)%batch_size == 0 || j ==
n_of_inputs-1) ? '\n' : ' ');

                printf("op_char = \n");
                for ( j=0; j < n_of_outputs; j++)
                    printf("%f%c",op_char[j],(j ==
n_of_outputs-1) ? '\n' : ' ');
            }
            /* end DEBUG */
```

```
                /* initialise the input vector from the "alphabet"
header file*/
                for ( j=0; j<n_of_inputs; j++)
                      inputs[j].activation = ip_char[j];



                /* Feed forward through the network */
                feedforward(network);


                /* Make calculations of peptidergic noise for each
iteration*/
                /* Not the usual method */
                /*if (useNoise)
                      noise_calc(network);   */

                /* calc errors and assign output figures */
                for ( mean_err = 0, j=0; j<n_of_outputs; j++,
mean_err += error)
                {
                      /* calculate errors */
                      error = outputs[j].errors = (op_char[j] -
outputs[j].activation);

                      /* count number of OK errors */
                      if(fabs(error) < crit_error) crit++;

                      if(DEBUG || i>(n-50) || finish > 195 )
                      {

                            /* Assign Real Values */
                            real_op[j]=outputs[j].activation;
                      }
                      if(!lastone && onceOnly)
                      {

                          /* Assign Real Values */
                          real_op[j]=outputs[j].activation;
                      }
                }

                batch_err += (mean_err/n_of_outputs);
```

```
                if (DEBUG || i>(n-5) || (!lastone  && onceOnly) ||
finish > 195 )
                {
                        /* Report the error indicator, and  outputs
(Binary and Real)*/

                        if (OUTPUT)
                        {
                                for(j=0;j<n_of_outputs;j++)
fprintf(fp,"%f,",real_op[j]);
                                fprintf(fp,"\n");
                        }
                        if(i == (n-1) || finish == 200  )
                        {
                                fprintf(stdout,"Final  Condition\n");
                                for(j=0;j<n_of_outputs;j++)
fprintf(stdout,"%f,",real_op[j]);
                                fprintf(stdout,"\n");
                        }

                }
                /* if the net has just converged , give the state of
the outputs */
                if(!lastone  && onceOnly )
                {
                        for(j=0;j<n_of_outputs;j++)
fprintf(stdout,"%f,",real_op[j]);
                        fprintf(stdout,"\n");
                        fflush(stdout);
                }

                /* update the weight error derivatives here but
don't change weights */
                /* weight change is done after each batch  (later)
* /
                /* If end of batch don't do feedback here! */
                if(c != batch_size-1)
                        feedback(network);



        }               /* end "for each input pattern" */


        /* set the converged state if converged */
```

```
            lastone = onceOnly;

            /* report if network has converged */
            if(crit == (n_of_outputs*batch_size) && !onceOnly)
            {
                  if (OUTPUT)
                        fprintf(fp,"****** iteration %d, network
converged  *****\n",i);

                  fprintf(stdout,"****** iteration %d, network
converged  *****\n",i);

                  if (RESULTS)

      fprintf(fr,"%f\t%d\t%s\n",PepFactor,i,fname);

                  onceOnly = 1;
            }

            if(OUTPUT)
            {
                  if ( DEBUG  ||  i>(n-5) || finish > 195)
                        fprintf(fp,"iteration %d mean error
%f\n",i,batch_err/batch_size);


                  fprintf(fp,"%f\n",batch_err/batch_size);
            }

            /* Do P-activation calculation here if batch method
selected */
            /*if  (useNoise)
                  noise_calc(network); */

            /* Feed errors back through the network  - batch
method */
            update = 1;
            feedback(network);

            if (onceOnly)
            {
                  finish++;
                  if (finish > 200)
                        break;
            }
            RotateCursor(i);
```

```
        }           /* end "for each iteration" */


        if (OUTPUT)
        {
              fclose(fp);

        }

        if (RESULTS)
        {
              fclose(fr);
        }

        fprintf(stdout,"***** Finished ***** TotFact =
%f\n",TotFact);

        if (ONLINE)
        {
              SysBeep(200);
              SysBeep(200);
        }

        /* end of test */
        return(0);

}
```

```
/* Matrix.h Header FIle - for use with NN.h and alpha.c
 *  This file defines the Inter- and Intra-layer distances between
nodes
 */

#define __MATRIX__

 /*   DELAYNO defines the number of lists in use
  */
#define DELAYNO 4


/* Extra-Layer (Inter-layer) distances: Vertical, Horizontal and
Offset   */
/* The Pythagorean distances are :
            V1   1.030776406
            V2   1.118033989
            V3   1.25
            V4   1.414213562
            H1   V2
            H2   V4
            O11  1.145643924
            O12  1.224744871
            O13  1.346291202
            O14  1.5
            O21  1.436140662
            O22  O14
            O23  1.600781059
            O24  1.732050808
* /
/* Translated to 1/exp(distance)  (for speed of execution) this is:
* /
#define    V1   0.3567298857980956
#define    V2   0.3269218952699931
#define    V3   0.2865047968601901
#define    V4   0.2431167345249198
#define    H1   0.3269218952699931
#define    H2   0.2431167345249198
#define    O11  0.3180190717321045
#define    O12  0.2938326559931346
#define    O13  0.2602035155609003
#define    O14  0.2231301601484298
#define    O21  0.2378439097155739
#define    O22  0.2231301601484298
#define    O23  0.2017388864699812
#define    O24  0.1769212062414894
```

```
/* Intra-layer distances: Vertical, Horizontal and Offset  */
/* The Pythagorean distances are :
            IV1  0.25
            IV2  0.5
            IV3  0.75
            IV4  1.0
            IH1  IV2
            IH2  IV4
            I11  0.559016994
            I12  0.707106781
            I13  0.901387818
            I14  1.118033989
            I21  1.030776406
            I22  I14
            I23  1.25
            I24  1.414213562
* /
/* Translated to 1/exp(distance)  (for speed of execution) this is:
* /
#define    IV1  0.7788007830714049
#define    IV2  0.6065306597126334
#define    IV3  0.4723665527410147
#define    IV4  0.3678794411714423
#define    IH1  0.6065306597126334
#define    IH2  0.3678794411714423
#define    I11  0.57177084185617l3
#define    I12  0.4930686914872205
#define    I13  0.4060058064019639
#define    I14  0.3269218952699931
#define    I21  0.3567298857980956
#define    I22  0.3269218952699931
#define    I23  0.2865047968601901
#define    I24  0.2431167345249198


/* Extra-Layer (Inter-layer) (2nd Order) distances: Vertical,
Horizontal and Offset  */
/* The Pythagorean distances are :
            EV1  2.015564437
            EV2  2.061552813
            EV3  2.136000936
            EV4  2.236067977
            EH1  EV2
            EH2  EV4
            E11  2.076655966
```

```
            E12  2.121320344
            E13  2.193741097
            E14  2.291287847
            E21  2.25
            E22  E14
            E23  2.358495283
            E24  2.449489743
* /
/* Translated to 1/exp(distance)  (for speed of execution) this is:
* /
#define    EV1  0.1332451736306551
#define    EV2  0.1272562112942661
#define    EV3  0.1181262943225883
#define    EV4  0.1068779257138023
#define    EH1  0.1272562112942661
#define    EH2  0.1068779257138023
#define    E11  0.1253486823750868
#define    E12  0.1198732500509750
#define    E13  0.1114988394116792
#define    E14  0.1011361300864856
#define    E21  0.1053992245618643
#define    E22  0.1011361300864856
#define    E23  0.0945624058567956
#define    E24  0.0863376296416422



/* Node Number for continued peptidergic activation calculation
* /
int nodeno = 0;

/* Distance Matrices (1/exp(distance)) : extra-layer   (Inter-layer)
* /
/* Dummy elements allow us to forget that C arrays start at zero
* /

float eMatrix[16][16] =
{
      {       0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
      }     ,
      {
0,1,V1,V2,V3,V4,H1,O11,O12,O13,O14,H2,O21,O22,O23,O24
      }     ,
      {
0,V1,1,V1,V2,V3,O11,H1,O11,O12,O13,O21,H2,O21,O22,O23
```

```
        }       ,
        {
0,V2,V1,1,V1,V2,O12,O11,H1,O11,O12,O22,O21,H2,O21,O22
        }       ,
        {
0,V3,V2,V1,1,V1,O13,O12,O11,H1,O11,O23,O22,O21,H2,O21
        }       ,
        {
0,V4,V3,V2,V1,1,O14,O13,O12,O11,H1,O24,O23,O22,O21,H2
        }       ,
        {
0,H1,O11,O12,O13,O14,1,V1,V2,V3,V4,H1,O11,O12,O13,O14
        }       ,
        {
0,O11,H1,O11,O12,O13,V1,1,V1,V2,V3,O11,H1,O11,O12,O13
        }       ,
        {
0,O12,O11,H1,O11,O12,V2,V1,1,V1,V2,O12,O11,H1,O11,O12
        }       ,
        {
0,O13,O12,O11,H1,O11,V3,V2,V1,1,V1,O13,O12,O11,H1,O11
        }       ,
        {
0,O14,O13,O12,O11,H1,V4,V3,V2,V1,1,O14,O13,O12,O11,H1
        }       ,
        {
0,H2,O21,O22,O23,O24,H1,O11,O12,O13,O14,1,V1,V2,V3,V4
        }       ,
        {
0,O21,H2,O21,O22,O23,O11,H1,O11,O21,O22,V1,1,V1,V2,V3
        }       ,
        {
0,O22,O21,H2,O21,O22,O12,O11,H1,O11,O12,V2,V1,1,V1,V2
        }       ,
        {
0,O23,O22,O21,H2,O21,O13,O12,O11,H1,O11,V3,V2,V1,1,V1
        }       ,
        {
0,O24,O23,O22,O21,H2,O14,O13,O12,O11,H1,V4,V3,V2,V1,1
        }
}       ;


/* Distance Matrices (1/exp(distance)): Intra-layer */
/* Dummy elements allow us to forget that C arrays start at zero
* /
```

```
float iMatrix[16][16] =
{
        {       0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        }       ,
        {
0,0,IV1,IV2,IV3,IV4,IH1,I11,I12,I13,I14,IH2,I21,I22,I23,I24
        }       ,
        {
0,IV1,0,IV1,IV2,IV3,I11,IH1,I11,I12,I13,I21,IH2,I21,I22,I23
        }       ,
        {
0,IV2,IV1,0,IV1,IV2,I12,I11,IH1,I11,I12,I22,I21,IH2,I21,I22
        }       ,
        {
0,IV3,IV2,IV1,0,IV1,I13,I12,I11,IH1,I11,I23,I22,I21,IH2,I21
        }       ,
        {
0,IV4,IV3,IV2,IV1,0,I14,I13,I12,I11,IH1,I24,I23,I22,I21,IH2
        }       ,
        {
0,IH1,I11,I12,I13,I14,0,IV1,IV2,IV3,IV4,IH1,I11,I12,I13,I14
        }       ,
        {
0,I11,IH1,I11,I12,I13,IV1,0,IV1,IV2,IV3,I11,IH1,I11,I12,I13
        }       ,
        {
0,I12,I11,IH1,I11,I12,IV2,IV1,0,IV1,IV2,I12,I11,IH1,I11,I12
        }       ,
        {
0,I13,I12,I11,IH1,I11,IV3,IV2,IV1,0,IV1,I13,I12,I11,IH1,I11
        }       ,
        {
0,I14,I13,I12,I11,IH1,IV4,IV3,IV2,IV1,0,I14,I13,I12,I11,IH1
        }       ,
        {
0,IH2,I21,I22,I23,I24,IH1,I11,I12,I13,I14,0,IV1,IV2,IV3,IV4
        }       ,
        {
0,I21,IH2,I21,I22,I23,I11,IH1,I11,I21,I22,IV1,0,IV1,IV2,IV3
        }       ,
        {
0,I22,I21,IH2,I21,I22,I12,I11,IH1,I11,I12,IV2,IV1,0,IV1,IV2
        }       ,
        {
0,I23,I22,I21,IH2,I21,I13,I12,I11,IH1,I11,IV3,IV2,IV1,0,IV1
```

```
        }       ,
        {
0,I24,I23,I22,I21,IH2,I14,I13,I12,I11,IH1,IV4,IV3,IV2,IV1,0
        }
}       ;

/* Distance Matrices (1/exp(distance)): 2nd order Inter-layer */
/* Dummy elements allow us to forget that C arrays start at zero
* /

float  exMatrix[16][16] =
{
        {       0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        }       ,
        {
0,0,EV1,EV2,EV3,EV4,EH1,E11,E12,E13,E14,EH2,E21,E22,E23,E24
        }       ,
        {
0,EV1,0,EV1,EV2,EV3,E11,EH1,E11,E12,E13,E21,EH2,E21,E22,E23
        }       ,
        {
0,EV2,EV1,0,EV1,EV2,E12,E11,EH1,E11,E12,E22,E21,EH2,E21,E22
        }       ,
        {
0,EV3,EV2,EV1,0,EV1,E13,E12,E11,EH1,E11,E23,E22,E21,EH2,E21
        }       ,
        {
0,EV4,EV3,EV2,EV1,0,E14,E13,E12,E11,EH1,E24,E23,E22,E21,EH2
        }       ,
        {
0,EH1,E11,E12,E13,E14,0,EV1,EV2,EV3,EV4,EH1,E11,E12,E13,E14
        }       ,
        {
0,E11,EH1,E11,E12,E13,EV1,0,EV1,EV2,EV3,E11,EH1,E11,E12,E13
        }       ,
        {
0,E12,E11,EH1,E11,E12,EV2,EV1,0,EV1,EV2,E12,E11,EH1,E11,E12
        }       ,
        {
0,E13,E12,E11,EH1,E11,EV3,EV2,EV1,0,EV1,E13,E12,E11,EH1,E11
        }       ,
        {
0,E14,E13,E12,E11,EH1,EV4,EV3,EV2,EV1,0,E14,E13,E12,E11,EH1
        }       ,
        {
0,EH2,E21,E22,E23,E24,EH1,E11,E12,E13,E14,0,EV1,EV2,EV3,EV4
```

```
        }     ,
        {
 0,E21,EH2,E21,E22,E23,E11,EH1,E11,E21,E22,EV1,0,EV1,EV2,EV3
        }     ,
        {
 0,E22,E21,EH2,E21,E22,E12,E11,EH1,E11,E12,EV2,EV1,0,EV1,EV2
        }     ,
        {
 0,E23,E22,E21,EH2,E21,E13,E12,E11,EH1,E11,EV3,EV2,EV1,0,EV1
        }     ,
        {
 0,E24,E23,E22,E21,EH2,E14,E13,E12,E11,EH1,EV4,EV3,EV2,EV1,0
        }
}       ;



/* The output layer requires a lookup table for distance calculation
* /

int OLIndex[9] =
{       0,2,3,4,7,9,12,13,14
}       ;


/* The 4th Dimension - Time; a round linked list that can grow to
represent
 *  varying delays - one storage unit needed for each node
  */

struct  delayline
{
        struct  delayline  *nextPtr;                    /* Ptr to next
list   */
        float   dlist[39];                              /* list array -
one longer, as usual for above reasons */
        struct  delayline  *prevPtr;
}       ;

typedef struct delayline DelayLine;
typedef  DelayLine *DelayLinePtr;

DelayLinePtr DPtr;                              /* Create the
necessary  Ptr */
```

```
/* PEP_STRENGTH is the binding strength of peptides relative to
   * classic transmitters and is used as a factor in the calulation of
  * activation   */

/* PEP_LEAK is the amount of peptidergic signal leaked from
   * a synapse on a target cell - this goes to make up the diffusible
   * peptidergic signal */

float  PepLeak,PepStrength,PepFactor;
```

```
/* Alpha.h - defines global inputs and outputs for each character
* /


/*Inputs are 3x5 matries of alpha chars */

float ip_array[3][15] =
{
/* "A"   */
      {     0.0,1.0,0.0,
            1.0,0.0,1.0,
            1.0,1.0,1.0,
            1.0,0.0,1.0,
            1.0,0.0,1.0
      }     ,
/* "B"   */
      {     1.0,1.0,0.0,
            1.0,0.0,1.0,
            1.0,1.0,0.0,
            1.0,0.0,1.0,
            1.0,1.0,0.0
      }     ,
/* "C"   */
      {     0.0,1.0,0.0,
            1.0,0.0,1.0,
            1.0,0.0,0.0,
            1.0,0.0,1.0,
            0.0,1.0,0.0
      }
}     ;



float op_array[3][8]=
{

      {     0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0
      }     ,/* ASCII "A"   */
      {     0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0
      }     ,/* ASCII "B"   */
      {     0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0
      }       /* ASCII "C"   */
}     ;
```

```
/********** Neural Nets include file : "NN.h" ************/
/* now needs matrix header file to be loaded previously  */

#ifndef __MATRIX__
#include "matrix.h"
#endif __MATRIX__

#define MAX_SYNAPSE 20.0
#define MAX_BIAS 20.0

/* Random number generator: period 4286449341. */
/* Result is unsigned short with uniform distribution: range 0-
65535  */
/* Seeds the same */

#define U2RAND(seed1,seed2) \
(((seed1)*=65421,++(seed1))+((seed2)*=65521,++(seed2)))

/* Random Noise maker for the noisy back prop model - uses
U2RAND to produce */
/* 0-65535 number, shifts range to -32767 to +32768, and divides
by  */
/* 54613.3- to produce range of -0.6 to + 0.6 */
/* 65536 to produce range of -0.5 to + 0.5 */
/* 81920 to produce range of -0.4 to + 0.4 */
/* 109226.6667 to produce range of -0.3 to 0.3 */
/* 163840 to produce range of -0.2 to 0.2 */
/* 218453.3333 to produce range of -0.15 to 0.15 */
/* 436906.6667 to produce range of -0.075 to 0.075 ---
etc,etc,etc*/

#define NOISE(seed1,seed2,Factor) \
((float)((short)(((long)U2RAND(seed1,seed2))-
32768)))/(32768/Factor)

typedef  struct
{           float activation;/* activation of neuron for feed
forward  */
     float errors;/* sum of errors from feedback */
     float bias;/* activation bias */
     float bed;/* bias error derivative for batching */
     float olddbias;/* old delta bias for bias updating */
     float sumsyn;/* Sum of noise affecting this node */
     float rawact; /* Raw activation of a node */
```

```
        short number;/* index for debugging */
}               NEURON;


typedef float SYNAPSE;


typedef struct layer
{               struct layer *prev_layer;/* ptr to prev layer */
        int n_inputs;/* No. of input neurons */
        NEURON *inputs;/* same address as o/p of prev layer */
        SYNAPSE *synapses;/* synapses[n_inputs][n-outputs] */
        SYNAPSE *history;/* prev vals for use in learning */
        SYNAPSE *weds;/* weight error derivatives for batching */
        int number;/* Layer number for debugging */
        int n_outputs;/* No. of output neurons */
        NEURON *outputs;/* same address as i/p of next layer */
        struct layer *next_layer;/* ptr to next layer */
}               LAYER;

typedef struct
{               LAYER *first_layer;
        LAYER *last_layer;
}               NETWORK;


/* Glabal variables - for convenience rather than elegance */

/* Random number seed placeholders */
static unsigned short ranSeed1,ranSeed2;

/* option flags - for features and each extension in this model */
int
useBias,useMomentum,useNoise,useGRA,updateGRA,update,useCGA;

/* floating point learning rate */
float frate;




/*      ********************/

DelayLinePtr DCreate(int);
```

```
void DFree(void);
void noise_calc(NETWORK *);
void npcalc(int,NETWORK *);
void noise_add(NETWORK *);
void zerosyn(NETWORK *);

/********** Addition of Peptidergic noise section
*****************/



/* Function DCreate:   Create the DelayLineLine list
 *  with a size of n structures
 *  returns a Ptr to the start of the list or 0 on error
 */

DelayLinePtr DCreate(n)
int n;
{
      int i,j;
      DelayLinePtr StartPtr,DelayLineList;

      extern char *calloc();


      if (n < 4 )  /* It's rather pointless coming here so bail out */
      {
            fprintf(stderr,"DCreate: Sorry you've got to have at
least 4 lists\n");
            return(0);
      }

      DelayLineList = (DelayLinePtr) calloc(1,sizeof(DelayLine));
/* Create first structure */
      if ( !DelayLineList )          /* Problems with memory */
      {
            fprintf(stderr,"DCreate: Trouble allocating memory for
the DelayLine List\n");
            return(0);
      }
      StartPtr = DelayLineList;
                        /* Keep its address */

      for( i=1;  i< n ; i++)          /* Create the rest and link */
      {
```

```
            DelayLineList->nextPtr = (DelayLinePtr)
calloc(1,sizeof(DelayLine));
            if ( !DelayLineList->nextPtr )          /* Problems
with memory */
            {
                  fprintf(stderr,"DCreate: Trouble allocating
memory for the DelayLine List\n");
                  return(0);
            }
            DelayLineList->nextPtr->prevPtr = DelayLineList;    /*
Point back */
            DelayLineList = DelayLineList->nextPtr;

            /* Initialise each List to zero */
            for (j=0;j < 39 ; j++)
                  DelayLineList->dlist[j] = 0;
      }

      DelayLineList->nextPtr = StartPtr;
      /* Round off the list */
      StartPtr->prevPtr = DelayLineList;
      /* and point back */

      return(StartPtr);
                  /* And return the Ptr */

}
/* DFree()  - Free's the allocation for the delay list */
/* This is a round, linked list so as long as nothing naughty has
been going
 *  on, the external DPtr should hold a valid address which we can
use to free
 *  all elements  */
void DFree()
{
      DelayLinePtr NPtr,SPtr;
      extern DelayLinePtr DPtr;

      SPtr = DPtr;

      do
      {
            NPtr = DPtr->nextPtr  ;
            free(DPtr);
      }           while( NPtr != SPtr);
}
```

```
/* Sum effects of secondary messengers for the network*/
/* Stored in DPtr for use in activ_calc()    */

void  noise_calc(network)
NETWORK *network;
{
      register  int  i,n_inputs;
      register NEURON *output, *end;
      LAYER *layer;

      extern  DelayLinePtr  DPtr;


      if ( !DPtr )              /* First Time here & no Delay list? */
            DPtr = DCreate(DELAYNO);          /*  create
DelayLineNO DelayLine lists   */

      /*loop  forward  through  all  layers  */
      for (   i=1,layer=  network->first_layer;  layer  ;  layer=  layer-
>next_layer)
      {
            n_inputs  =  layer->n_inputs;

            for(; --n_inputs >= 0; i++)                    /*  For
each node */
            {
                  npcalc(i,network);   /* Calculate all the
permutations   */
                  lpcalc(i,network);
            }
      }
      /* Last layer only  ----   calc for the outputs as well*/
      layer  =  network->last_layer;
      output  =  layer->outputs;
      end  =  layer->n_outputs  +  output;
      for ( ; output < end; output++,i++)       /*  Do last layer */
      {
            npcalc(i,network);
            lpcalc(i,network);
      }
```

```
        /* Decrement the DelayLine Pointer for addition into the
 activation function   */
        if (D_DEBUG)
        {
                for(i=0;i < 39;i++)
                        printf("%f,",DPtr->dlist[i]);
                printf("\n");
        }

        DPtr = DPtr->prevPtr;

        /* Zero the synapse sums through the net here */
        zerosyn(network);


        /* call the noise distibutor here */
        /* Noise distrib now moved to function: activ_calc()    */
        /*  noise_add(network);  */

}


void  npcalc(ucnt,network)   /*  */
int  ucnt;
NETWORK *network;
{
        int  i,il,n,cur_layer,layer_num,layer_diff;
        int  n_inputs;
        register  int  nips;
        SYNAPSE *synapses;
        register SYNAPSE  *syn;
        register  float  iactive,sum;
        NEURON   *inputs,*output,  *end;
        register NEURON *ips;
        LAYER *layer;
        float  sumsynapse;

        extern int OLIndex[];                   /* conversion array
for output nodes */
        extern  float  PepStrength;
        extern  DelayLinePtr  DPtr;


        /* This routine is called by noise_calc() - once for each node
in each
          * layer of the network.
```

```
        * Here we have to calculate the 'noise' reaching the node
from the rest of
        * the network - comprised of all the somatic propagations
of all other nodes
        * plus the synaptic leakage of all other nodes
        */

      if (N_DEBUG) printf("Strength = %-1.8f, Node
%d",PepStrength,ucnt);


      if ( ucnt < 16)                                          / *
First  Layer  */
      {
            n=ucnt;
            cur_layer=1;
      }
      if ( ucnt > 15 && ucnt < 31)        /* Second Layer */
      {
            n=ucnt-15;
            cur_layer=2;
      }
      if ( ucnt > 30 && ucnt < 39)        /* Output Layer - use index
* /
      {
            n=OLIndex[ucnt-30];
            cur_layer=3;
      }

      if (N_DEBUG) printf(", node index = %d, layer =
%d\n",n,cur_layer);

      /* n is the node index inside the layer and tells us which
matrix element to use.
        * ie. it's the index number of the current node to be
computed in the layer  */

        /* Start off the P activation at zero for this node in the
delay  list  */
      DPtr->prevPtr->prevPtr->prevPtr->dlist[ucnt]  =  0;

      /* Here we begin to collect P-activations from each node:
        * First we get the Somatic P-activation, calculating it as
the
        * output-activation * av. of its output weights *
Peptidergic  factor
```

```
        * This is derived from the node in question.
        * Then we get the leakage activation which is calculated
from the
        * nodes' input activation * input weights * Peptidergic
factor.
        * Note: LEAKAGE will merely be a proportion of the o/p
activation
        * as ordinarily calulated in back prop. Therefore, Leak is
defined
        * as o/p activation * Leak factor. This can be incorporated
in the
        * model simply
        * Remember - leakage occurs at TARGET nodes
        * LEAK is calculated in function lpcalc()
        */

      /* REMEMBER - WE MUST DO ALL NODES */
      /* loop forward through all layers */
      for ( layer= network->first_layer, layer_num=1; layer ;
layer= layer->next_layer,layer_num++)
      {
            synapses = layer->synapses;
            n_inputs = layer->n_inputs;
            inputs = layer->inputs;
            layer_diff = abs(layer_num - cur_layer);    /* layer
differential */
            sum = 0;          /* Start sum off at a reasonable
number */

            if(N_DEBUG) printf("Layer Differential =
%d\n",layer_diff);

            /* USE INPUT NODES - for first layers */
            for(i=1;--n_inputs >= 0;i++,inputs++)
            {
                  /* Get the 'sign' & magnitude of the activation */
                  sumsynapse = inputs->sumsyn/layer->n_outputs ;

                        /* Get the somatic activation for a node */
                  iactive = inputs->activation * PepStrength *
sumsynapse ;

                  if (N_DEBUG) printf("input %d - activ %f,
sumsynapse  %f\n",n_inputs,inputs->activation,sumsynapse);
```

```
                    /*  we get matrix references to layer 3
transformed here */
                if ( layer_num == 3)
                    il = OLIndex[i];
                else
                    il = i;

                    /* Mult by the distance between the node
and this one */
                    /* and store, delayed by the
distance/diffusion analogy */
                switch(layer_diff)
                {
                case (0): /* do intra-layer activations */
                    DPtr->prevPtr->dlist[ucnt] += iactive *
iMatrix[n][il];

                        /* delay list */
                    break;

                case (1): /* do 1st order extra-layer activation */
                    DPtr->prevPtr->prevPtr->dlist[ucnt] +=
iactive * eMatrix[n][il];

    /* delay list */
                    break;

                case (2): /* do 2nd order extra-layer activation
* /
                    DPtr->prevPtr->prevPtr->prevPtr-
>dlist[ucnt] += iactive * exMatrix[n][il];

    /* delay list */
                    break;
                }
            }       /* End for all inputs */

            /* For the output layer */
            if (layer == network->last_layer)
            {
                output = layer->outputs;
                end = layer->n_outputs + output;
                layer_num++;
                layer_diff = abs(layer_num - cur_layer);    /*
layer differential */
```

```
                    if(N_DEBUG) printf("Layer Differential =
%d\n",layer_diff);

                        /* for each node in the output layer  */
                    for( i=1; output < end ; output++,i++, synapses
+=n_inputs)
                    {
                        /* We have to assume that the sign of the
activation  in output
                         * nodes is positive, after all, there would
be no output if they weren't,
                         * They can't be calculated anyway; and we
can simplify things
                         * by leaving it out as it will only be a
factor of 1 anyway
                         *
                         *  output->sumsyn = 1.0 ;
                         * iactive = output->activation *
PepStrength * output->sumsyn ;
                         * -- see what I mean
                         */

                        /* Get the somatic activation for a node */
                        iactive = output->activation * PepStrength ;

                        if (N_DEBUG) printf("output %d - activ %f,
pos sumsyn \n",i,output->activation);

                        /* ADD section here for Model 3 - - better
emulation  */



                        /*  we get matrix references to layer 3
transformed here */
                        if ( layer_num == 3)
                            il = OLIndex[i];
                        else
                            il = i;

                        /* Mult by the distance between the node
and this one */
                        /* and store, delayed by the
distance/diffusion analogy  */
                        switch(layer_diff)
                        {
```

```
                        case (0): /* do intra-layer activations */
                              DPtr->prevPtr->dlist[ucnt] += iactive
 *  iMatrix[n][il];

                              /* delay list */
                              break;

                        case (1): /* do 1st order extra-layer
 activation  */
                              DPtr->prevPtr->prevPtr->dlist[ucnt]
 += iactive * eMatrix[n][il];

      /* delay list */
                              break;

                        case (2): /* do 2nd order extra-layer
 activation  */
                              DPtr->prevPtr->prevPtr->prevPtr-
 >dlist[ucnt] += iactive * exMatrix[n][il];

      /* delay list */
                              break;
                        }
                  }      /* End for each node in the output layer */
            }      /* End for the output layer */
      }       /* End for each layer */

}      /* exit */

lpcalc(ucnt,network)   /*   */
int ucnt;
NETWORK *network;
{
      int  i,il,n,cur_layer,layer_num,layer_diff;
      int n_inputs;
      register int nips;
      SYNAPSE *synapses;
      register SYNAPSE *syn;
      register float iactive,sum;
      NEURON  *inputs,*output, *end_out;
      register NEURON *ips;
      LAYER *layer;
      float sumsynapse;

      extern int OLIndex[];                  /* conversion array
 for output nodes */
```

```
        extern float PepLeak;
        extern DelayLinePtr DPtr;


        /* This routine is called by noise_calc() - once for each node
in the network.
          * Here we have to calculate the 'noise' reaching the node
from the rest of
          * the network - comprised of the synaptic leakage of all
other nodes
          */

        if (N_DEBUG) printf("Leak = %-1.8f, Node %d",PepLeak,ucnt);


        if ( ucnt < 16)                                          / *
First  Layer  */
        {
              n=ucnt;
              cur_layer=1;
        }
        if ( ucnt > 15 && ucnt < 31)        /* Second Layer */
        {
              n=ucnt-15;
              cur_layer=2;
        }
        if ( ucnt > 30 && ucnt < 39)        /* Output Layer - use index
* /
        {
              n=OLIndex[ucnt-30];
              cur_layer=3;
        }

        if (N_DEBUG) printf(", node index = %d, layer =
%d\n",n,cur_layer);


        /* n is the node index inside the layer and tells us which
matrix element to use.
          * ie. it's the index number of the current node to be
computed in the layer  */

        /* Here's where we get the 'Leakage' activation from each
node  */
        /* Start at layer 2 (of nodes) as there won't be any "Leakage"
from  */
```

```
	/* input nodes */
	for (layer_num=2,layer= network->first_layer; layer ; layer=
layer->next_layer,layer_num++)
	{
		synapses = layer->synapses;
		n_inputs = layer->n_inputs;
		inputs = layer->inputs;
		output = layer->outputs;
		end_out = layer->n_outputs + output;
		layer_diff = abs(layer_num - cur_layer);	/* layer
differential */

		if(N_DEBUG) printf("Layer Differential =
%d\n",layer_diff);

		/*calc p-leak through this layer */
		for( i=1; output < end_out; output++,i++)
		{
			/* all secondary msger activation in this
part emanates from synapses!!!!! */
			/* PEP_LEAK% is leaked from synapses of
the target cell */
			/* and is related to the binding of peptides
by PEP_STRENGTH */
			/* This obviously only works for the last 2
layers */

			iactive = output->rawact * PepLeak;

			if (N_DEBUG) printf("rawact = %f\n",output-
>rawact);

			/* we get matrix references to layer 3
transformed here */
			if ( layer_num == 3)
				il = OLIndex[i];
			else
				il = i;

			switch(layer_diff)
			{
			case (0): /* do intra-layer activations */
				DPtr->prevPtr->dlist[ucnt] += iactive *
iMatrix[n][il];

					/* delay list */
```

```
                        break;

                case (1): /* do 1st order extra-layer activation */
                        DPtr->prevPtr->prevPtr->dlist[ucnt]  +=
 iactive  *  eMatrix[n][il];

        /*  delay  list  */
                        break;

                case (2): /* do 2nd order extra-layer activation
 * /
                        DPtr->prevPtr->prevPtr->prevPtr-
 >dlist[ucnt] += iactive  *  exMatrix[n][il];

        /*  delay  list  */
                        break;
                }
            }
        }
        /*  and  exit  */
}


/* Zero the synapse sums throughout the net */
void  zerosyn(network)
NETWORK *network;
{
        int  n_inputs;
        NEURON *inputs;
        LAYER  *  layer;

        /*loop  forward  through  all  layers  */
        for  (layer=  network->first_layer;  layer  ;  layer=  layer-
 >next_layer)
        {
                n_inputs  =  layer->n_inputs;
                inputs  =  layer->inputs;

                for(;  --n_inputs  >=  0;  )                    /*  For
 each node */
                        (inputs++)->sumsyn  =  0;
        }

}
```

```
/* Add the noise produced by P-activation to the nodes of the
network   */
void  noise_add(network)
NETWORK * network;
{
      /* For each value in DPtr->dlist[i], add in to the activation
cycle of the
        * relevant node in the network. This can be done either by
adding into
        * the actual activation, or by adding a factor to the weights
        */
      int  n_inputs,i;
      SYNAPSE *synapses;
      NEURON  *inputs,*output,*end;
      LAYER *layer;

      if(N_DEBUG) printf("adding noise\n");

                        /*loop forward through all layers */
      for (   i=1,layer= network->first_layer; layer ; layer= layer-
>next_layer)
      {
            n_inputs = layer->n_inputs;
            inputs = layer->inputs;

            for(; --n_inputs >= 0; i++,inputs++)
      /* For each node */
            {
                  inputs->activation += DPtr->dlist[i];           / *
Add  P-activation  */
                  if(N_DEBUG) printf("node %d - noise %f\n",i,DPtr-
>dlist[i]);
            }
      }
      /* Last layer only  ----  calc for the outputs as well*/
      layer = network->last_layer;
      output = layer->outputs;
      end = layer->n_outputs + output;
      for ( ; output < end; output++,i++)                  /* Do last
layer  */
      {
            output->activation += DPtr->dlist[i];      /* Add P-
activation  */
            if(N_DEBUG) printf("node %d - noise %f\n",i,DPtr-
>dlist[i]);
```

```
        }
}

/****** end of peptidergic addition **********/


/* Logistic function for back prop model */

float  logistic(x)
float x ;
{
        extern float exp();

/* Find out if it's over the capabilities of the system */


        if (x > 11.5129)
                return(.99999);
        else
                if (x < -11.5129)
                        return(.00001);


/* or otherwise */
/* Calculate logistic */
                return(((1.0/(1.0 + (float) exp((double) ((-1.0) * x)) ))));


}



/* Create a new network of n_layer synapse layers with
n_neurons_ip[i] input neurons and
n_neurons_op[i] output neurons
for each layer.
returns ptr to network or 0 if out of memory
* /


NETWORK *new_network(n_layer,neurons_ip,neurons_op)
int  n_layer[],neurons_ip[],neurons_op[];

{
```

```
        NETWORK *network;
        int  i,j,n_inputs;
        LAYER  *layer,  *prev_layer=0;
        extern  char  *calloc();
        NEURON *inputs;

        network = (NETWORK *)calloc(1,sizeof(NETWORK));
        if  (!  network)
                return 0 ;

        for  (  i=0;  i  <  n_layer[0];  i++,  prev_layer=layer)
        {
                layer = (LAYER *)calloc(1,sizeof(LAYER));
                if(!  layer)
                        return  0;

                layer->n_inputs  =  neurons_ip[i];
                layer->n_outputs  =  neurons_op[i];
                layer->number  =  i+1;
                if  (BIG_DEBUG)  printf("  layer  %d  allocating...\n",i+1);
                layer->inputs  =  (NEURON  *)calloc(layer-
>n_inputs,sizeof(NEURON));
                if(!  layer->inputs)
                        return  0;

                n_inputs  =  layer->n_inputs;
                inputs  =  layer->inputs;
                for(j=1;j  <=  n_inputs;j++)
                {
                        (inputs++)->number  =  ((i+1)*100)+j;
                        if  (BIG_DEBUG)  printf("  Input  cell  %d
allocated....\n",j);
                }

                if(prev_layer)
                {
                        layer->prev_layer  =  prev_layer;
                        layer->prev_layer->next_layer  =  layer;
                        layer->prev_layer->outputs  =  layer->inputs;
                }
                else
                        network->first_layer  =  layer;

                layer->synapses  =
                (SYNAPSE  *)calloc(layer->n_inputs*layer-
>n_outputs,sizeof(SYNAPSE));
```

```
            if( ! layer->synapses)
                  return 0;

            if(useMomentum || useGRA)
            {
                  layer->history =
                  (SYNAPSE *)calloc(layer->n_inputs*layer-
>n_outputs,sizeof(SYNAPSE));
                  if(! layer->history)
                        return 0;
            }
            else
                  layer->history = 0;


            layer->weds =
            (SYNAPSE *)calloc(layer->n_inputs*layer-
>n_outputs,sizeof(SYNAPSE));
            if( ! layer->weds)
                  return 0;



      }

      layer->outputs = (NEURON *)calloc(layer-
>n_outputs,sizeof(NEURON));
      if(! layer->outputs)
            return 0;

      n_inputs = layer->n_outputs;
      inputs = layer->outputs;
      for(j=1;j <= n_inputs;j++)
      {
            (inputs++)->number = 900+j;
            if (BIG_DEBUG) printf(" output cell %d allocated....\n",j);
      }


      network->last_layer = layer;
      return network;
}



/* Feed activation forward through the network */
```

```
feedforward(network)  /*  */
NETWORK *network;

{
      int  n_inputs,i;
      SYNAPSE *synapses;
      NEURON  *inputs,*output,*end_out;
      LAYER *layer;

/* loop forward through all layers */
      for ( i=16,layer= network->first_layer; layer ; layer= layer-
>next_layer)
      {
            synapses = layer->synapses;
            n_inputs = layer->n_inputs;
            inputs = layer->inputs;
            output = layer->outputs;
            end_out = layer->n_outputs + output;

            /*feed activation forward through this layer */
            for( ; output < end_out; output++, synapses
+=n_inputs,i++)
                  activ_calc(n_inputs,inputs,output,synapses,i);
      }
}


/* calculate activation for feedforward */
activ_calc(n_inputs,inputs,output,synapses,i)    /*  */

register int n_inputs;/* No. of i/p neurons */
register NEURON *inputs;/* vect of i/p neurons */
NEURON *output;/* o/p neuron */
register SYNAPSE *synapses;/* vect of synapses on o/p */
int  i;

{

      register float sum = 0;

            /* Here we assume that the unit has an intrinsic */
            / *    "will" to communicate despite whether the */
            /*    node will be allowed to learn bias or not  */
      sum = output->bias;              /* Delete for a test sometimes */
```

```
        /* feed i/p activation forward by accumulating products */
        while (--n_inputs >= 0)
        {

                /* This bit is for P-activation gathering, and is here
because I'm
                 * trying to save some time, rather than because I'm
lazy.
                 * This variable is zeroed in function: npcalc()
                 */
                /* inputs->sumsyn += *synapses;*/

                if (BIG_DEBUG)
                        printf("*synapses %f i/p activation
%f\n",*synapses,inputs->activation);

                sum += ((*synapses++) * (inputs++)->activation)    ;


        }
        /* save raw activation for leak calculation */
        output->rawact = sum;

        /*if  (useNoise)
                sum  += DPtr->dlist[i];*/
        /*if  (N_DEBUG)  printf("noise  was  %f\n",DPtr->dlist[i]);*/

        output->activation =   logistic(sum);
        if (BIG_DEBUG) printf("cell %d bias %f, calc activation as
%f\n",output->number,output->bias,output->activation);

}



/* feed errors back through all layers of the network*/
feedback(network)  /*  */
NETWORK *network;

{
        int  n_inputs;
        SYNAPSE *synapses, *history, *weds;
        NEURON *inputs, *output, *end;
        LAYER *layer;
```

```
	/*loop back through all layers */
	for(layer = network->last_layer; layer; layer = layer-
>prev_layer)
	{
		/* clear prev errors */
		n_inputs = layer->n_inputs;
		inputs = layer->inputs;
		while(--n_inputs >= 0)
			(inputs++)->errors = 0.0;

		/* feed errors back through this layer */
		n_inputs = layer->n_inputs;
		inputs = layer->inputs;
		output = layer->outputs;
		synapses = layer->synapses;
		weds = layer->weds;
		history = layer->history;
		end = output + layer->n_outputs;


		for(; output < end; output++,synapses += n_inputs,
history +=n_inputs, weds +=n_inputs)

	delta_calc(n_inputs,inputs,output,synapses,history,weds);
	}
}



/* Calculate delta and errors for feedback */

delta_calc(n_inputs,inputs,output,synapses,history,weds)	/* */
register int n_inputs;/* No. of i/p neurons */
register NEURON *inputs;/* vect of i/p neurons */
NEURON *output;/* o/p neuron */
register SYNAPSE *synapses;/* vect of synapses on o/p */
SYNAPSE *history;/* vect of synapse history */
SYNAPSE *weds;/* weight error derivatives */

{
	float weight,noise;/* synapse weight */
	float error,delta;/* unit error and delta terms */
	float dbias,bias;/* for bias calculation */
	float fback,x;/* for feedback */
```

```
        extern float frate,PepFactor,TotFact;/* Learning rate */
        extern unsigned short ranSeed1,ranSeed2;   /* Random
number placeholders */
        extern int update,updateGRA,useGRA ;  /* indicators for
extensions */
        extern DelayLinePtr DPtr;



        /* calculate delta for this unit */
        delta = output->errors * output->activation *
        (1.0 - output->activation);


        if (BIG_DEBUG) printf("cell %d calcd delta %f prev error %f
o/p activation %f\n",output->number,delta,output->errors,output-
>activation);



/* For all input units connecting to this layer */
        while(--n_inputs >= 0 )
        {

/* Continue Calc of previous layer errors */
                fback =  delta * (*synapses);
                inputs->errors += fback;

                if(BIG_DEBUG)
                        printf("n_inputs %d *synapse %f delta %f ip.err %f
\nfedback %f ip.activ %f\n",n_inputs,*synapses,delta,inputs-
>errors,fback,inputs->activation);


                if(BIG_DEBUG) printf("weds was %f ",*weds);

                /* save the weight error deriv for batching */
                *weds += (SYNAPSE) (delta * inputs->activation);

                if(BIG_DEBUG) printf("is now %f\n",*weds);



                /* If update flag is set - do all the  gradient calcs */
                if( update )
```

```
        {


                /* calc new weight : delta-weight = learning rate
* delta * input  */
                weight =  (*weds * frate);

                if(BIG_DEBUG)
                     printf("new delta %f i/p activation %f rate
%f\n",weight,inputs->activation,frate);
                if (BIG_DEBUG && error != 0)
                     printf("%d,%d old weight %f \n",inputs-
>number,output->number,*synapses);




        }          /* end if ( update )*/



        /* common code for update operations */
        if( update)
        {

             /* Add momentum if turned on */

             if (useMomentum)
             {
                  weight += (0.9 * (*history));/* add
momentum to error */
                  if(BIG_DEBUG) printf("old hist is %f new
hist is %f\n",*history,weight);
                  *history = (SYNAPSE)weight; /* save error
for next cycle */

             }


             /* zero the weight error derivative for next batch
if update only */
             if( update) *weds = 0;
```

```
                /* If weights plus random noise are required -
add in here */

                if (useNoise)
                {
                    /* add  noise to new weight    */
                    noise =
NOISE(ranSeed1,ranSeed2,PepFactor);   /* Add total noise    */
                    weight += noise;
                    TotFact += noise;
                    if(N_DEBUG) printf("noise was %f\n",noise);
                }

                /* put weight into synapse */
                *synapses += (SYNAPSE) weight;
                if(BIG_DEBUG) printf("New synapse is
%f\n",*synapses);


                /* Check for over or underflow of synapse value
range */
                if (*synapses > MAX_SYNAPSE )
                    *synapses = MAX_SYNAPSE;/* Limit weight
in case of overflow */
                else if (*synapses <  -MAX_SYNAPSE)
                    *synapses = -MAX_SYNAPSE; /* Same for
underflow  */



            }           /* end if (update ) */



            /* increment weights synapse pointer */
            synapses++;
            /* increment weight error derivative pointer */
            weds++;
            /* increment inputs index */
            inputs++;
            /* increment history if necessary */
            if(useMomentum)  history++;
        }

        /* Calculate the unit bias if turned on */
```

```
        if ( useBias )
        {
              /* calculate bias term */
              /* save bias error derivative for batching */
              output->bed += delta;
              if (update)
              {
                    dbias = (frate*delta) ;
                    if (useMomentum)
                    {
                          dbias += 0.9*output->olddbias;
                          output->olddbias = dbias;
                    }
                    bias = dbias + output->bias;

                    /* zero the bias error derivative for next batch */
                    output->bed = 0;

                    /* Check for over or underflow of bias value range
* /
                    if (bias > MAX_BIAS )
                          bias = MAX_BIAS;/* Limit bias in case of
overflow  */
                    else if (bias <  -MAX_BIAS)
                          bias = -MAX_BIAS; /* Same for underflow
* /

                    if (BIG_DEBUG)
                          printf("%d old bias %f, new bias
%f\n",output->number,output->bias,bias);


                    output->bias = bias;


              }           /* end if update */

        }


}
```

```
/* add a signed pseudo random value to all weights of a network */
/* ditto the bias values of all units */
/* also zeros the network weds and beds */
/* Returns a floating point number from -max/100 to +max/100 */

randomise(network,max,seed)
NETWORK *network;
float max;
long seed;

{
      static unsigned short seed1,seed2 = 65535;
      float div = 32768.0/max;
      register SYNAPSE *weight, *end, *wed, *history;
      LAYER *layer = network->first_layer;
      NEURON *output;
      int n_outputs;
      extern unsigned short ranSeed1, ranSeed2;

      seed1 = seed;
      do
      {
            weight = layer->synapses;
            wed = layer->weds;
            if(useMomentum) history = layer->history;
            if (weight)
            {
                  end = weight + layer->n_inputs * layer-
>n_outputs;
                  do
                  {
                        *weight
=(((short)((long)((U2RAND(seed1,seed2))-32768)))/div)/100.0;
                        if (useMomentum) *history++ = 0;
                        *wed++ = 0;
                        if (BIG_DEBUG) printf("weight
%f\n",*weight);
                        weight++;
                  }
                  while(weight < end);
            }

            output = layer->outputs;
            n_outputs = layer->n_outputs;
            while(--n_outputs >= 0)
```

```
            {
                output->bias =
(((short)((long)((U2RAND(seed1,seed2))-32768)))/div)/100.0;
                output->olddbias = 0;
                (output++)->bed = 0;

            }
        }
        while (layer = layer->next_layer);

        if (BIG_DEBUG)
            printf("Example random number
%f\n",(((short)((long)((U2RAND(seed1,seed2))-
32768)))/div)/100.0);


        ranSeed1 = seed1;
        ranSeed2 = seed2;

}


/*********** END of include file "NN.h" *************/
```

Appendix B: Preliminary Results of Pilot tests

Noise	seed 89	BPN	BPMN	BPL	BPML	BPNL	BPNML	BPXL	BPXML	BPPML
0.00	Learning		1608		1608					
0.05	Rate 0.5				1087	2865	356		1087	1315
0.10					990	1750	393		990	1250
0.15					452	1703	372		452	1305
0.20		1979		11444	715		289	11444	715	1727
0.25		2275	1469	12535	662		152	12535	662	2168
0.30		2237	1491	13506	835		189	13506	835	4709
0.35		2237	1428	13703	558	643	3924	13703	558	13097
0.40		1915	286	11401	383	583	1980	11401	383	
0.45		1438	305		854	1560			854	
0.50		1428	371			1559	1468			
0.55		654	373			1563	139			
0.60		820	396			1121	2331			
0.65		511	296			1598	303			
0.70		473	283			1010	972			
0.75		473	321			962				
0.80		529	220			1250				
0.85		505	445			961				
0.90		472	1018			3479				
0.95		437	522							
1.00		4148	537							
1.05		4174	1136							
1.10		4169	175							
1.15		11559	1000							
1.20			1121							
1.25		11574	160							
1.30			475							
1.35			1184							
1.40			208							
1.45			3448							
1.50			2088							
1.55			409							
1.60			1058							
1.65			1204							
1.70			2571							
1.75			457							
1.80			431							
1.85			1194							
1.90			1519							
1.95			963							
2.00										

Data from Preliminary tests on Noise parameters effect on network performance, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added
-L = Leak ('peptidergic') added
-X = Without Auto-Activation of node in Leak mode
-P = Without Inter-Activation of the Node Plane in Leak Mode

Appendix B: Preliminary Results of Pilot tests

Noise	seed 67	BPN	BPMN	BPL	BPML	BPNL	BPNML	BPXL	BPXML	BPPML
0.00	Learning	8713	789	8713	789					
0.05	Rate 0.5	9720	415	9316	977	2529	119	9316	977	986
0.10		2923	423	9340	533		80	9340	533	1000
0.15		2671	164	9033	712		259	9033	712	995
0.20		3575	168	7866	998		102	7866	998	960
0.25		12740	163	6764	1832		273	6764	1832	963
0.30		5994	801	7196	360	342	262	7196	360	939
0.35		4343	520	8037	309	339	68	8037	309	962
0.40		1969	457	7199	532	342	72	7199	532	1028
0.45		10558	354	12155	763			12155	763	1009
0.50		1118	427	7942	1499			7942	1499	791
0.55		952	330	8683	6921			8683	6921	564
0.60		952	473		19137				19137	739
0.65		948	324		14450				14450	557
0.70		320	412							
0.75			166							
0.80			224		9839				9839	
0.85			450							
0.90			224							
0.95			355							
1.00			523							
1.05			232							
1.10			1134							
1.15			3139							
1.20			987							
1.25			2981							
1.30			1179							
1.35			579							
1.40			1066							
1.45			285							
1.50			312							
1.55			1313							
1.60			309							
1.65			312							
1.70			459							
1.75			486							
1.80			195							
1.85			1542							
1.90			297							
1.95			729							
2.00										

Data from Preliminary tests on Noise parameters effect on network performance, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added
-L = Leak ('peptidergic') added
-X = Without Auto-Activation of node in Leak mode
-P = Without Inter-Activation of the Node Plane in Leak Mode

Noise	seed 43	BPN	BPMN	BPL	BPML	BPXL	BPXML	BPNL	BPNML
0.00	Learning	10784	378	10784	378				
0.05	Rate 0.5	13362	1118	10617	888	10617	888	1848	84
0.10		5508	709	9822	423	9822	423	1888	130
0.15		2909	353	8352	543	8352	543	794	100
0.20		2741	277	6596	513	6596	513	836	77
0.25		1809	320	6181	231	6181	231	1733	35
0.30		1842	107	6539	649	6539	649	1712	212
0.35		804	78	7500	248	7500	248	1021	212
0.40		1039	77	10117	250	10117	250	684	350
0.45		1453	127	8759	575	8759	575	637	112
0.50		910	77	8531	344	8531	344	612	96
0.55		1023	77	9444	862	9444	862	877	108
0.60			53	8164	242	8164	242	640	104
0.65		5599	462	10247	57	10247	57	1198	2599
0.70		5926	129	9446	1140	9446	1140	963	1637
0.75		9978	50	12154	17288	12154	17288	1022	1163
0.80		5579	54	6458	199	6458	199	1074	1563
0.85		3568	86	11592	6473	11592	6473	838	767
0.90		5603	2029		17692		17692	858	2801
0.95			1071				1761	964	
1.00		15610	1432		1761				
1.05			109						
1.10			2218						
1.15			721						
1.20			2041						
1.25			1001						
1.30			1384						
1.35			1417						
1.40			3516						
1.45			1393						
1.50			1124						
1.55			2474						
1.60			463						
1.65			1018						
1.70			1527						
1.75			484						
1.80			416						
1.85			1900						
1.90			548						
1.95			4121						
2.00									

Data from Preliminary tests on Noise parameters effect on network performance, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added
-L = Leak ('peptidergic') added
-X = Without Auto-Activation of node in Leak mode
-P = Without Inter-Activation of the Node Plane in Leak Mode

Appendix B: Preliminary Results of Pilot tests

Noise	seed 89	BPN	BPMN	BPL	BPML	BPNL	BPNML	BPXL	BPXML
0.00	Learning								
0.05	Rate 0.75				553	1598	341		553
0.10					478	1071	388		478
0.15			4080		150	1615	74		150
0.20		1981	1420	7656	184	697	148	7656	184
0.25		1858	1445	8322	201	545	214	8322	201
0.30		2101	1437	8864	288	678	49	8864	288
0.35		2186	285	8776	95	554		8776	95
0.40		1799	253	7298	290	569	2040	7298	290
0.45		1651	280		727	1550	1431		727
0.50		547	391		1379	546	230		1379
0.55		455	254			1549	301		
0.60		440	1418			1589	833		
0.65		437	271			1612	2326		
0.70		440	250			1010	3427		
0.75		581	298		9401	1679			9401
0.80		539	182			1556			
0.85		473	349			1587			
0.90		475	213			1689			
0.95		480	556			1647			
1.00		472	175						
1.05		731	1033						
1.10		4130	270						
1.15		4174	1179						
1.20		2643	1128						
1.25		8652	1181						
1.30		11568	564						
1.35		4181	1182						
1.40		11641	950						
1.45		11593	1120						
1.50			1311						
1.55			1183						
1.60			499						
1.65		16288	1328						
1.70			3506						
1.75			929						
1.80			2833						
1.85			2011						
1.90			2056						
1.95									
2.00									

Data from Preliminary tests on Noise parameters effect on network performance, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added
-L = Leak ('peptidergic') added
-X = Without Auto-Activation of node in Leak mode
-P = Without Inter-Activation of the Node Plane in Leak Mode

Noise	seed 67	BPN	BPMN	BPL	BPML	BPNL	BPNML	BPXL	BPXML
0.00	Learning								
0.05	Rate 0.75	3773	347	6199	600	2528	97	6199	600
0.10		2670	186	6218	669		80	6218	669
0.15		2664	163	6006	362		57	6006	362
0.20		3568	164	5350	698	2061	94	5350	698
0.25		1582	127	4640	410		82	410	410
0.30		1557	153	4948	199	341	38	199	199
0.35		1562	69	4922	148	338	56	148	148
0.40		1123	133	4596	159	337	51	159	159
0.45		1123	507	9795	343	826		343	343
0.50		1122	460	4620	1040			1040	1040
0.55		975	463	5472	1767			1767	1767
0.60		931	1008		19272			19272	19272
0.65		339	1016		16577			16577	16577
0.70		339	605		11307			11307	11307
0.75		338	394						
0.80		1444	217						
0.85		965	463						
0.90		1662	1056						
0.95		960	473						
1.00		298	1026						
1.05			1046						
1.10			1026						
1.15		333	1013						
1.20			219						
1.25			472						
1.30			197						
1.35			1324						
1.40		11670	193						
1.45			1179						
1.50			3762						
1.55			224						
1.60			1310						
1.65			2102						
1.70			321						
1.75			834						
1.80			292						
1.85			405						
1.90			2566						
1.95			188						
2.00									

Data from Preliminary tests on Noise parameters effect on network performance, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added
-L = Leak ('peptidergic') added
-X = Without Auto-Activation of node in Leak mode
-P = Without Inter-Activation of the Node Plane in Leak Mode

Appendix B: Preliminary Results of Pilot tests

Noise	seed 43	BPN	BPMN	BPL	BPML	BPNL	BPNML	BPXL	BPXML
0.00	Learning								
0.05	Rate 0.75	6053	177	7070	348	1740	99	7070	348
0.10		4952	351	6616	488	711	35	6616	488
0.15		2907	330	5656	198	704	42	5656	198
0.20		1837	216	4484	193	1731	68	4484	193
0.25		1981	101	4118	58	706	38	4118	58
0.30		820	76	4411	147	705	68	4411	147
0.35		903	78	5016	190	1404	54	5016	190
0.40		1311	42	6547	185	639	63	6547	185
0.45		903	94	6108	1094	591	77	6108	1094
0.50		823	73	5757	783	595	49	5757	783
0.55		910	47	6392	86	674	51	6392	86
0.60		1437	62	5005	294	608	112	5005	294
0.65		1027	70	6355	44	617	690	6355	44
0.70		1055	78	6437	161	1568	3382	6437	161
0.75		5602	69	8129		1601	12482	8129	
0.80		3558	72	4534	1409	1645	969	4534	1409
0.85		1342	2445	7691	389	763	3638	7691	389
0.90		3568	54		1394	922			1394
0.95		5533	1091		2442	933	232		2442
1.00		3551	1000		338				338
1.05			326						
1.10		11263	357						
1.15		5164	2176						
1.20			389						
1.25			361						
1.30			1334						
1.35			995						
1.40			1872						
1.45			996						
1.50			1051						
1.55			2003						
1.60			994						
1.65			1082						
1.70			999						
1.75			2456						
1.80			2082						
1.85			979						
1.90			1908						
1.95			3459						
2.00									

Data from Preliminary tests on Noise parameters effect on network performance, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added
-L = Leak ('peptidergic') added
-X = Without Auto-Activation of node in Leak mode
-P = Without Inter-Activation of the Node Plane in Leak Mode

Appendix B: Preliminary Results of Pilot tests

Noise	seed 89	BPN	BPMN	BPL	BPML	BPNL	BPNML	BPXL	BPXML
0.00	Learning								
0.05	Rate 1.0		1471		280	1880	396		280
0.10			3944		218	1942	338		218
0.15			2036		101	2750	64		101
0.20		1872	1959	5742	51	566	214	5742	51
0.25		1731	2038	6187	139	527	67	6187	139
0.30		2092	1479	6531	203	885	64	6531	203
0.35		1938	558	6380	77	547	1422	6380	77
0.40		1799	1433	5293	105	440	1985	5293	105
0.45		1800	447		8181	521	1410		8181
0.50		455	543		825	1425	1695		825
0.55		455	307			1502	1122		
0.60		454	238				903		
0.65		637	210			2801	2751		
0.70		580	369		2105	2803	3918		2105
0.75		510	1186		5998				5998
0.80		425	250		887	1579			887
0.85		510	643			1127			
0.90		1192	173						
0.95		1392	1199						
1.00		429	351						
1.05		431	175						
1.10		234	348						
1.15		554	929						
1.20		2647	911						
1.25		268	1678						
1.30		267	255						
1.35		6876	1083						
1.40		2040	1167						
1.45		385	552						
1.50		11574	1181						
1.55		19032	1506						
1.60		9497	353						
1.65		11647	1024						
1.70		4976	1194						
1.75			1460						
1.80			1741						
1.85			3483						
1.90			1103						
1.95									
2.00									

Data from Preliminary tests on Noise parameters effect on network performance, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added
-L = Leak ('peptidergic') added
-X = Without Auto-Activation of node in Leak mode
-P = Without Inter-Activation of the Node Plane in Leak Mode

Appendix B: Preliminary Results of Pilot tests

Noise	seed 67	BPN	BPMN	BPL	BPML	BPNL	BPNML	BPXL	BPXML
0.00	Learning								
0.05	Rate 1.0	3768	264	4656	348		92	4656	348
0.10		2669	162	4660	333		68	4660	333
0.15		2611	150	4487	298		165	4487	298
0.20		2986	102	4003	80	2062	53	4003	80
0.25		1582	102	3428	108	342	68	3428	108
0.30		1558	104	3473	36	339	77	3473	36
0.35		1551	167	3484	32	338	23	3484	32
0.40		1123	132	3337	158		84	3337	158
0.45		336	203	5319	276	706		5319	276
0.50		336	930	3465	2171			3465	2171
0.55		336	442	4072	3460			4072	3460
0.60		910	451		5754				5754
0.65		329	1009		8999				8999
0.70		260	982						
0.75		329	298					19751	
0.80		338	1014		10649				10649
0.85		3157	1017		5690				
0.90		928	289		9775				
0.95		302	1004						
1.00		1151	1074						
1.05		341	984						
1.10		1562	325						
1.15		316	442						
1.20		335	97						
1.25			1173						
1.30			1168						
1.35			296						
1.40		1578	141						
1.45			949						
1.50			569						
1.55			282						
1.60			363						
1.65			235						
1.70			207						
1.75			931						
1.80			1121						
1.85			1262						
1.90			975						
1.95			405						
2.00									

Data from Preliminary tests on Noise parameters effect on network performance, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added
-L = Leak ('peptidergic') added
-X = Without Auto-Activation of node in Leak mode
-P = Without Inter-Activation of the Node Plane in Leak Mode

Appendix B: Preliminary Results of Pilot tests

Noise	seed 43	BPN	BPMN	BPL	BPML	BPNL	BPNML	BPXL	BPXML
0.00	Learning								
0.05	Rate 1.0	5166	330	5255	114	1749	129	5255	114
0.10		3179	130	4995	204	1664	135	4995	204
0.15		1743	74	4337	226	1290	33	4337	226
0.20		3153	44	3519	82	707	60	3519	82
0.25		1982	53	3143	121	709	26	3143	121
0.30		872	44	3482	76	710	33	3482	76
0.35		1034	45	3911	88	670	127	3911	88
0.40		508	200	4834	52	579	33	4834	52
0.45		877	38	5010	708	600	80	5010	708
0.50		895	45	4424	120	549	64	4424	120
0.55		1055	75	4975	118	580	55	4975	118
0.60		1767	92	3780	169	245	63	3780	169
0.65		878	486	4604	1372	590	363	4604	1372
0.70		953	77	4912	539	511		4912	539
0.75		3568	48	6214	19756		212	6214	19756
0.80		2812	85	8882	3176	1125	1017	8882	3176
0.85		9981	340	5690	377	763	3633	5690	377
0.90		1337	47	9775		868	1777	9775	
0.95		891	1082		813	606			813
1.00		3564	1452						
1.05		3635	2150						
1.10		1975	78						
1.15		11702	2013						
1.20		10527	1883						
1.25			2458						
1.30			1893						
1.35		13214	540						
1.40			1207						
1.45		6228	948						
1.50			1465						
1.55			2778						
1.60			1419						
1.65			966						
1.70			983						
1.75			1204						
1.80			527						
1.85			359						
1.90			1120						
1.95			1875						
2.00									

Data from Preliminary tests on Noise parameters effect on network performance, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added
-L = Leak ('peptidergic') added
-X = Without Auto-Activation of node in Leak mode
-P = Without Inter-Activation of the Node Plane in Leak Mode

Noise	Seed 89	BPN	BPMN	BPL	BPML	BPNL	BPNML	BPXL	BPXML
0.00	Learning								
0.05	Rate 1.25		518	17962	205	1741	148	17962	205
0.10				18353	387	1596	232	18353	387
0.15			3840		58	1729	46		58
0.20		1872	3767	4615	104	1711	53	4615	104
0.25		1868	1655	4897	43	545	52	4897	43
0.30		1547	1454	5095	217	569	56	5095	217
0.35		1427	2066	4982	191	521	1536	4982	191
0.40		1440	383	4119	4018	613		4119	4018
0.45		1421	537		9930	492	3789		9930
0.50		1617	319		6201	459			6201
0.55		455	480			1347			
0.60		454	365						
0.65		637	332		12491	1538	2342		12491
0.70		523	1185			1026			
0.75		524	110			1639			
0.80		542	132		10750	1505			10750
0.85		1192	195			2041	541		
0.90		609	177						6233
0.95		428	3565						
1.00		1699	908	10643					
1.05		436	5001						
1.10		436	1113						
1.15		1188	206						
1.20		2609	1199						
1.25		230	1175						
1.30		268	393						
1.35		2197	3405						
1.40		381	219						
1.45		2600	236						
1.50		224	3322						
1.55		2834	1179						
1.60		1752	1330						
1.65		726	1758						
1.70		4983	997						
1.75			1091						
1.80			2845						
1.85			3247						
1.90			1120						
1.95			2848						
2.00									

Data from Preliminary tests on Noise parameters effect on network performance, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added
-L = Leak ('peptidergic') added
-X = Without Auto-Activation of node in Leak mode
-P = Without Inter-Activation of the Node Plane in Leak Mode

Noise	Seed 67	BPN	BPMN	BPL	BPML	BPNL	BPNML	BPXL	BPXML
0.00	Learning								
0.05	Rate 1.25	3752	160	3720	223		63	3720	223
0.10		2635	181	3743	300	2601	74	3743	300
0.15		2554	122	3563	138		44	3563	138
0.20		1159	114	3230	98	2062	120	3230	98
0.25		1558	112	2629	88	342	30	2629	88
0.30		3962	143	2731	85	329	64	2731	85
0.35		1057	85	2707	162	349	34	2707	162
0.40		335	152	3268	142	1423	34	3268	142
0.45		334	59	3144	399			3144	399
0.50		321	98	2938	6156	366		2938	6156
0.55		329	456	3145	7987			3145	7987
0.60		328	986	18248	10555			18248	10555
0.65		137	321		8342				8342
0.70		279	267		14532				14532
0.75		339	1007	18239	4418			18239	4418
0.80		136	1012						
0.85		399	419						
0.90		1593	1000						
0.95		1736	249						
1.00		298	209						
1.05		339	164						
1.10		682	1930						
1.15		298	155						
1.20		338	225						
1.25		312	390						
1.30		7350	655						
1.35		958	523						
1.40			1153						
1.45		11671	419						
1.50		1548	227						
1.55		1527	533						
1.60			279						
1.65		337	204						
1.70			805						
1.75			564						
1.80			205						
1.85			1273						
1.90			280						
1.95			254						
2.00									

Data from Preliminary tests on Noise parameters effect on network performance, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added
-L = Leak ('peptidergic') added
-X = Without Auto-Activation of node in Leak mode
-P = Without Inter-Activation of the Node Plane in Leak Mode

Appendix B: Preliminary Results of Pilot tests

Noise	Seed 43	BPN	BPMN	BPL	BPML	BPNL	BPNML	BPXL	BPXML
0.00	Learning								
0.05	Rate 1.25	4928	211	4201	196	1021	46	4201	196
0.10		2607	129	4053	403	1740	83	4053	403
0.15		1745	72	3555	228	881	112	3555	228
0.20		2466	68	2849	88	709	45	2849	88
0.25		1872	41	2580	38	707	26	2580	38
0.30		605	43	2905	79	642	32	2905	79
0.35		809	41	3317	124	701	66	3317	124
0.40		749	37	3883	53	599	79	3883	53
0.45		878	38	3940	83	610	45	3940	83
0.50		507	36	2987	230	548	38	2987	230
0.55		953	33	4654	77	541	3148	4654	77
0.60		1025		3013	79	253	197	3013	79
0.65		787	360	3808	81	708	697	3808	81
0.70		605	72	3945	75	1174	605	3945	75
0.75		3586	59	4768	8244	369	207	4768	8244
0.80		508	42	2623	11495	1666	282	2623	11495
0.85		508	107	4338	1003	763	696	4338	1003
0.90		6921	337	11484	8644	517	1029	11484	8644
0.95		3478	95		186	916	133		186
1.00		3498	3538			750			
1.05		1443	4295						
1.10		2364	68						
1.15		2759	82						
1.20		9865	960						
1.25		3363	1907						
1.30		13586	1455						
1.35		13577	986						
1.40		5154	2524						
1.45		5365	1074						
1.50			1194						
1.55		11256	1410						
1.60		11259	3366						
1.65			964						
1.70			2116						
1.75			1210						
1.80		11258	1114						
1.85			1967						
1.90			1114						
1.95			2718						
2.00									

Data from Preliminary tests on Noise parameters effect on network performance, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added
-L = Leak ('peptidergic') added
-X = Without Auto-Activation of node in Leak mode
-P = Without Inter-Activation of the Node Plane in Leak Mode

Noise	Seed 89	BPN	BPMN	BPL	BPML	BPNL	BPNML	BPXL	BPXML
0.00	Learning		289		289				
0.05	Rate 1.5		1577	15138	101	1588	73	15138	101
0.10				16004	141	1596	150	16004	141
0.15			2015	16744	31	1797	64	16744	31
0.20		1872	1535	3771	38	1707	169	3771	38
0.25		2185	1530	4044	63	418	49	4044	63
0.30		1618	1433	4177	5697	421	38	4177	5697
0.35		1547	1436	4054	308	440	244	4054	308
0.40		1420	349	3306	3714	545		3306	3714
0.45		1421	143		3772	1422	1697		3772
0.50		683	128		1229	1385	1444		1229
0.55		454	288			1386	2396		
0.60		424	268			1612	3345		
0.65		453	251			1774			
0.70		519	287		282	1085			282
0.75		510	126		11229				11229
0.80		426	82			993			
0.85		475	737				1370		
0.90		436	445						
0.95		527	449			1507			
1.00		482	471						
1.05		1192	942						
1.10		231	1165						
1.15		436	1101						
1.20		253	912						
1.25		566	3479						
1.30		528	1138						
1.35		1127	354						
1.40		327	1135						
1.45		1033	2515						
1.50		338	1084						
1.55		3220	2310						
1.60		1921	1177						
1.65		2704	2500						
1.70		2012	1110						
1.75		8755	520						
1.80		4982	1360						
1.85		19602	1090						
1.90			139						
1.95		4980	1179						
2.00			947						

Data from Preliminary tests on Noise parameters effect on network performance, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added
-L = Leak ('peptidergic') added
-X = Without Auto-Activation of node in Leak mode
-P = Without Inter-Activation of the Node Plane in Leak Mode

Appendix B: Preliminary Results of Pilot tests

Noise	Seed 67	BPN	BPMN	BPL	BPML	BPNL	BPNML	BPXL	BPXML
0.00	Learning	2945	90	2945	90				
0.05	Rate 1.5	3752	133	3096	80	2446	185	3096	80
0.10		2635	148	3121	181	972	75	3121	181
0.15		2609	301	2979	192	2062	163	2979	192
0.20		1537	291	2677	34	339	109	2677	34
0.25		1554	259	2175	88	338	68	2175	88
0.30		1548	163	2187	79	340	35	2187	79
0.35		1954	163	2945	252	461	99	2945	252
0.40		335	51	2875	887		75	2875	887
0.45		219	102	2938	1040	727		2938	1040
0.50		320	1240	2365	9934			2365	9934
0.55		320	277	3028				3028	
0.60		87	1028	16350	8317			16350	8317
0.65		311	450		5213				5213
0.70		261	289						
0.75		160	1031		7027				7027
0.80		162	1465						
0.85		327	999						
0.90		264	308						
0.95		337	434						
1.00		766	1255						
1.05		320	420						
1.10		320	1039						
1.15		284	1036						
1.20		304	261						
1.25		1561	208						
1.30		12834	220						
1.35		5855	309						
1.40		1153	406						
1.45		1062	208						
1.50			716						
1.55		1133	406						
1.60		9482	417						
1.65			236						
1.70			524						
1.75			695						
1.80			208						
1.85			246						
1.90			2845						
1.95			2391						
2.00									

Data from Preliminary tests on Noise parameters effect on network performance, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added
-L = Leak ('peptidergic') added
-X = Without Auto-Activation of node in Leak mode
-P = Without Inter-Activation of the Node Plane in Leak Mode

Appendix B: Preliminary Results of Pilot tests

Noise	Seed 43	BPN	BPMN	BPL	BPML	BPNL	BPNML	BPXL	BPXML
0.00	Learning	3236	184	3236	184				
0.05	Rate 1.5	4925	141	3596	246	1019	35	3596	246
0.10		2604	64	3356	458	794	112	3356	458
0.15		1747	53	2971	105	880	48	2971	105
0.20		1769	84	2401	47	709	45	2401	47
0.25		1238	163	2250	346	719	31	2250	346
0.30		829	50	2328	91	668	39	2328	91
0.35		639	43	2804	75	642	50	2804	75
0.40		691	30	3278	60	580	24	3278	60
0.45		708	35	3459	55	627	157	3459	55
0.50		327	37	2398	350	545	32	2398	350
0.55		594	33	2945	41	614	33	2945	41
0.60		504	40	2418	47	624	47	2418	47
0.65		418	2789	3182	210	237	219	3182	210
0.70		600	77	3353	2973	237		3353	2973
0.75		923	54	4277	10003	338	612	4277	10003
0.80		3508	47	5526	113	761	633	5526	113
0.85		14283	38	5675			846	5675	15242
0.90		1455	75	12284	15242	3184	361	12284	
0.95		3565	2576		107				107
1.00		2221	5905			883			
1.05		2414	1329						
1.10		1080	4721						
1.15		2117	2152						
1.20		2760	524						
1.25		1299	2621						
1.30		2761	1034						
1.35		5167	1121						
1.40		10511	2756						
1.45		7037	1473						
1.50		4170	1047						
1.55		8820	2206						
1.60		4154	2802						
1.65		5342	3437						
1.70		6230	1369						
1.75		11259	1263						
1.80		6228	534						
1.85		11260	1957						
1.90			1982						
1.95		11258	1094						
2.00									

Data from Preliminary tests on Noise parameters effect on network performance, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added
-L = Leak ('peptidergic') added
-X = Without Auto-Activation of node in Leak mode
-P = Without Inter-Activation of the Node Plane in Leak Mode

Appendix C: Results of Final tests

Learning Rate	Seed 43	BP	BPM	BPN	BPMN	BPL	BPML
5			9540		1693		6617
10			6400		510		2822
15			4527		777		2275
20			3611		361	18748	1827
25			3020	1740	330	15001	1320
30		18090	2486	1868	313	12505	481
35		15487	2079	1738	96	10721	310
40		13534	1754	1918	77	9379	273
45		12010	859	1842	77	8334	284
50		10784	378	1842	107	7500	248
55		9779	571	1844	95	6816	266
60		8944	504	879	61	6249	295
65		8231	820	879	44	5775	206
70		7611	268	1726	44	5367	154
75		7062	514	820	76	5016	190
80		6579	197	1709	75	4715	159
85		6145	182	943	53	4450	136
90		5758	257	891	67	4260	181
95		5406	315	866	53	4088	196
100		5072	291	872	44	3911	88
105		4769	241	866	45	3783	65
110		4504	205	1286	47	3683	37
115		4262	240	797	43	3562	40
120		4040	229	829	43	3423	114
125		3845	180	605	43	3317	124
130		3683	210	605	60	3269	120
135		3555	192	819	54	3155	97
140		3447	220	836	86	3032	84
145		3344	243	866	53	2918	54
150		3236	184	829	50	2804	75
155		3124	58	832	51	2667	39
160		3024	176	797	34	2511	39
165		2965	40	642	38	2505	45
170		2925	48	640	18	2434	49
175		2868	57	866	27	2360	64
180		2792	112	866	98	2321	70
185		2708	137	835	60	2289	65
190		2626	159	835	59	2256	68
195		2545	119	835	59	2212	177
200		2465	306	835	67	2172	156

Data from Final tests on effect of Learning Rate on Noise parameters, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added (at 0.3 level)
-L = Leak ('peptidergic') added (at 0.35 level)

Appendix C: Results of Final tests

Learning Rate	Seed 43	BP	BPM	BPN	BPMN	BPL	BPML
205		2385	545	835	86	2137	67
210		2303	1412	835	37	2058	34
215		2233	2378	835		1990	46
220		2221	2892	834	1324	1909	65
225		2307	2753	636	2548	1827	48
230		2270	4446	691		1768	51
235		2219	4624	691		1716	65
240		2165	4331	688		1668	80
245		2097	2698	633	73	1618	146
250		2003	31	637	31	1583	111
255		1980	34	637	32	1563	71
260		1941	26	636	30	1531	40
265		1898	1654	706	43	1480	79
270		1848	5667	1010	30	1432	142
275		1867	8371	618	26	1419	55
280		1805	10643	831	32	1350	82
285		1801	10428	692	34	1385	195
290		1670	10007	695	23	1384	223
295		1630	8343	689	47	1373	383
300		1565	6085	831	51	1348	34
305		1558	3400	693	19	1290	49
310		1660	64	831		1155	30
315		1559	47	830		1241	28
320		1534	38	689		1170	23
325		1621	25	796		1175	35
330		1514	34	804		1198	53
335		1399	111	831		1260	23
340		1512	1215	830		1100	41
345		1450	2643	830		1149	15110
350		1330	3701	804		968	
355		1303	5900	690		860	
360		1551	5846	796		844	7192
365		1279	6883	831		865	40
370		1267	7852	804		787	3518
375		1361	8832	906		947	49
380		1250	9789	638		965	11458
385		1193		641		912	3127
390		1372		638		1000	25
395		1180		322		892	3096
400		1042		611		814	

Data from Final tests on effect of Learning Rate on Noise parameters, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added (at 0.3 level)
-L = Leak ('peptidergic') added (at 0.35 level)

Appendix C: Results of Final tests

Learning Rate	Seed 43	BP	BPM	BPN	BPMN	BPL	BPML
405		1298		794		864	
410		1008		640		918	
415		1108		797		1034	
420		1170		794		922	
425		1043		796		870	
430		1291		699		846	
435		991		797		783	
440		1182		508		828	
445		953		507		817	
450		929		624		774	
455		1243		833		715	
460		1204		800		748	
465		874		512		923	
470		1118		826		583	
475		1106		881		551	
480		888		1022		760	
485		865		665		901	
490		1011		794		876	
495		846		925		894	
500		1035		507		867	
505		953		508		876	
510		1096		1031		866	
515		1118		796		957	
520		924		323		957	
525		1017		323		892	
530		1093		614		893	
535		923		635		545	
540		1054		833		545	
545		1343		625		538	
550		951		507		465	
555		978		323		526	
560		1019		814		584	
565		928		605		754	
570		918		323		679	
575		950		606		867	
580		1052		505		894	
585		1055		322		880	
590		832		545		865	
595		1039		640		873	
600		924		786		894	

Data from Final tests on effect of Learning Rate on Noise parameters, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added (at 0.3 level)
-L = Leak ('peptidergic') added (at 0.35 level)

Learning Rate	Seed 43	BP	BPM	BPN	BPMN	BPL	BPML
605		1012		540		899	
610		1026		413		871	
615		932		796		823	
620		966		323		780	
625		939		796		755	
630		801		641		633	
635		861		322		588	
640		881		613		496	
645		1004		817		682	
650		789		322		668	
655		1819		604		248	
660		872		322		428	
665		1101		323		728	
670		1620		636		723	
675		1553		323		604	
680		1244		605		153	
685		1176				94	
690		1205		227		487	
695		1636		689		450	
700		1571		249		515	
705		1670		475		520	
710		1565		552		471	
715		1558		599		480	
720		1510		598		116	
725		1576				122	
730		1502		599		433	
735		1426		755		740	
740		1190		249		27	
745		1117		822		460	
750		1368		225		164	
755		676		601		501	
760		738		752		497	
765		265		322		677	
770		634		499		629	
775		1547		506		623	
780		744				96	
785		1012		505		264	
790		1140		802		261	
795		1382		669		471	
800		898		702		139	

Data from Final tests on effect of Learning Rate on Noise parameters, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added (at 0.3 level)
-L = Leak ('peptidergic') added (at 0.35 level)

Appendix C: Results of Final tests

Learning Rate	Seed 43	BP	BPM	BPN	BPMN	BPL	BPML
805		971				412	
810		1289		607		359	
815		1369		604		58	
820		1273		317		433	
825		1850		344		487	
830		1307		593		394	
835		531		613		416	
840		1442		1309		541	
845		850		583		856	
850		815				469	
855		1331		301		1139	
860		1073		695		166	
865		2694		552		306	
870		1369		621		579	
875		1370				297	
880		1084				92	
885		1352				299	
890		1174				536	
895		1131				480	
900		1245		705		271	
905		1012				435	
910		1694		507		27	
915		742				153	
920		1043		550		55	
925		1666				411	
930		2263		1320		187	
935		2083				844	
940		2392				40	
945		2189		960		56	
950		2102		323		143	
955		2809		817		719	
960		5053				270	
965		2406				108	
970		2686				775	
975		3215				318	
980		7913				2120	
985		13646				101	
990		2483				511	
995		3956				3504	
1000		3472					

Data from Final tests on effect of Learning Rate on Noise parameters, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added (at 0.3 level)
-L = Leak ('peptidergic') added (at 0.35 level)

Appendix C: Results of Final tests

Learning Rate	Seed 43	BP	BPM	BPN	BPMN	BPL	BPML
1005		3614					
1010		6896					
1015		3807					
1020		4373					
1025		4747					
1030		4924					
1035		4671					
1040		5677					
1045		6029					
1050		6457					
1055		6445					
1060		10075					
1065		6775					
1070		7030					
1075		6791					
1080		7198					
1085		7341					
1090		7564					
1095		7590					
1100		5099					
1105		7372					
1110		7840					
1115		10539					
1120		10718					
1125		7990					
1130		6585					
1135		8214					
1140		8432					
1145		6632					
1150		6087					
1155		8606					
1160		8819					
1165		7224					
1170		9007					
1175		9022					
1180		8510					
1185		9180					
1190		10446					

Data from Final tests on effect of Learning Rate on Noise parameters, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added (at 0.3 level)
-L = Leak ('peptidergic') added (at 0.35 level)

Appendix C: Results of Final tests

Learning Rate	seed 89	BP	BPM	BPN	BPMN	BPL	BPML
5					1623		10691
10					1716		4984
15			19708	1994	1438		3245
20			14480	2024	1513		2432
25			9953	2773	1427		1818
30			6179	2276	1466		1427
35			4123	2276	1469		1097
40			2507	2102	1447	17447	879
45			2211	2276	1490	15366	721
50			1608	2237	1491	13703	558
55			1441	2276	1430	12348	463
60			1245	2102	1414	11223	388
65			1058	2101	1437	10274	251
70			969	2102	1429	9467	210
75			865	2101	1445	8776	95
80			719	2101	2059	8170	113
85			622	2101	1535	7636	97
90			568	2101	1522	7167	99
95			544	2091	2042	6750	108
100			582	2092	2038	6380	77
105			636	2092	1525	6056	77
110			588	2092	1401	5755	111
115			446	2092	1430	5460	141
120			378	2102	1430	5229	181
125			352	1547	1454	4982	191
130			341	1649	1403	4777	194
135			329	1635	1506	4560	193
140			322	1681	1456	4395	207
145			292	1680	2017	4184	258
150			289	1618	1433	4054	308
155		19525	323	1581	1390	3879	433
160		18933	467	1616	1357	3706	178
165		18380	1014	1616	1418	3594	85
170		17858	1680	1621	1480	3445	117
175		17377	1629	1620	1378	3290	96
180		16917	1587	1634	1442	3178	98
185		16471	1401	1621	1393	3126	97
190		16051	1198	1617	1374	3042	112
195		15578	1078	1578	1375	2935	189
200		15146	941	1575	1388	2858	278

Data from Final tests on effect of Learning Rate on Noise parameters, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added (at 0.3 level)
-L = Leak ('peptidergic') added (at 0.35 level)

Appendix C: Results of Final tests

Learning Rate	seed 89	BP	BPM	BPN	BPMN	BPL	BPML
205		14748	1064	1575	1985	2819	625
210		14368	490	1716	1461	2701	782
215		14003	265	1577	1472	2591	930
220		13658	194	1700	1385	2493	996
225		13330	186	1436	1467	2402	1169
230		13025	334	1436	1367	2321	1387
235		12725	188	1588	2129	2261	1447
240		12456	184	1552	1425	2256	1729
245		12213	210	1575	1368	2316	1489
250		11964	182	1575	1370	2257	1664
255		11743	199	1597		2093	2036
260		11521	250		1460	2000	2122
265		11317	512	1489	1968	1980	1937
270		11107	790	1964	1925	1994	1613
275		10910	376	1828	1373	1973	1365
280		10742	174	1548	1459	1915	1193
285		10556	192	1579	2005	1977	1118
290		10371	2263	1617	1347	1747	1143
295		10204	5349	1578	1415	1692	1457
300		10031	6041	1434	1460	1668	1692
305		9863	6980	1679	1464	1735	1799
310		9729	5379	1470	1460	1957	3899
315		9604	3575	1711	1366	1804	1724
320		9428	987	1430	1320	1742	1482
325		9230	92	1423	1365	1740	7510
330		9039	72	1496		1493	349
335		8991	3107	1547	1934	1630	85
340		8807	5224	1547	1471	1555	2811
345		8731	5542	1426	1809	1608	
350		8547	9537	1929	1409	1578	51
355		8349	10742	2172	1987	1542	85
360		8376	12497	1431		1244	46
365		8151	13744	1432	290	1340	41
370		8057	13637	1432	1199	1976	23
375		7985	326	1644		1372	60
380		7747	173	1742	1212	1095	67
385		7835	220	1443	1355	1389	27
390		7553	673	1741	1442	4015	39
395		7547	812	1934	2904	1437	16751
400		7484	2105	1621		1403	

Data from Final tests on effect of Learning Rate on Noise parameters, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added (at 0.3 level)
-L = Leak ('peptidergic') added (at 0.35 level)

Learning Rate	seed 89	BP	BPM	BPN	BPMN	BPL	BPML
405		7336	6254	1608		3023	
410		7242	7881	1846		2016	
415		7018	8057	1936		855	
420		6927				825	
425		6836	5013			1456	
430		6816	190	1935		1432	
435		6695	6685	1547		1624	
440		6573	617	1549		2199	
445		6511		1426		2681	
450		6516		1545		2402	
455		6441		1934		3179	
460		6381		1577		5196	
465		6331	11164	1934		2443	
470		6099	577	1934		3602	
475		6157	9733	1428		3796	
480		6175		1538		5105	
485		6191	15824	1497		5863	
490		6005	198	1435		9577	
495		5775	192	1538		7222	
500		5708		1479		17429	
505		6088	9760	1491		12405	
510		5852				18230	
515		5609	6227	1915			
520		5601		1472			
525		5559		1491			
530		5735		1418			
535		5584		1619			
540		5455		1488			
545		5382		1491			
550		5264		1414			
555		5352		2103			
560		5104		1522			
565		5070		1419			
570		5109		1492			
575		5050		1491			
580		4847		1473			
585		4854		1635			
590		4596		1934			
595		4787		1468			
600		4887		1418			

Data from Final tests on effect of Learning Rate on Noise parameters, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added (at 0.3 level)
-L = Leak ('peptidergic') added (at 0.35 level)

Learning Rate	seed 89	BP	BPM	BPN	BPMN	BPL	BPML
605		4942		1934			
610		4718		1657			
615		4776		1543			
620		4376		1545			
625		4547		1414			
630		4701		1491			
635		4604		1543			
640		4398		1543			
645		4507		1543			
650		4448		1414			
655		4446		1425			
660		4472		1550			
665		4162		1642			
670		4410		1518			
675		4337		1506			
680		4203		1489			
685		4119		1479			
690		4343		1491			
695		4696		1521			
700		4034		1543			
705		4779		1468			
710		4539		1544			
715		4093					
720		4385		1441			
725		5321		1518			
730		4743		1715			
735		4096		1469			
740		4709		1421			
745		4066		1467			
750		4498		1418			
755		5526		1436			
760		4038		1545			
765		5181		3927			
770		3909		1458			
775		5566		1641			
780		6283		1715			
785		6188		1451			
790		5557		1405			
795		5464		2105			
800		5245		1435			

Data from Final tests on effect of Learning Rate on Noise parameters, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added (at 0.3 level)
-L = Leak ('peptidergic') added (at 0.35 level)

Learning Rate	seed 89	BP	BPM	BPN	BPMN	BPL	BPML
805		10190					
810		6913		1549			
815		5882		1470			
820		4241		1540			
825		4053		1414			
830		7493		1491			
835		8824		1461			
840		3989		1414			
845		4837		1548			
850		18045		1499			
855		11898		1471			
860		8025		1468			
865		3239		5000			
870		7024		1518			
875		3911		1426			
880		4819		1415			
885		4939		1474			
890		5007		1491			
895		3015		1418			
900		4187		1593			
905		2708		1938			
910		7814		1544			
915		6756		1485			
920		19554		1416			
925		6533		1472			
930		9608		1915			
935		6203		3787			
940		12840		1431			
945		9552		2117			
950		18464		1539			
955		3134		1420			
960		4696		1606			
965		11207		1475			
970		2294		1418			
975		11806		1469			
980		13389		2015			
985		11538		1520			
990		4567		1469			
995				1457			
1000							

Data from Final tests on effect of Learning Rate on Noise parameters, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added (at 0.3 level)
-L = Leak ('peptidergic') added (at 0.35 level)

Learning Rate	seed 67	BP	BPM	BPN	BPMN	BPL	BPML
5			9005		1970		9379
10			4642		275		4028
15			3015		202		2792
20			2096	7279	275	18322	2031
25		17527	1512	12762	263	14917	1475
30		14587	1230	6675	119	12653	1030
35		12489	1082	6675	160	11043	723
40		10915	1076	6667	166	9842	542
45		9692	917	6643	166	8886	406
50		8713	789	5994	801	8037	309
55		7910	775	3932	705	7233	231
60		7241	743	5917	160	6503	174
65		6677	680	1554	60	5887	216
70		6195	610	1554	104	5361	202
75		5779	537	1557	153	4922	148
80		5414	458	1558	153	4576	189
85		5090	379	1555	127	4240	182
90		4803	324	1554	168	3937	199
95		4551	307	1558	153	3683	180
100		4327	293	1558	104	3484	32
105		4124	279	4795	127	3254	32
110		3940	253	1558	127	3092	138
115		3775	223	1558	142	2946	206
120		3630	190	1551	126	2926	222
125		3488	162	3962	143	2707	162
130		3352	146	1554	126	2846	418
135		3244	130	6017	99	2971	159
140		3134	116	2016	210	2700	98
145		3032	103	1558	163	2550	44
150		2945	90	1548	163	2945	252
155		2855	81	1550	165	2996	35
160		2732	78	1596	102	2997	62
165		2691	79	3933	90	2903	125
170		2580	83	1543	219	2845	78
175		2505	85	5024	322	2864	131
180		2485	85	1990	276	2872	131
185		2433	82	2953	267	2828	56
190		2374	81	5023	283	2533	34
195		2318	79	1488	1026	2542	33
200		2251	69	336	319	2547	28

Data from Final tests on effect of Learning Rate on Noise parameters, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added (at 0.3 level)
-L = Leak ('peptidergic') added (at 0.35 level)

Learning Rate	seed 67	BP	BPM	BPN	BPMN	BPL	BPML
205		2174	55	1188	343	1724	28
210		2138	50	336	320	1638	30
215		2103	48	336	597	1576	19
220		2041	51	1964	1110	1518	17
225		1957	48	336	1113	1563	19
230		1948	46	336	323	1856	23
235		1888	44	337	320	2650	30
240		1855	41	334	585	2395	15
245		1808	48	348	390	2170	5899
250		1751	37	1519	1252	2030	3713
255		1690	46	1548	1031	1357	5314
260		1686	23	1156	1103	1333	7390
265		1686	30	334	965	1329	6909
270		1688	35	1064	1015	1293	10444
275		1664	35	334	1262	1207	15985
280		1636	44	337	1022	1205	10423
285		1610	50	335	653	2702	16607
290		1599	75	838	413	2629	9602
295		1572	437	785	1346	2214	9715
300		1551	787	1154	85	1904	9326
305		1539	1041	1543	3578	1380	8089
310		1496	1338	336	798	2117	9749
315		1518	1624	723	1075	1952	7937
320		1501	1917	849	1057	2059	6919
325		1496	1414	321	1527	2189	13009
330		1421	1017	1517	466	1735	
335		1391	1707	243	1063	1919	16933
340		1366	3052	633	773	1326	25
345		1349	3259	930	790	1104	25
350		1334	3611	236	1042	1359	7671
355		1319	3717	704	976	1181	
360		1285	3909	236	648	989	2872
365		1280	4099	1309	847	1334	934
370		1218	4300	201	2321	1282	8678
375		1235	4475	245	1013	1704	
380		1225	4612	165	2472	1701	
385		1229	2631	335	1542	2197	
390		1208	4915	268	964	992	9085
395		1205	5491	329	1053	1501	16564
400		1170	3320	235	1160	550	9982

Data from Final tests on effect of Learning Rate on Noise parameters, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added (at 0.3 level)
-L = Leak ('peptidergic') added (at 0.35 level)

Learning Rate	seed 67	BP	BPM	BPN	BPMN	BPL	BPML
405		1182	6901	277	1164	521	1074
410		1051	6144	262	2163	569	1815
415		1135	5544	261	1032	735	12150
420		1114	13943	292	2050	671	1018
425		1092		235	1199	431	8661
430		1071	19519	236	1479	587	9165
435		1063		273	1187	723	12399
440		1042	9005	235	2267	601	5093
445		1018	5876	334	1057	632	10463
450		1008	5909	276	1043	887	6477
455		990	7868	275	1912	1523	
460		1066	8626	334	1928	1509	12624
465		974		322	1516	1445	2188
470		1005		237	1388	474	9649
475		1061	4078	337	1350	1523	584
480		1119		337	2409	1277	14129
485		1115		262	757	616	14988
490		1004		235	5950	792	11911
495		1063		333	19512	746	3159
500		1035	9332	235	8000	541	254
505		1046		192	4415	524	14133
510		1179	5789	256	3470	911	19007
515		1026	7746	320	1132	998	16101
520		916		718	2389	928	16523
525		1299		340	4768	847	15298
530		746		269	1031	1289	
535		915		581		1470	
540		752	9603	1027	2102	1324	
545		790		379	9670	455	
550		725		735		1376	
555		788	12543	950	3424	710	
560		962		509		896	
565		886	5709	1139	3872	520	
570		790		962	1468	755	
575		939		722	2274	742	
580		861		357		540	
585		939		827	4686	761	
590		643		733	2096	618	
595		1051		510	5009	690	
600		708		384		538	

Data from Final tests on effect of Learning Rate on Noise parameters, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added (at 0.3 level)
-L = Leak ('peptidergic') added (at 0.35 level)

Learning Rate	seed 67	BP	BPM	BPN	BPMN	BPL	BPML
605		822		113	4244	429	
610		687		1103	6496	771	
615		752		722		747	
620		1901		304		1168	
625		241		960		1262	
630		807		237	4490	1024	
635		717		1540		230	
640		1041		362	4708	219	
645		2033		648	5424	315	
650		956		327		675	
655		4563		950	6671	126	
660		5491		270	1641	139	
665		1885		358	4482	797	
670		598		754		330	
675		7507		936	4196	492	
680		739		956	5920	243	
685		4149		830		319	
690		625		1363	3916	702	
695		1130		1259		301	
700		1694		1205	5517	186	
705		2312		950		947	
710		2213		835	6971	1235	
715		2782		957	12915	389	
720		2953		343	948	520	
725		470		322		581	
730		750		708	1000	626	
735		1414		425	5569	530	
740		687		957	13712	697	
745		2753		211	8969	848	
750		2444		1171	18363	486	
755		1291		992	3290	610	
760		2605		932	13417	592	
765		562		950	4514	1257	
770		2400		1181		1260	
775		1344		957	18359	660	
780		882		508		922	
785		4588		1066	748	514	
790		3748		327	4711	541	
795		3568		708	11521	210	
800		2613		473	12644	850	

Data from Final tests on effect of Learning Rate on Noise parameters, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added (at 0.3 level)
-L = Leak ('peptidergic') added (at 0.35 level)

Learning Rate	seed 67	BP	BPM	BPN	BPMN	BPL	BPML
805		1552		704	10124	704	
810		2665		910	10500	294	
815		6248		237	14862	493	
820		3272		713		438	
825		4297		722	19663	250	
830		4719		628	3836	170	
835		4011		479		95	
840		2861		2054		122	
845		6441		712	8212	452	
850		5335		332		150	
855		7502		666	6429	165	
860		2644		194	832	156	
865		11513		1200	1534	326	
870		11259		228		423	
875		3768		1170		417	
880		1358		931		589	
885		1713		909	6375	1250	
890		9154		798		428	
895		5974		530		119	
900		7223		764	1768	278	
905		5796		321	8849	668	
910		9530		931	2939	480	
915		5049		838	6031	158	
920		1824		1026		594	
925		321		667		515	
930		5709		961		4152	
935		645		2115	12328	516	
940		6805		731	10532	335	
945		6008		795		2581	
950		12100		1124	11345	4896	
955		4633		909		1055	
960		12888		466		980	
965		1432		321	1917	1519	
970				360		666	
975		14511		948		2915	
980		6870		667		219	
985		5602		935		1094	
990		8430		479		694	
995		14149		700	8905	1236	
1000							

Data from Final tests on effect of Learning Rate on Noise parameters, number represents iteration of convergence.

KEY:

BP = Back Propagation base model
-M = Momentum term added
-N = Random Noise added (at 0.3 level)
-L = Leak ('peptidergic') added (at 0.35 level)

Appendix D:
Full graphs of Back Propagation Data

The following graphs are full representations of those shown abbreviated in Chapter 5. These are based directly on the data in Appendix C.

The figures shown in this appendix are as follows:

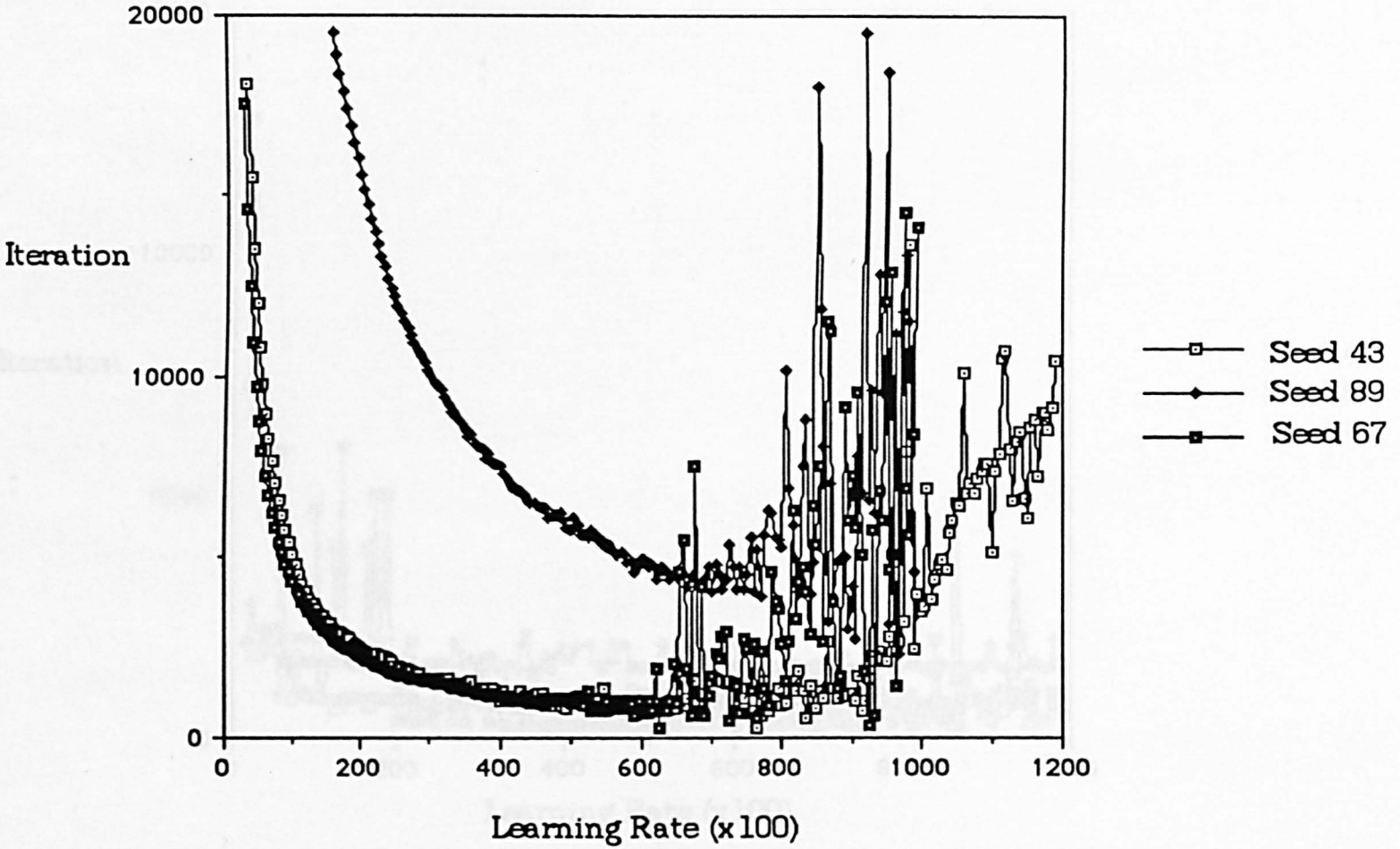

Figure D.1. Base Back Propagation model using three different initial random seed settings.

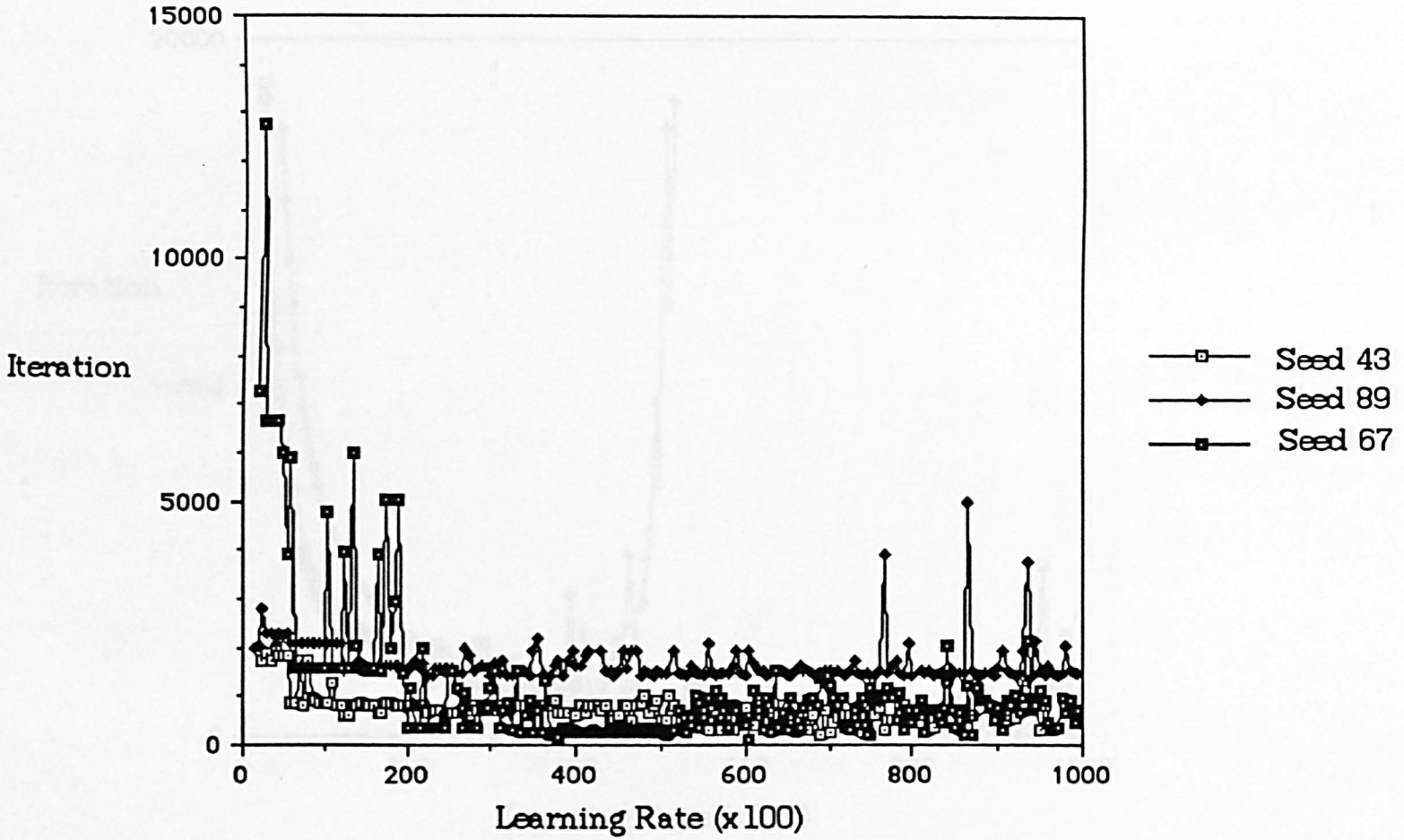

Figure D.2. Back Propagation model with random noise added, using three different initial random seed settings.

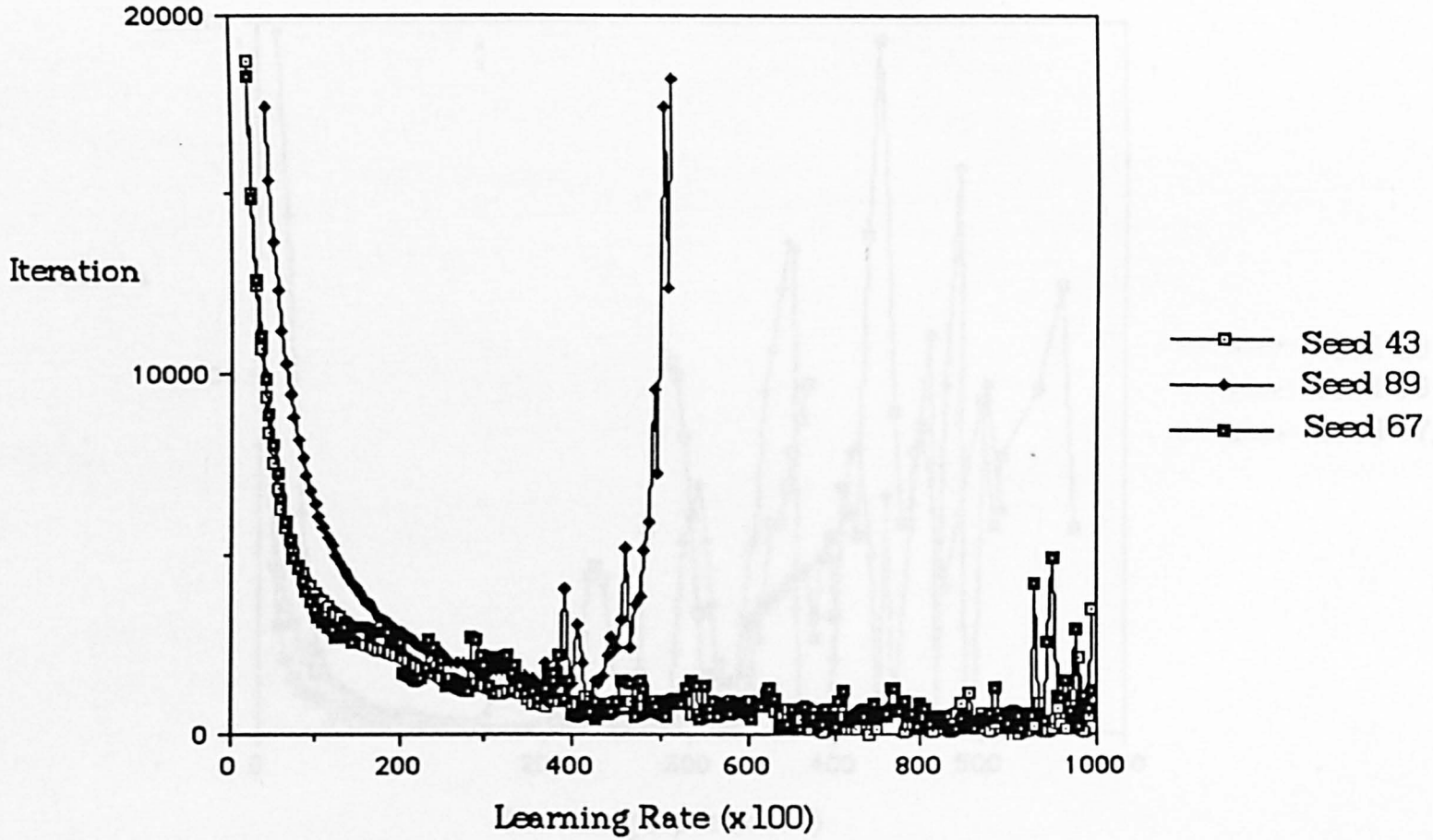

Figure D.3. Back Propagation model with 'Leak' activation added, using three different initial random seed settings.

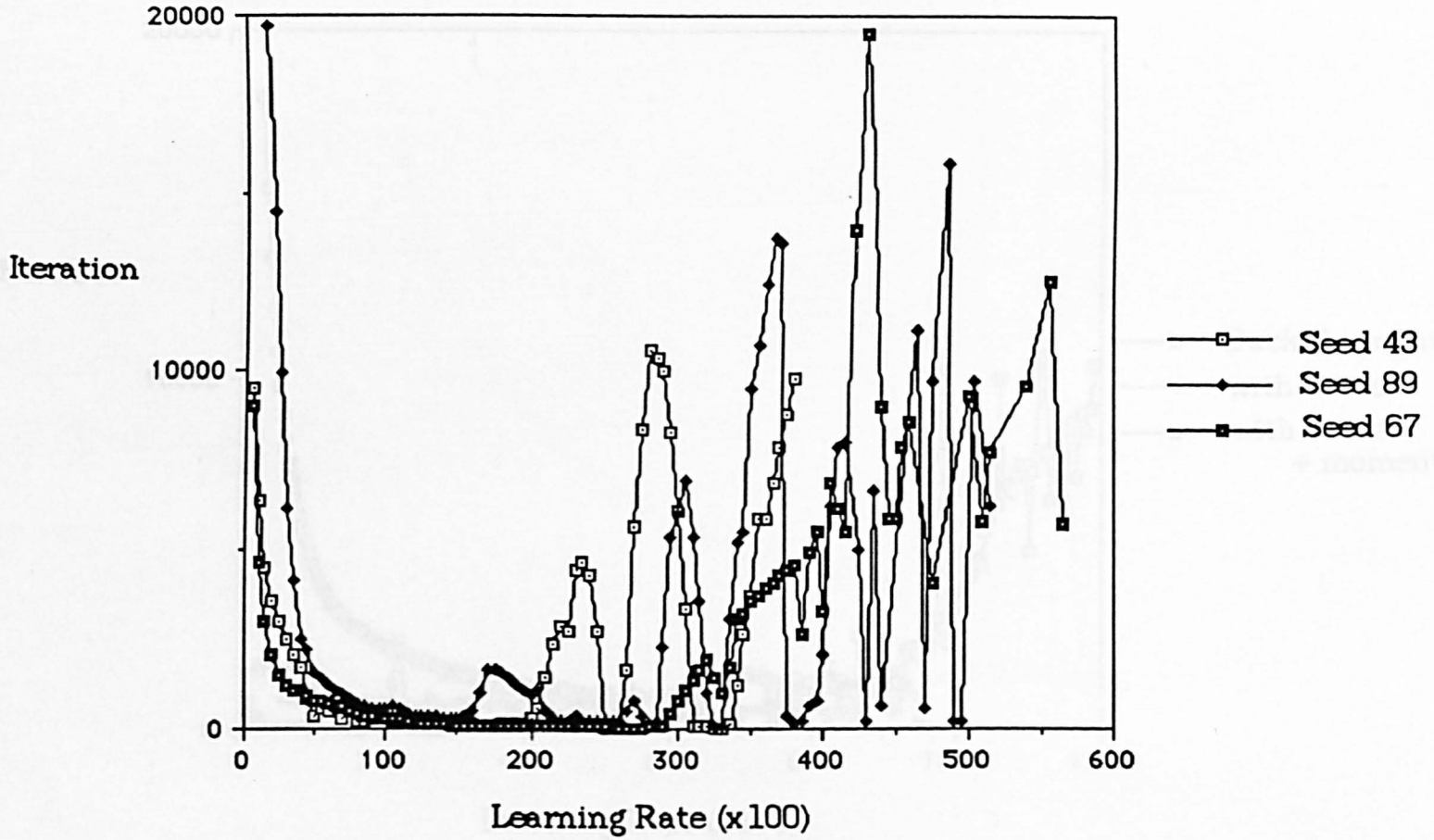

Figure D.4. Back Propagation model with Momentum added, using three different initial random seed settings.

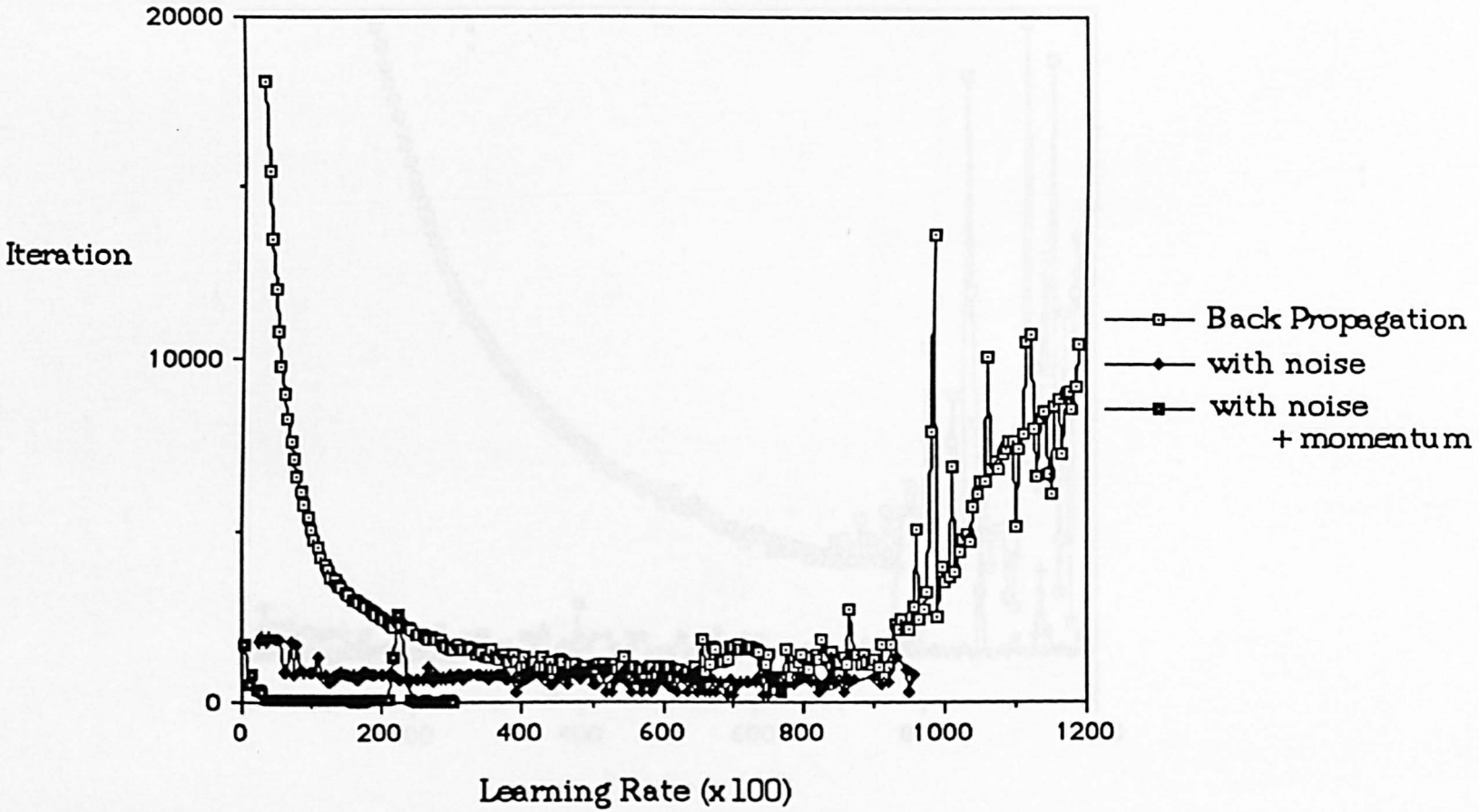

Figure D.5. Back Propagation model with Random noise and Momentum added, using a single initial random seed setting (43).

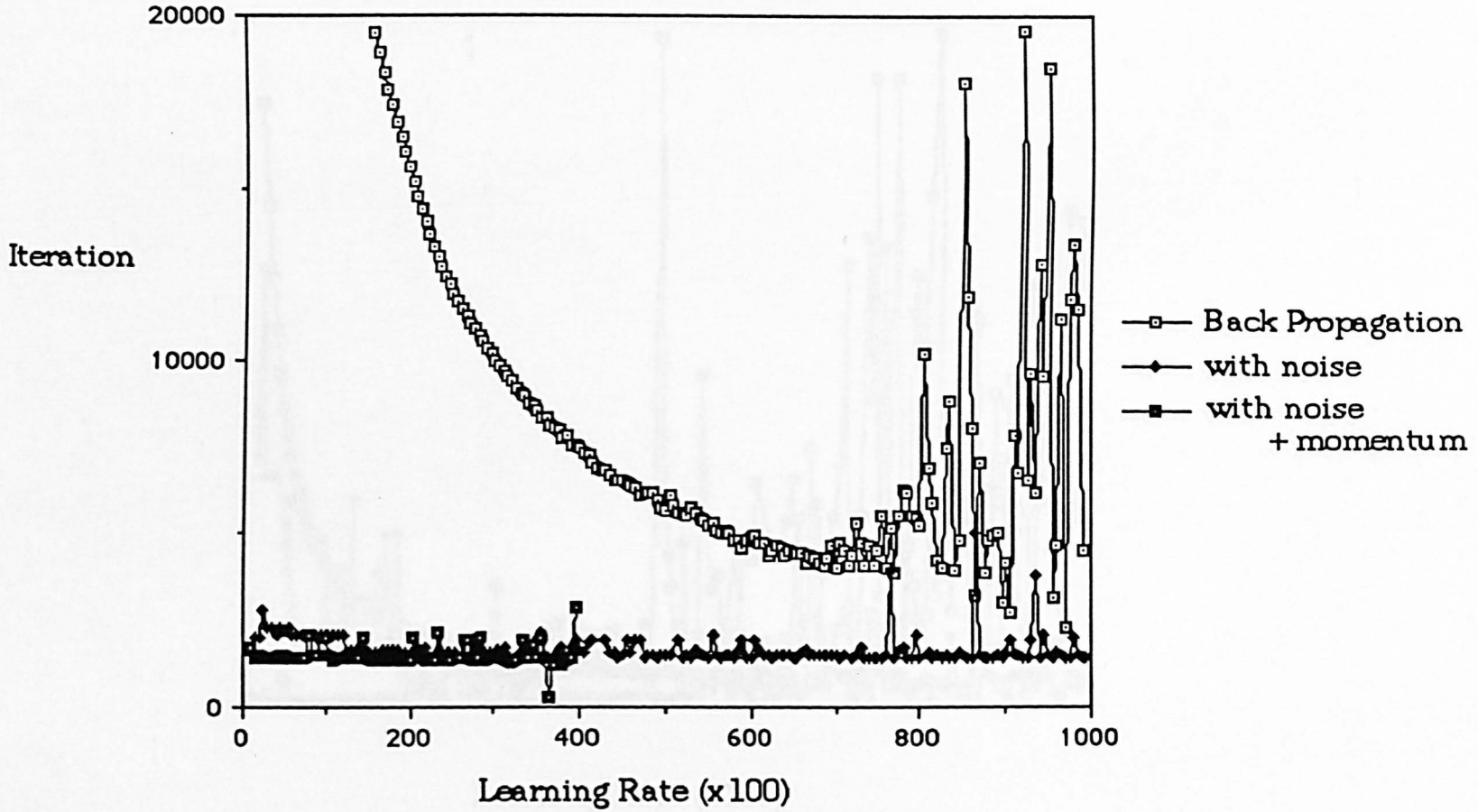

Figure D.6. Back Propagation model with Random noise and Momentum added, using a single initial random seed setting (89).

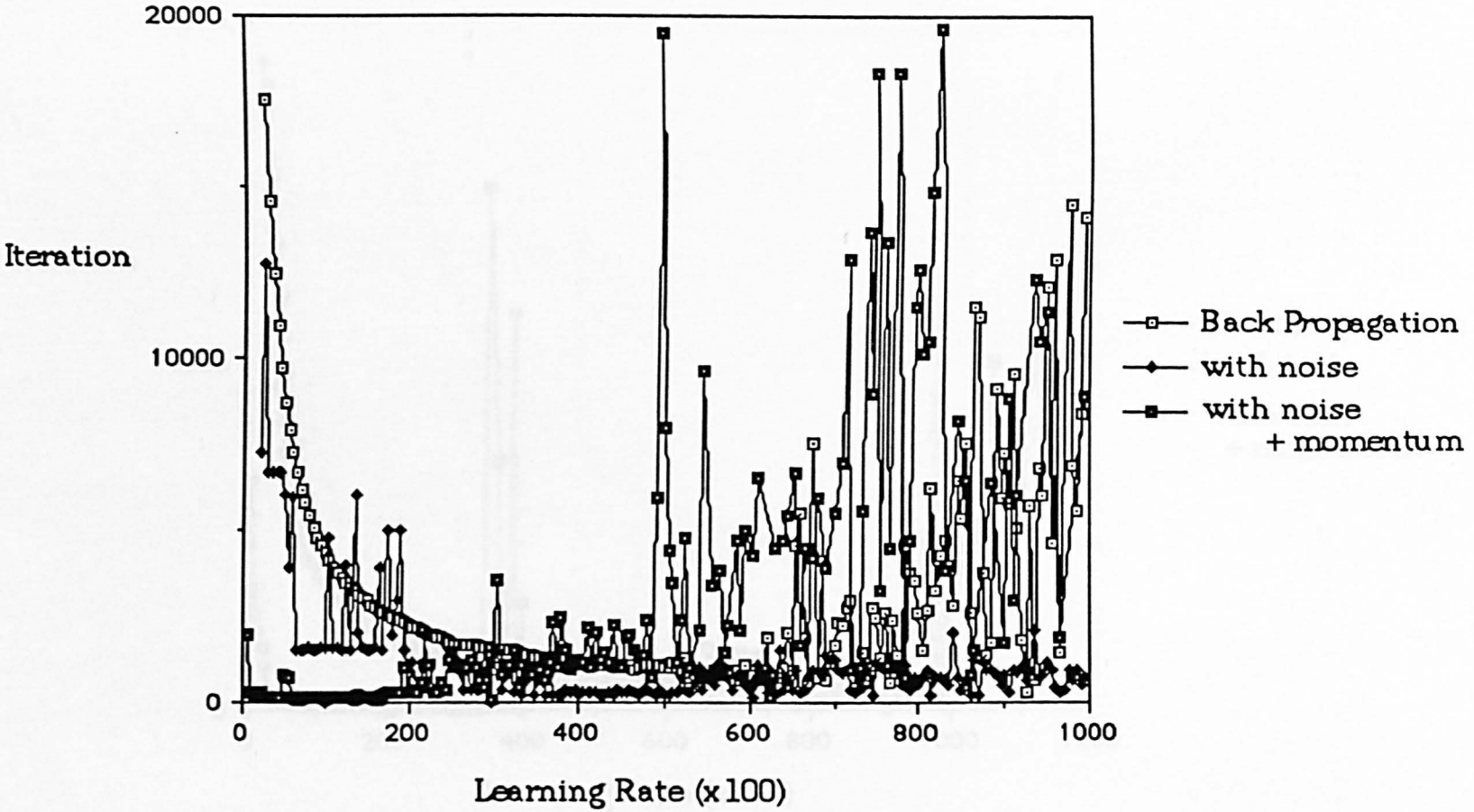

Figure D.7. Back Propagation model with Random noise and Momentum added, using a single initial random seed setting (67).

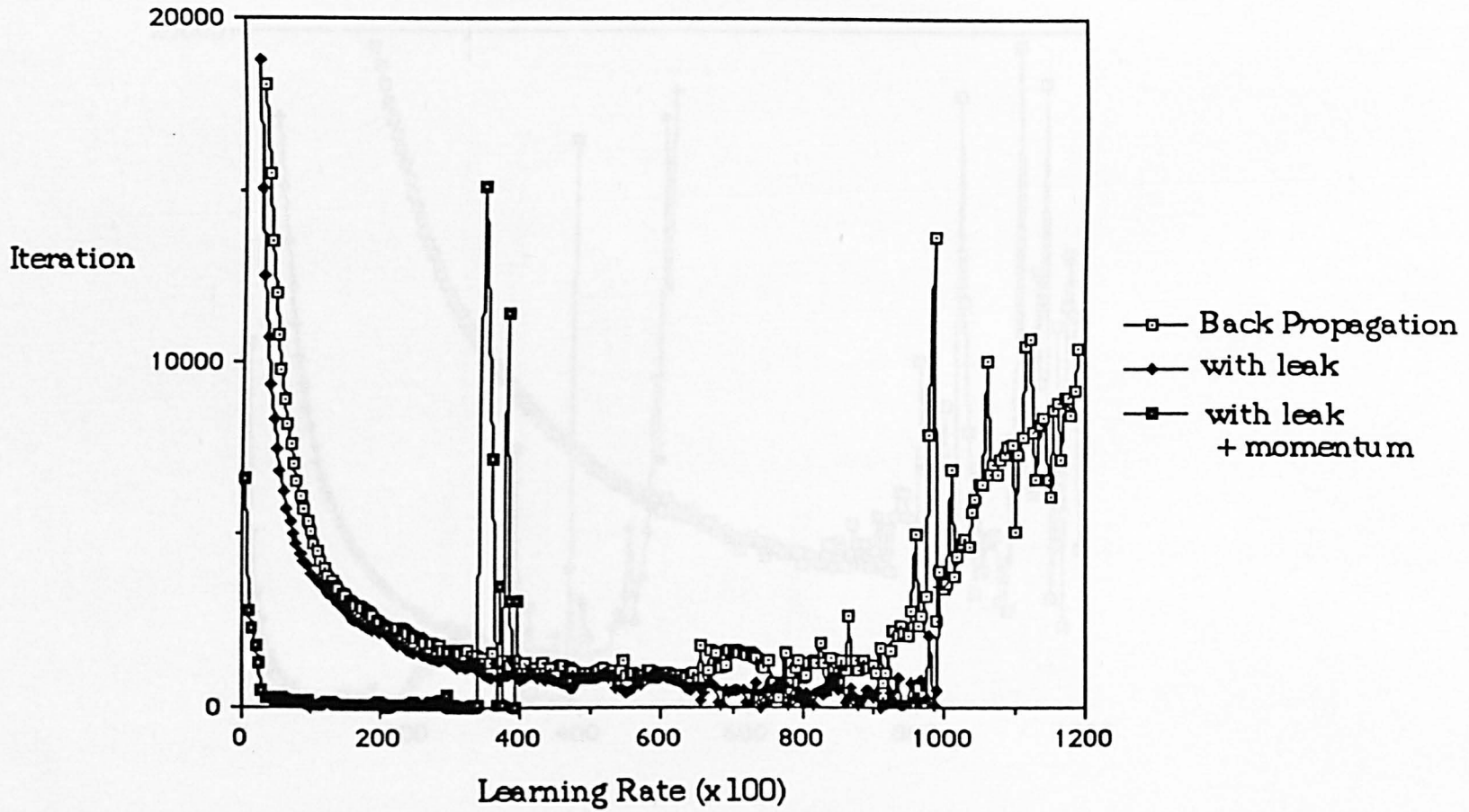

Figure D.8. Back Propagation model with 'Leak' activation and Momentum added, using a single initial random seed setting (43).

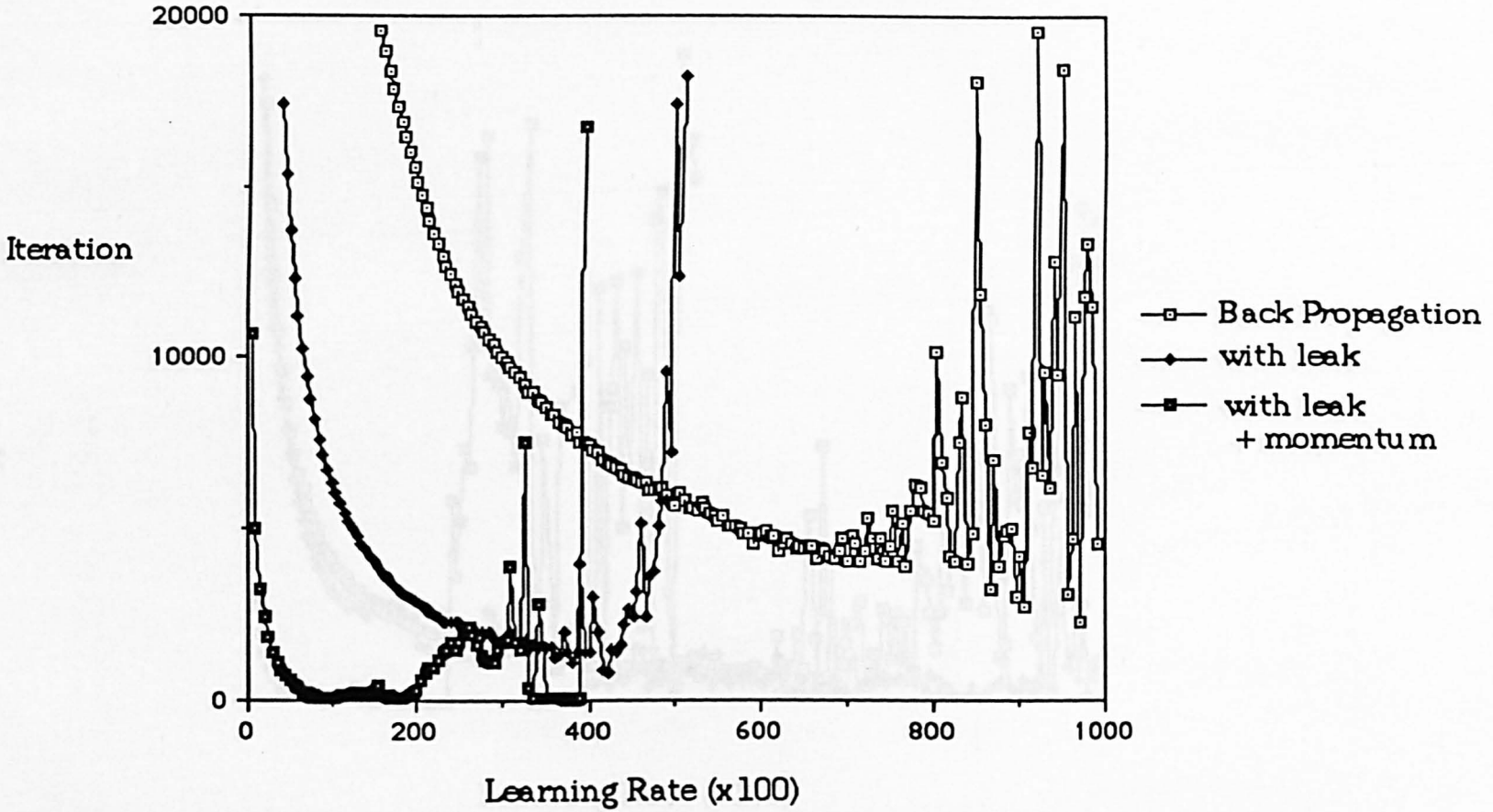

Figure D.9. Back Propagation model with 'Leak' activation and Momentum added, using a single initial random seed setting (89).

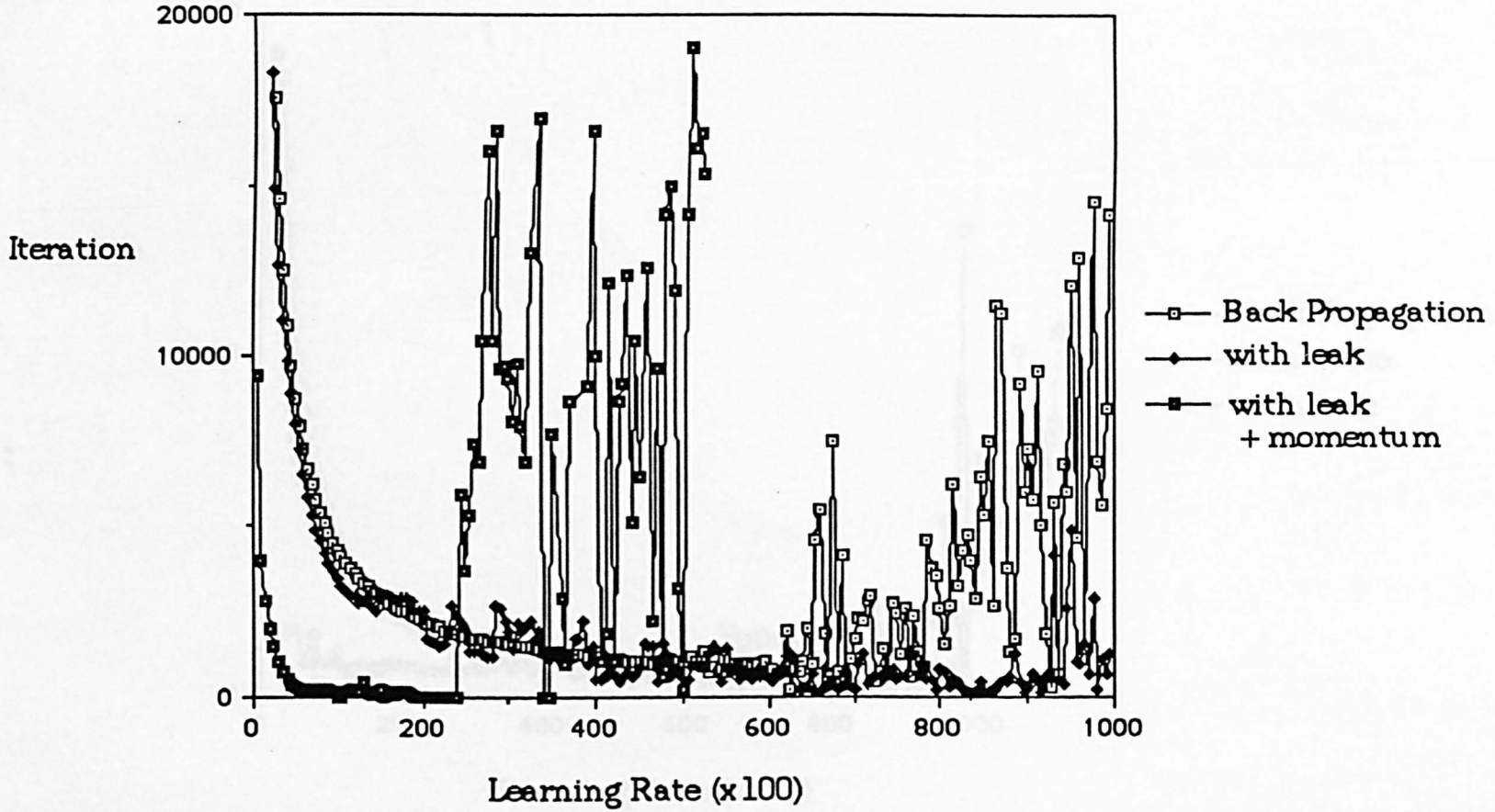

Figure D.10. Back Propagation model with 'Leak' activation and Momentum added, using a single initial random seed setting (67).

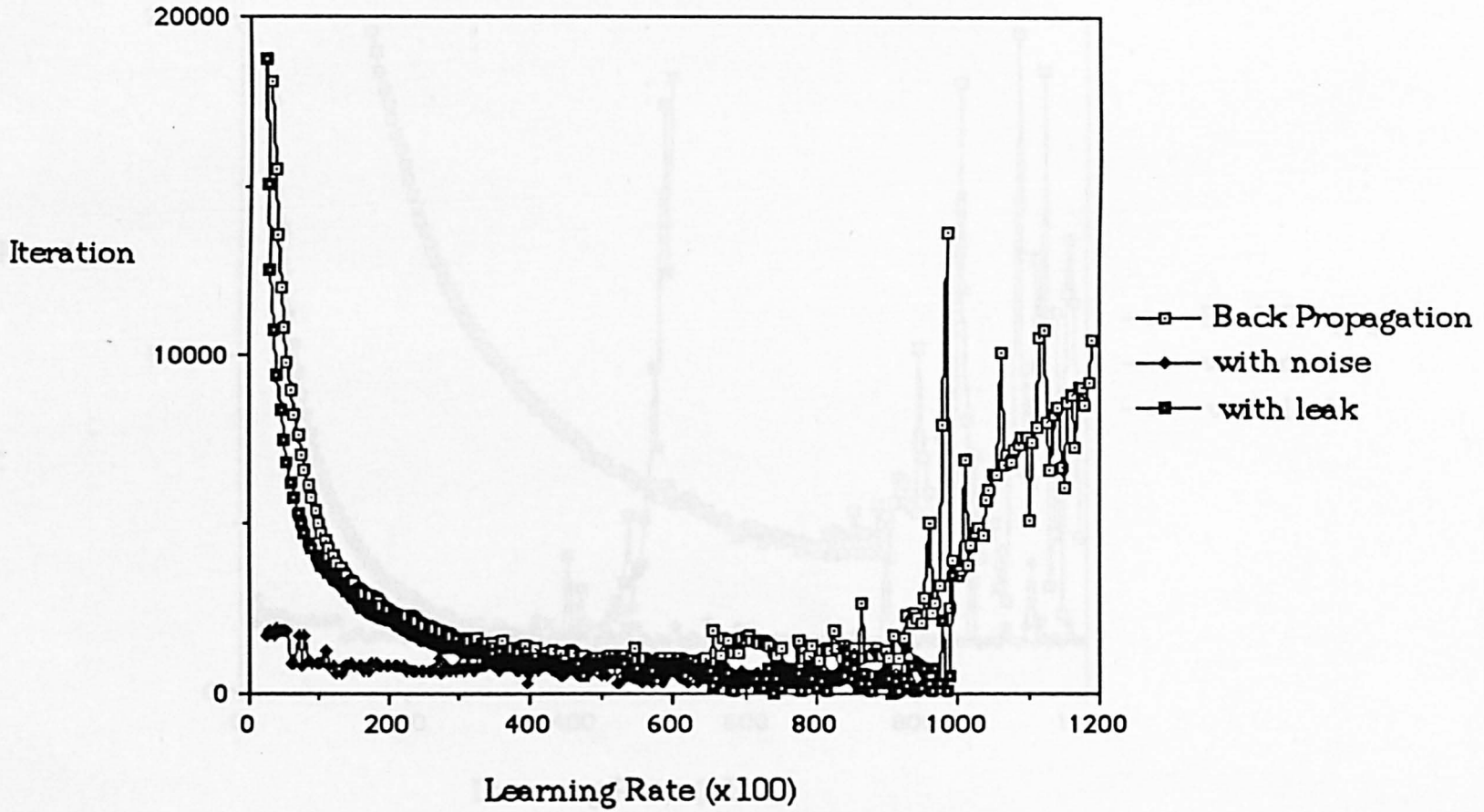

Figure D.11. Back Propagation model comparing random noise and 'Leak' activation, using a single initial random seed setting (43).

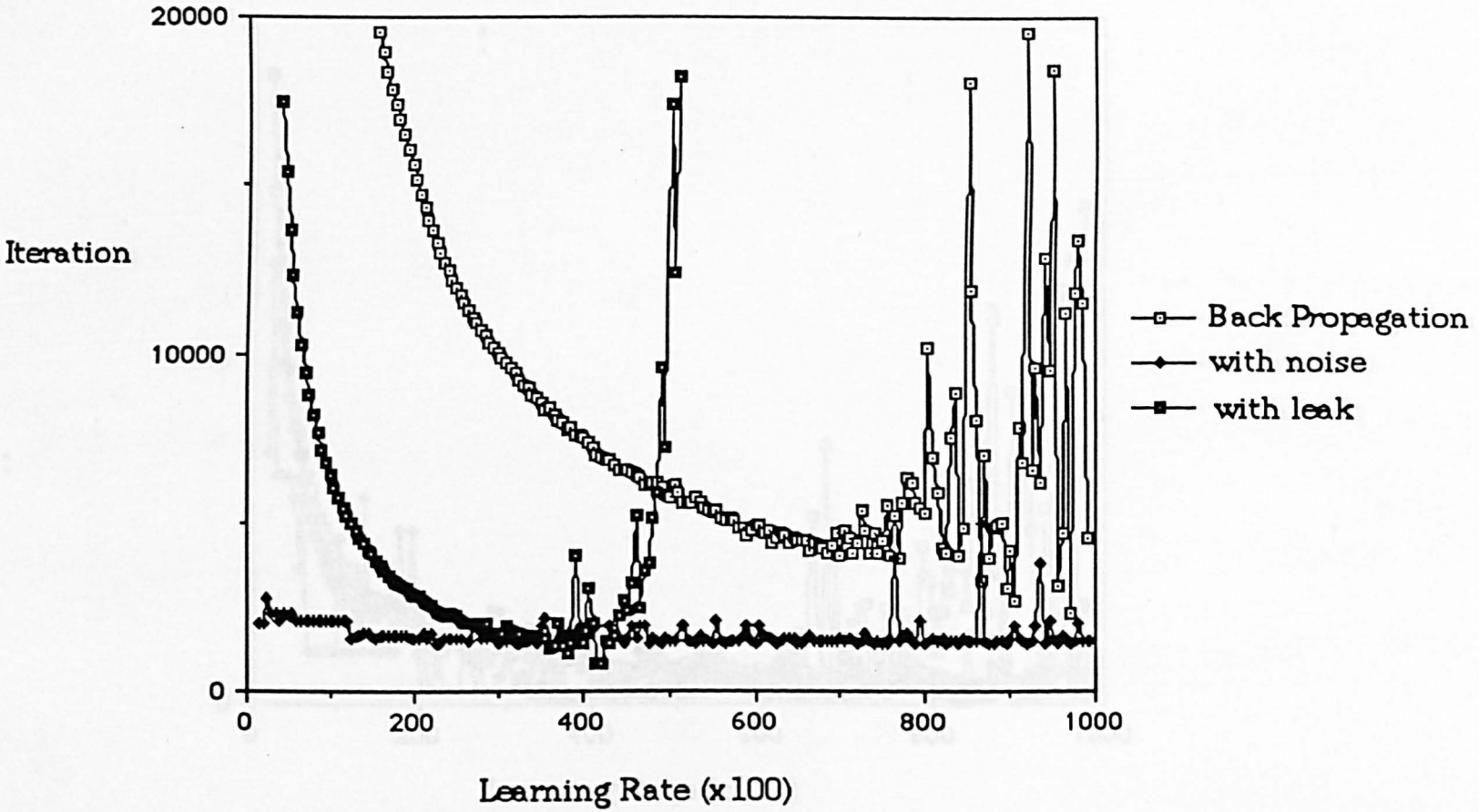

Figure D.12. Back Propagation model comparing random noise and 'Leak' activation, using a single initial random seed setting (89).

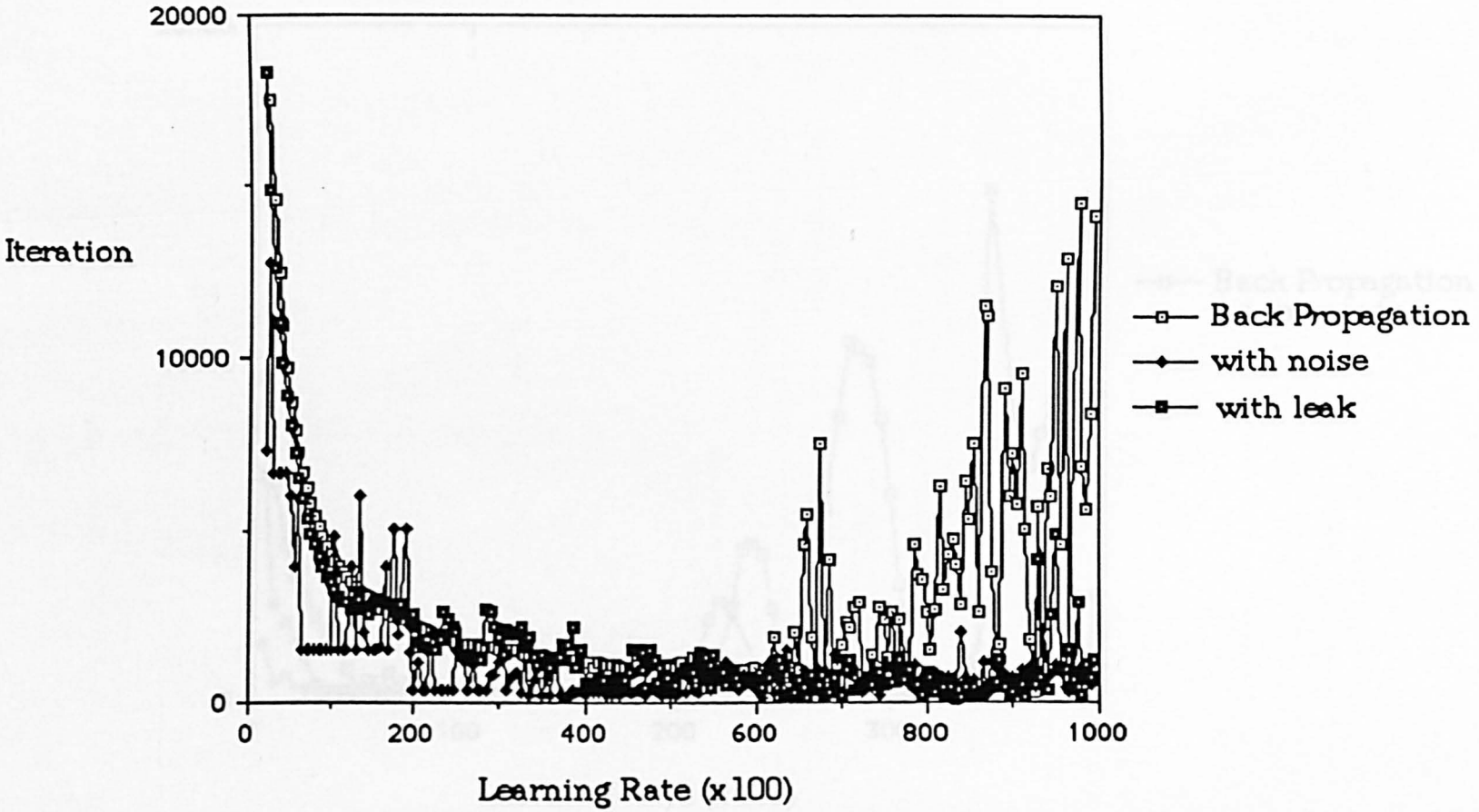

Figure D.13. Back Propagation model comparing random noise and 'Leak' activation, using a single initial random seed setting (67).

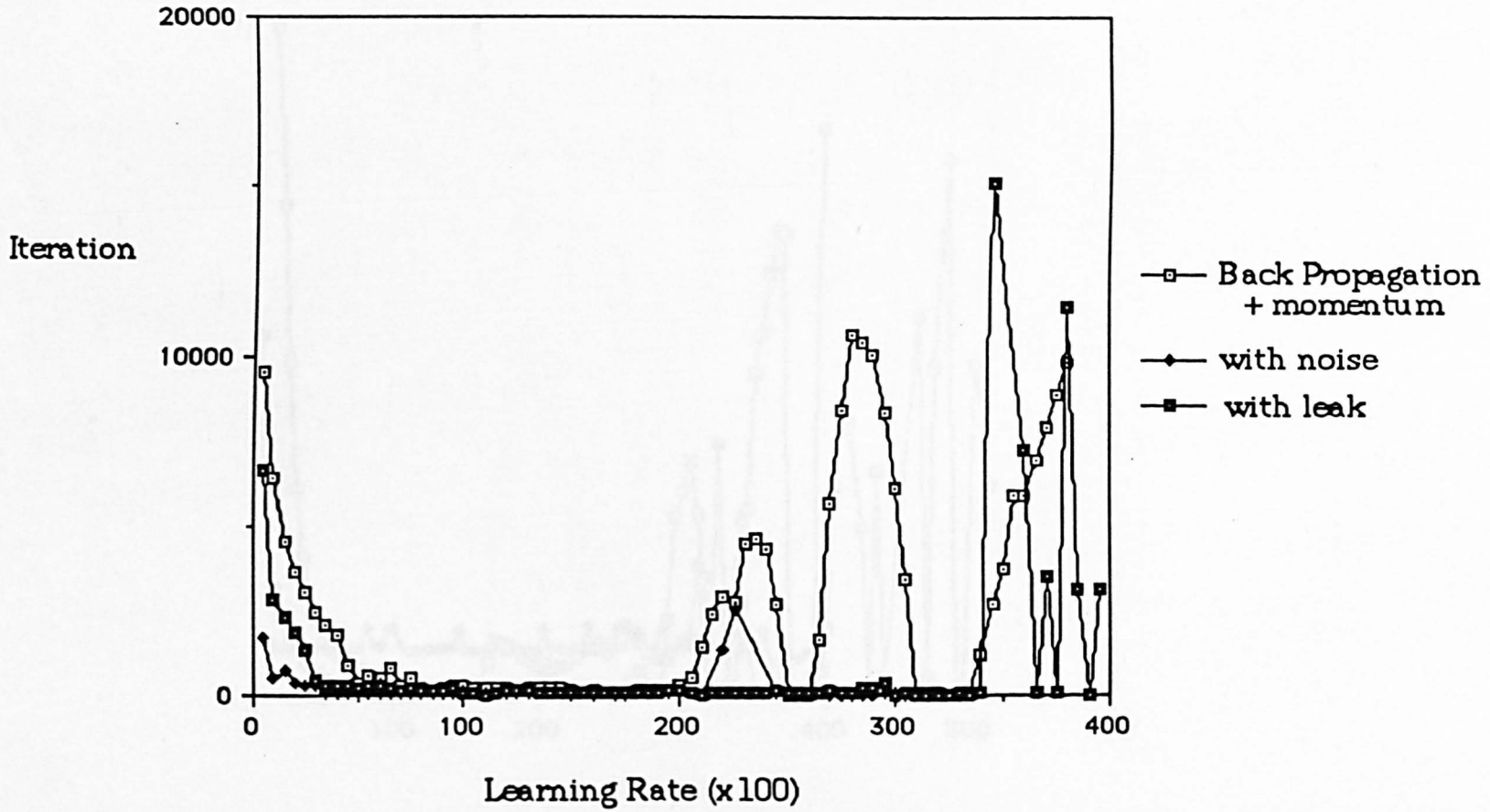

Figure D.14. Back Propagation model with momentum, comparing random noise and 'Leak' activation with added momentum, using a single initial random seed setting (43).

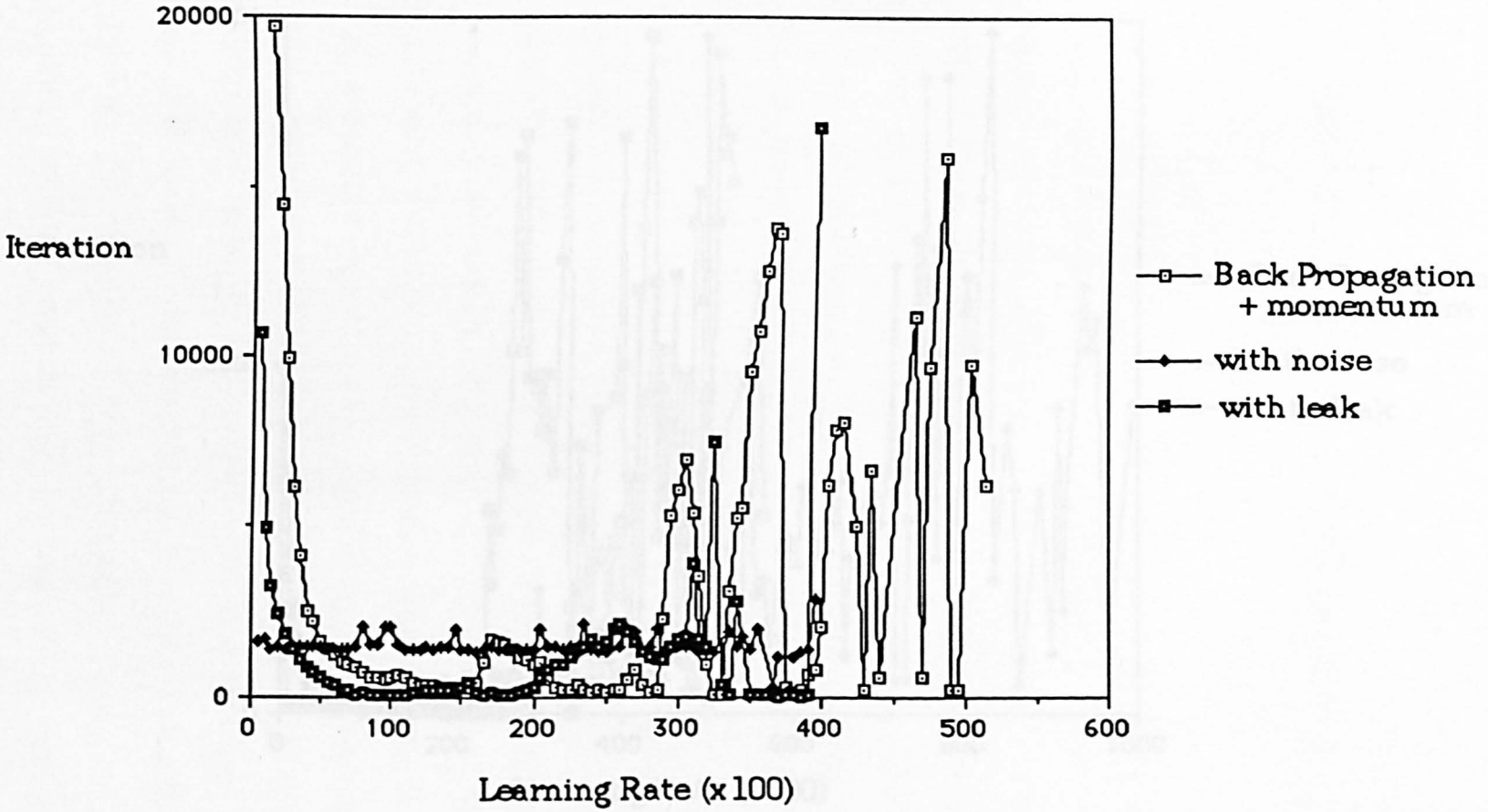

Figure D.15. Back Propagation model with momentum, comparing random noise and 'Leak' activation with added momentum, using a single initial random seed setting (89).

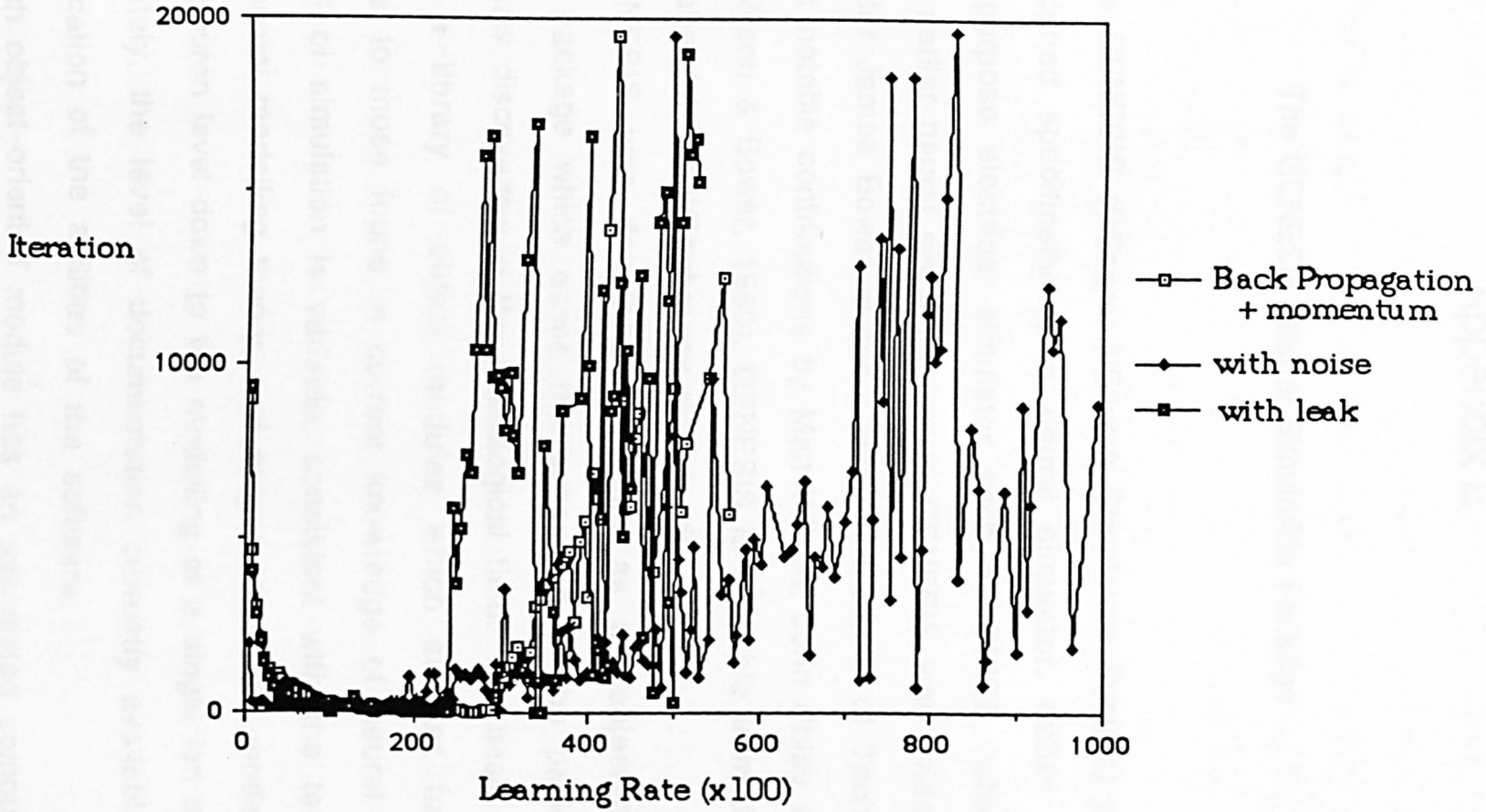

Figure D.16. Back Propagation model with momentum, comparing random noise and 'Leak' activation with added momentum, using a single initial random seed setting (67).

Appendix E

The GENESIS Neural Simulation Package

The GENESIS (GEneral NEtwork SImulation System) package was designed specifically as a neural simulator, rather than a general purpose electrical simulator such as SPICE, which was used for earlier neural simulation work. GENESIS was written by a team under James Bower at the California Institute of Technology, with most notable contributions by Matt Wilson, John Uhley and Upi Bhalla {Wilson & Bower, 1989}. GENESIS is available from the user group, Babel (at babel@babel.cns.caltech.edu)

GENESIS was designed and built as an object–oriented software package which eases the expansion of the package to include new discoveries in the neurological field. The basic system includes a library of object modules which support functions analogous to those found in current knowledge of neural reality. The level of simulation is variable, consistent with the tenets of compartmental modelling theory, and may extend from modelling at a single neuron level down to the modelling of a single ion channel. Unfortunately, the level of documentation currently available, is a poor indication of the abilities of the software.

Each object–oriented module has an associated computational function, which is activated by a message passing system. The function operates on the provided parameters and sets its own parameters based on the results of the calculation. The module may

be queried at any time, by any other element, to determine the values of its associated parameters. The five most important modules are the 'Compartment', 'ChannelC2', 'hh_Channel', 'Spike', and 'Axon' modules. The former four are referred to as elements, and the final module is a connection module.

Compartment

A compartment is the basic unit capable of charge capacity in the model, which simulates a section of passive membrane. The Compartment element includes Membrane resistance and capacitance with a resting equilibrium capacitance, and a potential Axial resistance component for linkage to other compartments. A direct path is allowed in the calculation mechanisms for the introduction of channel conductances, discussed in the next section. Finally, a current injection component allows constant or periodic current injection for network input and testing purposes. (see figure D1)

The Compartment element calculates the voltage across the simulated membrane using an integrative function of the form:

dVm/dt = \f(\f((Em - Vm),Rm) + Σ\f((Vm' - Vm),Ra') + \f((Vm'' - Vm),Ra) + Σ((Ea - Vm)*Ga + inject),Cm)

(Equation D1)

(Where Em is the Equilibrium constant of the membrane, Vm is the voltage across the membrane, Rm is the membrane resistance, Ra is the axial resistance, Ea is the equilibrium

constant of active channels, Ga is the conductance value of active channels, 'inject' is the current injected into the compartment and Cm is the membrane capacitance; time–steps are usually in msecs)

Different integrative methods can be used, chosen from Forward Euler, Exponential, Gear 2nd order, Adams-Bashforth (2 or 3 step) and Trapezoidal.

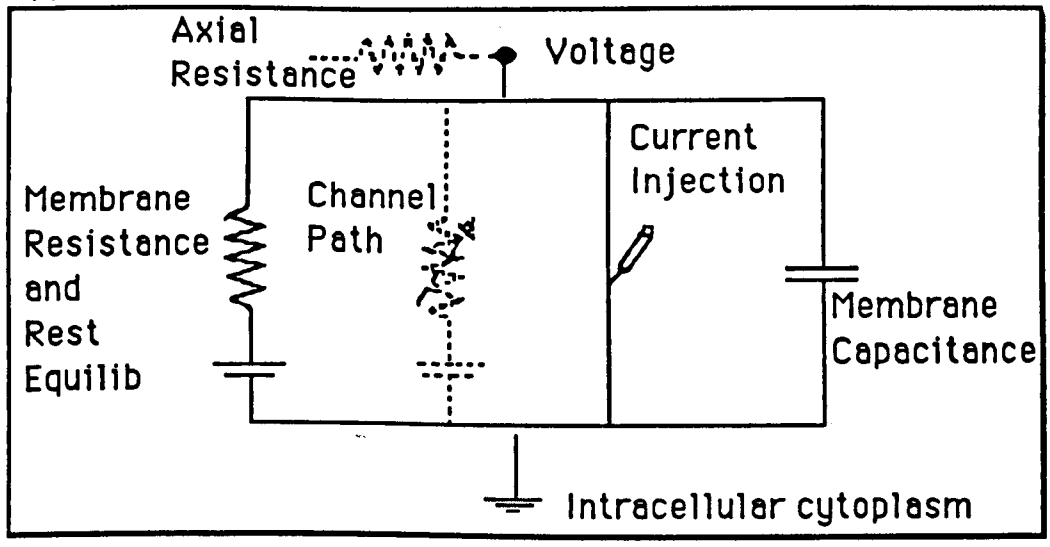

Figure D1. Compartment module of the GENESIS package.

ChannelC2

The channel element simulates a time dependent ionic channel, activated by a synaptic element with a particular weight. The channel element calculates the conductance of a particular channel based on the incoming activation, maximum possible activation and the supplied activity time constants. The channel conductance is calculated:

$$\text{activation} = d^2y/dt^2 + \frac{(tau1 + tau2)}{(tau1*tau2)} * dy/dt + \frac{1}{(tau1*tau2)} * y$$

$$Gk = gmax*A*y$$

(Equation D2)

(where Gk is the channel conductance, gmax is the maximum possible conductance and A is a normalization factor. Tau1 and tau2 are time constant factors which allows 'activation' to impose a damped second order time course on the conductance)

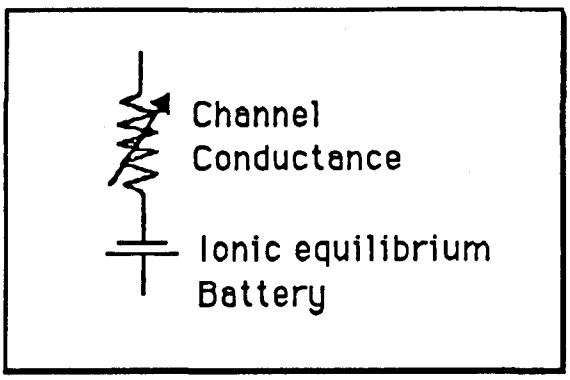

Figure D2. Ionic channel element of the GENESIS package.

hh_Channel

The hh_Channel element is analogous to the Hodgkin–Huxley, or voltage dependent, ion channels found in a cell membrane. These channels are operated by the voltage along the membrane of a compartment. These are characterised, in the model, by four sets of time constants which control the operation of the voltage gated channels at different voltage differential levels. The actions of these channels feed directly into the compartment in which they reside.

Spike

A spike event is generated by the spike element. This element produces a spike of pre–defined amplitude if the input voltage from the connected compartment is above a pre–defined threshold, as long as a pre–defined 'refractory' period has elapsed since the last spike. The spike element is usually coupled to an axonal element, for which it includes a timed buffer to store the spike event. When the timer on a particular buffer has elapsed the spike is delivered to the axon target.

Axon

An axon connection element interacts with the spike element to produce a delay between the onset of a spike in the somatic end of the axon and the delivery of the spike to the target cell process. During delivery of the spike, the activation provided to the target is scaled by an axonic weight, analogous of a connection weight in an artificial neural network.

The MultiCell Example

The MultiCell example is supplied with the GENESIS software as a set of five script files. Each is arranged to correspond to the building of a particular set of modules. The five script files are:

MultiCell.g

constants.g

channel.g

compartment.g

cells.g

The MultiCell.g file is a startup file which "includes" the other files, as part of the execution cycle. The process of building the simulation is split amongst the five files, beginning with the definition of constants and functions to be used in later parts of the script.

The constants are read in first from the constants.g script file. These include the equilibrium constants for membrane resting and leakage states, and for active Sodium and Potassium channels. Peak conductances for the active channels are also defined here.

The channel creation functions are located in the channel.g file, which is the next to be processed. One function is defined for the creation of active Sodium and Potassium channels. A further two functions are defined to create Hodgkin–Huxley Sodium and Potassium channels.

The general syntax for the creation of an active channel is:

```
create    ChannelC2 {compartment/channel}
```

where {compartment/channel} are supplied parameters used to localise the channel in a compartment. The channel parameters must then be set. These include the channel Equilibrium constant, the channel time constants and the peak conductance associated

with the channel. The syntax for the setting of these parameters is as follows:

```
set  ^\          // set parameters of the above element
     Ek          {Ek}       \// Equilibrium constant
     tau1        {tau1}     \// Time constant 1
     tau2        {tau2}     \// Time constant 2
     gmax        {gmax}     // Peak conductance value
```

Hodgkin–Huxley channels require a more detailed set of parameters, including the active area of the channels, in order to calculate the mean conductance, and several rate constants used to control the output of the channels at particular voltage values. Hodgkin–Huxley channels will not be altered in the modified model so they will not be described more fully here.

The compartment.g script file contains a function for creating a compartment within the GENESIS system. Parameters required for this function include the length and diameter of the compartment, from which the area and cross-sectional areas are calculated. These are then used with the resistive and capacitive constants to set membrane resistance, capacitance and axial resistance. Other required parameters are the resting equilibrium constant, and unitary measures of resistance and capacitance. A compartment is created in the following manner:

```
create    compartment    {path}    // location of
object
set       {path}    \
          Em        {Erest}    \// Rest Equilibrium
```

```
Rm          {rm/area} \// Membrane resistance
Cm          {cm*area} \// Membrane capacitance
Ra          {ra*1/xarea}// Axial resistance
```

The Cell.g script file contains a function which creates an entire cell. This function calls the compartment and channel creation functions defined previously. The spike element for cell output is also created in this script; this is created and set in the following manner:

```
create      spike     spike
set   spike       \
      thresh          0     \// Spike threshold
      abs_refract     10    \// Refractory period
      output_amp      1     // Output amplitude
```

Also in the cell.g script, the axon is created, and the message passing system is defined to link the soma with the spike element and the spike element with the axon. The final parameters define what type of message to send and the source of the value to send with the message:

```
create      axon        axon

sendmsg     soma              spike       INPUT Vm

sendmsg     spike       axon        BUFFER name
```

Finally, the remainder of the MultiCell.g script is executed which calls the creation routines for the two cells, positions them in three dimensional space and connects the axons to specific targets. The axon from cell 1 is connected to the Sodium channels of cell 2, and the axon parameters are set:

```
connect    /cell1/axon     /cell2/soma/dend/Na_channel \
            with synapse   // Use synapse function to
                              modify axonic spikes
set        /cell1/axon:0 \
     delay      5     \// 5 msecs transmission delay
     weight     30    // synapse weight

connect    /cell2/axon     /cell1/soma/dend/K_channel \
            with synapse   // Use synapse function to
                              modify axonic spikes
set        /cell1/axon:0 \
     delay      5     \// 5 msecs transmission delay
     weight     300   // synapse weight
```

As a source of excitation a current injection is supplied to cell 1, which should trigger the cell into spiking behaviour after a summation delay:

```
set  /cell1/soma     inject     3e-4 // microAmps
```

The Experimental modifications

The experiment requires few modifications, since there are only two cells to modify. As was explained in chapter six, the existence of only two cells negates the requirements for run–time calculations of point–to–point peptidergic concentrations in three dimensional space, because the only relevant measure becomes the distance between the two cells. In this case, we can simulate a basic peptidergic system by adding an extra axon from each cell and using the delay, time constant and weighting parameters to simulate peptidergic activity.

The modifications require only the definition of two new axons and their parameters, with added connections between the cells and a new message for each axon to link the spike element to the axon.

The new axons are defined and set as:

```
create      axon        /cell1/axon2
create      axon        /cell2/axon2

connect     /cell1/axon2     /cell2/soma/dend/Na_channel2 \
                  with synapse   // Use synapse function to
                                    modify axonic spikes
      set         /cell1/axon2:0 \
            delay       30    \// 5 msecs transmission delay
            weight      3     // synapse weight

connect     /cell2/axon2     /cell1/soma/dend/Na_channel2 \
```

```
            with synapse   // Use synapse function to
                                modify axonic spikes
set         /cell2/axon2:0 \
      delay      30    \// 5 msecs transmission delay
      weight     3     // synapse weight
```

```
sendmsg   spike      /cell1/axon2          BUFFER name
sendmsg   spike      /cell2/axon2          BUFFER name
```

The weight parameters are set to 3 in order to emulate a weak connection. The delay is set to six times the 'normal' transmission rate in order to simulate a diffusional process. This is aided by setting the time constants of the receiving active channels to a value of 20 milliseconds, which is done at run–time in a channel control window.