The synthesis of Artificial Neural Networks using Single String Evolutionary Techniques

A thesis submitted to
The Robert Gordon University
in partial fulfilment of the requirements for
the degree of Doctor of Philosophy

Christopher MacLeod

School of Electronic and Electrical Engineering
The Robert Gordon University
Aberdeen, Scotland, Spring 1999

Declaration

I hereby declare that this thesis is a record of work undertaken by myself. That it has not been the subject of any previous application for a degree and that all sources of information have been duly acknowledged.

Christopher MacLeod.

1999

Acknowledgements

Firstly, I am indebted to my supervisor Mr Grant M Maxwell for his help, support and encouragement throughout the project. Without his supervisory skills the project would have been much more difficult and not nearly as enjoyable. I am also indebted to the collaborating establishment, Eident Ltd and its director Mr John Anderson and my second supervisors Dr A Miller and Dr S Elder.

A special note of thanks is due to the following people, who assisted in proof reading the thesis and made many valuable suggestions during the project: Dr Lawrence Thirkell, Mr Robert Kenchington, Mr Angus MacIver and Mrs Anne Thirkell.

I would also like to thank Richard Dawkins and Adrian Thompson for their permission to include details of their work in the text.

Finally I would like to extend my appreciation to all the members of the Robert Gordon University, School of Electronic and Electrical Engineering who assisted me in any way during the project. In particular, I would like to mention members of the Electronics Group (Mr K Gow, Mr S Chalmers and Prof N Deans) and research co-ordinator Prof W T Thompson.

C MacLeod 8:3:99

Abbreviations used in text

To avoid repetition of certain common terms, the following abbreviations are used in the text.

ANN	-Artificial Neural Network.	*BNN*	-Biological Neural Network.
CNS	-Central Nervous System.	*PNS*	-Periphery Nervous System.
BP	-Back Propagation.	*BT*	-Boltzmann training algorithm.
ART	-Adaptive resonance theory.	*XOR*	-Exclusive OR.
GA	-Genetic algorithms.	*AI*	-Artificial Intelligence.
CT	-Cauchy Training.	*BAM*	-Bi-directional Associative Memory.
ASIC	-Application Specific Integrated Circuit.	*FPGA*	-Field Programmable Gate Array.
EANN	-Evolutionary ANN.	*EA*	-Embryological Algorithm.
BM	-Boltzmann Machine	*GAL*	-Grow and learn.
FF	-Feedforward.	*FB*	-Feedback.
DNA	-Deoxyribonucleic acid.	*RNA*	-Ribonucleic acid.
DSP	-Digital Signal Processing.	*FIR*	-Finite Impulse Response.
IIR	-Infinite Impulse Response.	*ES*	-Evolutionary Strategy.
EP	-Evolutionary Programming.		

These abbreviations will also be introduced before use in the text.

ABSTRACT

The research presented in this thesis is concerned with optimising the structure of Artificial Neural Networks. These techniques are based on computer modelling of biological evolution or foetal development. They are known as Evolutionary, Genetic or Embryological methods. Specifically, Embryological techniques are used to grow Artificial Neural Network topologies. The Embryological Algorithm is an alternative to the popular Genetic Algorithm, which is widely used to achieve similar results.

The algorithm grows in the sense that the network structure is added to incrementally and thus changes from a simple form to a more complex form. This is unlike the Genetic Algorithm, which causes the structure of the network to evolve in an unstructured or random way.

The thesis outlines the following original work: The operation of the Embryological Algorithm is described and compared with the Genetic Algorithm. The results of an exhaustive literature search in the subject area are reported. The growth strategies which may be used to evolve Artificial Neural Network structure are listed. These growth strategies are integrated into an algorithm for network growth. Experimental results obtained from using such a system are described and there is a discussion of the applications of the approach. Consideration is given of the advantages and disadvantages of this technique and suggestions are made for future work in the area. A new learning algorithm based on Taguchi methods is also described.

The report concludes that the method of incremental growth is a useful and powerful technique for defining neural network structures and is more efficient than its alternatives. Recommendations are also made with regard to the types of network to which this approach is best suited.

Finally, the report contains a discussion of two important aspects of Genetic or Evolutionary techniques related to the above. These are Modular networks (and their synthesis) and the functionality of the network itself.

Contents

Chapter 3. Artificial Neural Networks

Chapter 4. Biological Evolution and Embryology

Chapter 7. Growth strategies for Artificial Neural Networks

Chapter 8. Results obtained from application of Growth Strategies

Chapter 1
Introduction

1.1 Introduction to chapter

This chapter details with the background, objectives and structure of the thesis. A breakdown, chapter by chapter, of the report and list of abbreviations is included.

The research presented in this thesis is concerned with the use of optimisation techniques to overcome problems with Artificial Neural Networks, in particular the difficulty of defining a good structure for the network (which is usually done through experience or guesswork). These techniques are based on computer modelling of biological evolution or foetal development, and are known as Evolutionary, Genetic or Embryological methods. Specifically, Embryological techniques are used to grow neural network topologies.

The Artificial Neural Network is itself, like genetic methods, based on biology. It is an attempt to create Artificial Intelligence, using technology to model the structure of the brain.

The background to the research is given below in section 1.2 to section 1.5. These sections give a brief explanation of what the artificial neural network is and what its limitations are. The role of the evolutionary algorithm, in overcoming these limitations, is then considered.

1.2 The Artificial Neural Network

Neurologists have found that the animal nervous system, including the brain, is made up of tiny processing units called Neurons (or Neurones). Each neuron is a living biological cell which has become specialised, through evolution, in such a way that it can process electro-chemical signals.

Although there are many millions of neurons in the brain, each one can process only a small amount of information. It is the combination of all these many neurons, operating in parallel, which is thought to result in the attributes of biological intelligence, including consciousness itself.

In the 1940s, engineers began to wonder if they could create artificial intelligence by connecting up simple electronic processing units, based on the biological neuron, into a network. Such a network is called an Artificial Neural Network or ANN [1].

1.3 Limitations of Artificial Neural Networks

Although ANNs have been successful in many fields, they have failed to live up to the promise of a realistic simulation of the biological brain. The main area in which their success lies is in the recognition, reconstruction and classification of spatial patterns [2]. There are many reasons why they have had more limited success than originally hoped. Some of the more important reasons are listed below:

- Although the neurons in the brain are much slower than a computer or electronic circuits, the brain has much higher connectivity [3].

- The neurons of the brain are able to redistribute themselves by growing in different directions; in other words, connections within the brain are plastic [4].

- The biological neuron is not yet fully understood [5].

- No satisfactory method has yet been forthcoming for modelling the activity of the brain [6].

- Artificial neural systems also show enough complex activity to make modelling difficult or impossible [7].

- Design of an ANN is presently a matter of trial and error [8].

Although the first three of these problems are due to lack of understanding or technological limitations (and are therefore difficult to solve) the latter three may be approached using the evolutionary method outlined in this thesis.

1.4 The use of Evolutionary Artificial Neural Networks

The use of evolutionary techniques in science and engineering is a recent innovation [9]. It stems from the recognition that biological evolution is an efficient search mechanism for finding good solutions to the practical problems involving survival in the natural world.

The complexity and success of the biological brain is due to biological evolution. It is therefore reasonable to attempt to apply these techniques to the artificial equivalent, the ANN.

In the work presented here, the network *grows* by adding incrementally to its structure. It starts off simply and becomes complex enough to solve the problem by expanding its structure. This approach is different from the more usual types of evolutionary algorithms (which are directed random searches) and is termed Embryology (because it is similar to the growth of an embryo) or Incremental Evolution.

It is hoped that, by allowing the ANN to grow under the control of an evolutionary algorithm, it will develop the ability to solve difficult problems without the need for the human controller to understand the complex system thereby developed.

1.5 Unique aspects of the approach presented here

Although researchers have used genetic and evolutionary techniques with neural networks before, there are several unique aspects to the approach presented here. The most important of these are listed overleaf.

The use of single string technique for growing the ANN.

1. The use of an algorithm which allows the network to *grow* incrementally (rather than a directed random search).
2. A comprehensive system for the growth of the network from simple to complex form, capable of automatically finding the simplest implementation.
3. The classification of growth strategies for use in network growth, their effects and therefore the implementation of efficient search strategies.
4. The consideration of the hierarchical aspects of network organisation.

1.6 Objectives

This thesis is concerned with the application of Single String Evolutionary or Embryological Methods to the growth of artificial neural network topology, and also the consideration of important related aspects such as the applications of such networks.

The thesis does not consider the application of evolutionary methods to other aspects of neural networks such as network learning and neural functionality (it only considers the McCulloch Pitts Neural model), except where these aspects are directly related to topology growth (although these aspects are considered in the sections on further work).

The stated objectives of the project were:

- Study of relevant literature.
- Design and implementation of evolutionary algorithm.
- Combination of evolutionary algorithm and neural network.
- Comparison of generated network with standard network.
- Study of learning algorithms related to this approach.
- Application to non-specific problems
- Investigation of the applications of such networks.

1.7 Chapter overview

The thesis may be thought of as consisting of three sections:

- Chapters 1, 2, 3, 4 and 5 (first half) are introductory and contain background detail.
- Chapters 5 (second half), 6, 7, 8 and 10 present research, methods and experimental results.
- Chapters 9, 11, 12, 13 discuss and develop the results and related topics.

It is necessary to include extensive background and introductory chapters since the research material is interdisciplinary, containing ideas from neurology, embryology, biology, evolutionary and genetic techniques and neural networks.

Given below is an overview of each chapter.

Chapter 2: The Biological Neural Network

This is a background chapter, which contains an introduction to biological neural networks. Detail is given on the structure and operation of biological neurons and the large scale organisation and function of the biological brain.

Chapter 3: Artificial Neural Networks

This chapter deals with the history of ANNs, the theory behind the simple neuron and perceptron, as well as the basic feed-forward and recurrent topologies and learning algorithms. There are also sections on future developments and the practical implementation of ANNs.

Chapter 4: Biological Evolution and Embryology

The philosophy behind the artificial computer models is outlined in this chapter. The history, formation and theory of natural Evolution and Embryology are outlined and compared. Sections outlining modern perspectives are also included.

Chapter 5: Artificial Evolutionary methods and their application to Neural Networks

This chapter outlines both the Genetic Algorithm and the Artificial Embryology, compares each approach and outlines their application to the Artificial Neural Network. The section on the Artificial Embryology contains the main original idea behind the project.

Chapter 6: Literature search

During the project a comprehensive literature search was undertaken. This chapter explains the sources used in the search, the methods for finding relevant publications and a review of the important work found using these methods.

Chapter 7: Growth strategies of artificial neural networks

The basic experimental results are contained in this chapter and the next (chapter 8). An outline of the methods used to obtain the results is given and then each strategy is individually considered.

Chapter 8: Results obtained from the application of growth strategies

The growth strategies outlined in Chapter 7 are integrated into a single framework, using the ideas outlined in the latter part of Chapter 5. The resulting method is applied to a variety of problems representing all the common neural network applications. The chapter forms the focal point of the thesis.

Chapter 9: Learning in Evolutionary Artificial Neural Networks

Learning and how it relates to evolutionary algorithms is discussed here. The chapter discusses traditional learning algorithms, such as Back Propagation, then goes on to consider algorithms which are especially useful for evolutionary networks such as Incremental Learning.

Chapter 10: Application to non specific problems

This chapter considers how the techniques discussed earlier may be applied to ‘real world’ problems; that is, problems where the data-set is not artificially constrained. Approaches used by other researchers are reviewed and an approach is suggested based on this work.

Chapter 11: Applications of Genetically Generated Neural Networks

This section includes a discussion of simple problems, such as pattern recognition, followed by more difficult problems such as robot control. The use of these techniques as a route towards ‘real’ artificial intelligence is also discussed.

Chapter 12: Further work

In this chapter suggestions are made for further work. In particular, the need for research into unconstrained problems and neural functionality is discussed.

Chapter 13: Conclusions

The final chapter revisits the objectives outlined in the first chapter and critically assesses the success of the project. A summary of results is presented and a short section summarising the need for further work is included.

1.8 Papers included in appendix 1

Papers and reports produced during the course of the project are included in appendix 1.

These provide the reader with a ‘snap shot’ of the thinking at the time the papers were written. This view sometimes changed as the project progressed and more results were produced.

The papers provide a useful overview and summary of the work for readers not able to read the whole thesis in depth and they range over several aspects of the project. Because of this they are also important supplementary information (as are some of the other appendices).

1.9 A note on the use of the term ‘Embryology’

While reading the thesis, one might question the use of the term ‘Embryology’ or ‘Embryological Algorithm’, which are used to describe the technique used. The author prefers the term ‘*Incremental Evolution’* as this is more descriptive of the method. However, there are two reasons why the other terms are used: Firstly, the technique is loosely based on a method used by Dr Richard Dawkins [10] to illustrate foetal growth and the study of this field of biology is termed embryology. Secondly (and more importantly), only two papers [11][12] had been published on the application of Dawkins Biomorphs to ANNs. These refered to the technique as a ‘Constrained Embryology’, thereby setting a precedent. However the terms will be used interchangeably in the text.

Chapter 2

The Biological Neural Network

2.1 Introduction to chapter

This chapter deals with the basic ideas behind the Biological Neural Network (BNN). It is important to provide this background theory since any discussion of artificial neurons refers back to the biological neural model. Also, a thorough understanding of biological neural network operation is needed before the problems of artificial neural networks can be appreciated.

The chapter discusses three areas. Firstly, the structure of biological neurons. Secondly, the operation of biological neurons, and thirdly, the large scale architecture of the biological brain.

2.2 The biological neural network

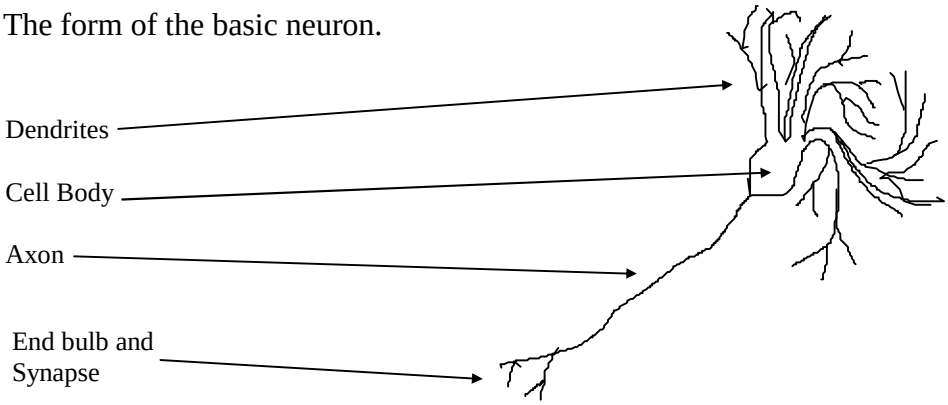
The basic unit of the BNN is the biological neuron or nerve cell. This has the same basic components as all the other cells in the body but is modified to allow the transmission of neural impulses called Action Potentials [1].

2.2.1 The structure of neurons

There are many different types of neuron in the human body [2], but these types all have a similar structure [3]. This structure consists of the cell body which contains the cell nucleus and chemical activity; the axon which transmits the nerve impulse and the dendrites which receive nerve impulses from other neurons. This is shown diagramatically in figure 2.1.

Figure 2.1

The form of the basic neuron.



The neuron receives nerve pulses from the axons of other neurons or from the body's environmental sensing system. These signals are picked up by the dendrites. When the stimulation of the dendrites is strong enough, the neuron is triggered and sends a nerve pulse down the axon. The axon ends in a small bulb which is separated from the next cell by a space called the synapse. The role of the synapse in the transmission of nerve impulses is critical and will be examined in section 2.2.3.

2.2.2 Types of biological neuron

It is possible to look at the different types of neurons in the body from the point of view of functional differences or structural differences. Functional differences are dealt with first.

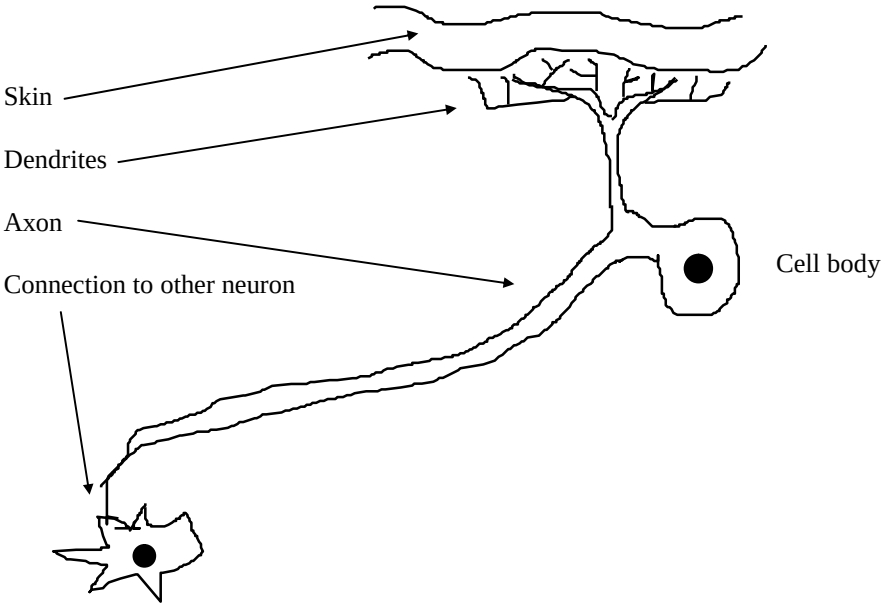
The BNN obtains information from the outside world, processes that information and causes a reaction resulting from that processing. Analysing the BNN in this way leads to the classification of neurons into three categories.

1. Afferent (sensory) neurons.

Afferent neurons collect information from sensory receptors on the skin, receptors on joints and elsewhere and send it towards the central nervous system for processing. This type of neuron is shown in figure 2.2.

Figure 2.2

Afferent neuron.



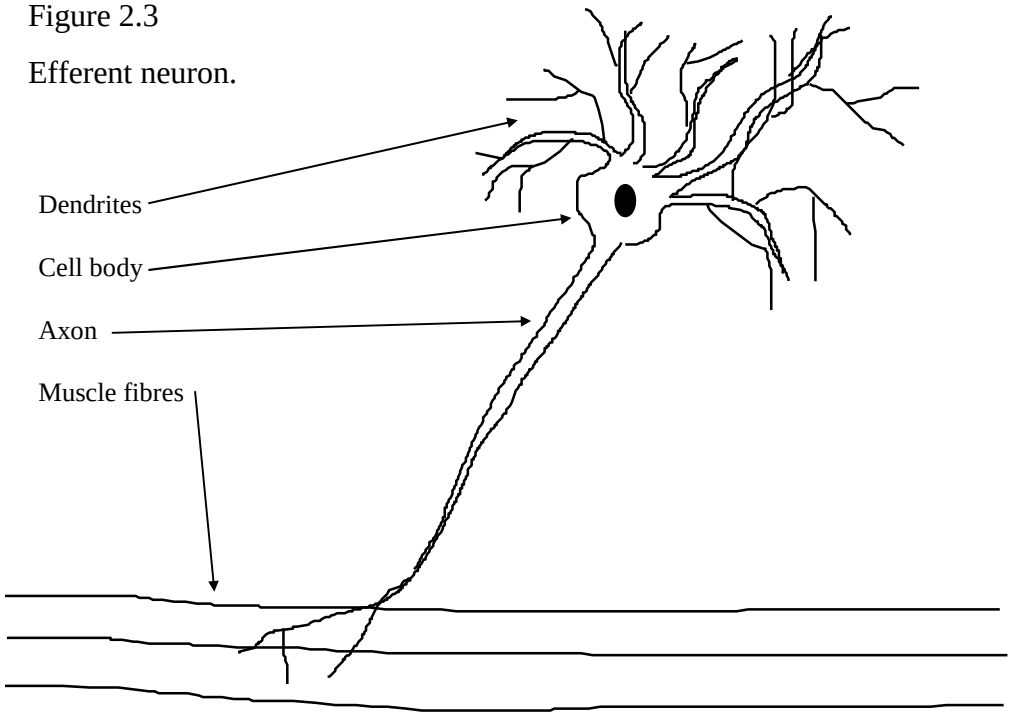
The diagram shows a neuron receiving input from sensory receptors under the skin and passing on this input to the next neuron in line.

2. Efferent (motor) neurons.

Efferent neurons carry information from the nervous system to effectors such as muscles and glands. They are the biological control system and are sometimes called motor neurons for this reason. Figure 2.3 shows a motor neuron.

Figure 2.3

Efferent neuron.



The diagram shows a motor neuron attached to some muscle fibres. When the neuron is stimulated, it will transmit an action potential to the muscle which causes it to contract. In this way the nervous system can influence the outside world.

3. Interneurons.

Most of the nervous system is made up of neurons, called interneurons, which connect only to other neurons. In fact 99% of the nervous system is made up of this type of neuron [4]. When generalised neurons are referred to, it is interneurons which form the model.

As stated earlier, neurons may also be categorised according to their structure. There are many structural types of neuron within the body but, generally, these fall into three categories. They are illustrated in figures 2.4 a, b and c.

Figure 2.4 a)

A multipolar neuron.

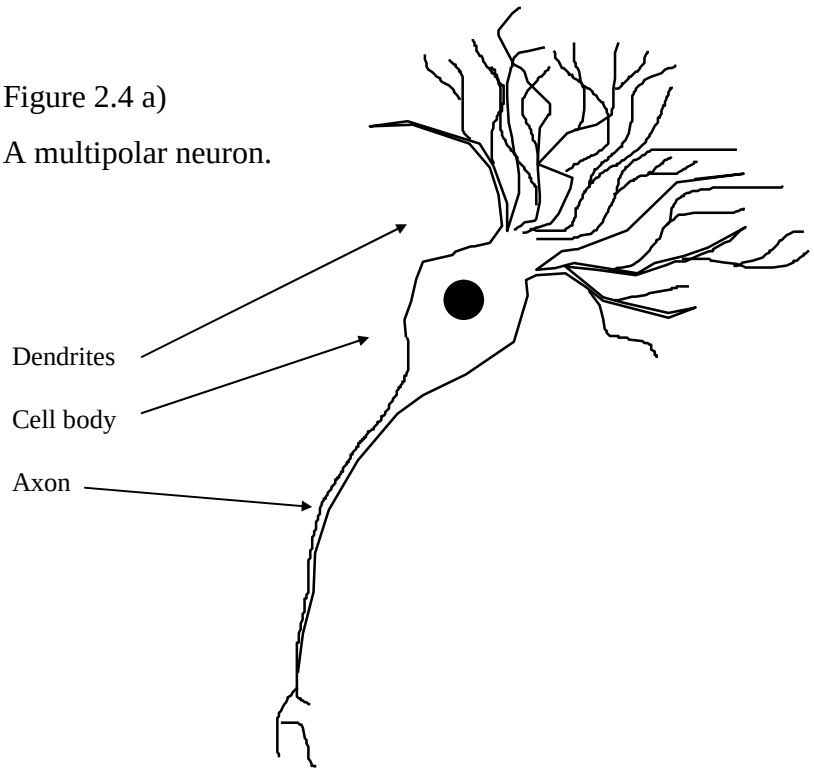


Figure 2.4 b)

A Bipolar neuron.

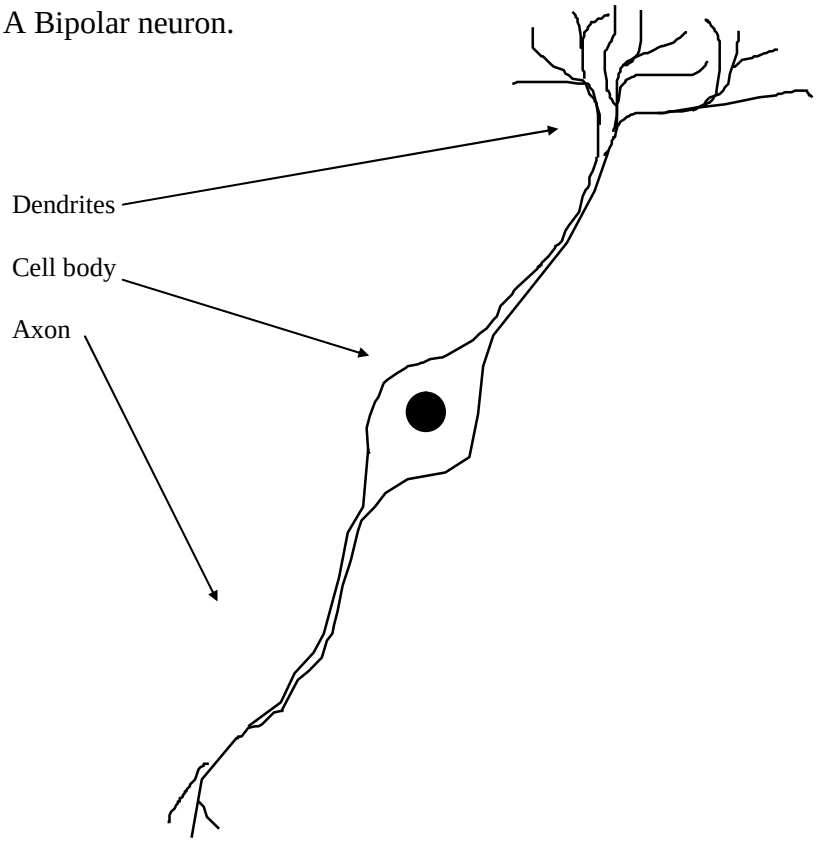
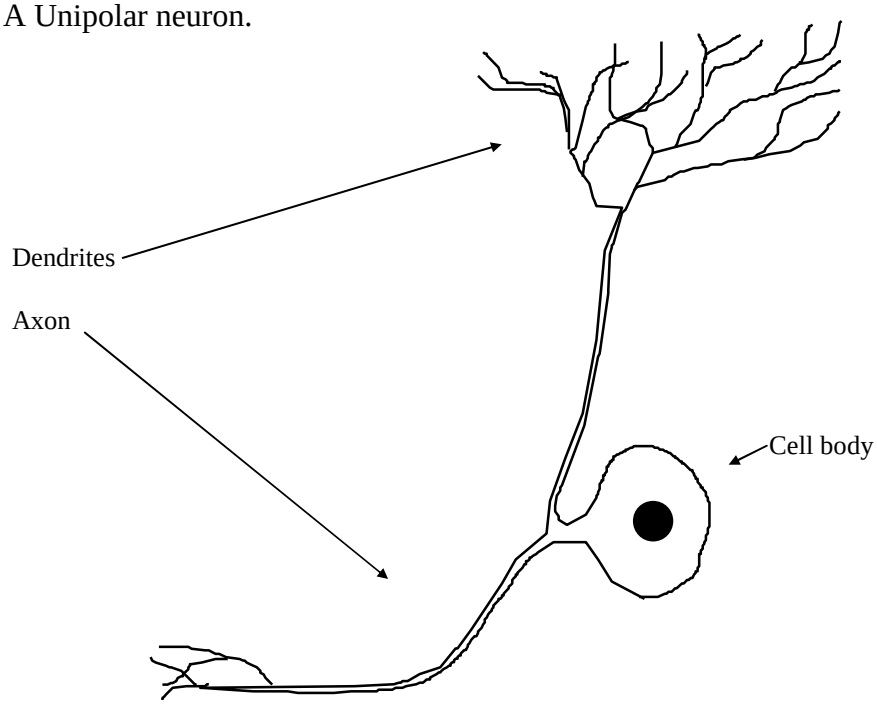


Figure 2.4 c)

A Unipolar neuron.



Multipolar neurons generally have dendrites growing directly from the cell body. Most of the neurons in the brain and spinal cord are multipolar neurons. Bipolar neurons have a dendrite growing out of one end of the cell and an axon growing out of the other. All adult neurons grow from bipolar neurons which are present in the foetus [5]. However, in the adult they are only present in the retina and several other small structures. Unipolar neurons have one branch which divides into two; one being the axon and the other the dendrites. This neuron acts as the wiring of the nervous system; the dendrites extending to the sensory receptors and the axon extending up into the brain or spinal cord. It is the long processes of these cells which we refer to as ‘nerves’, and they can run up to a metre in length. Unipolar cells are therefore the most common afferent neuron in the body [6].

The neurons themselves are surrounded by support cells called gial cells; these cells protect and supply the neurons. There is no need to examine the gial cells in detail, although they may be mentioned where relevant in the text. Gial cells outnumber neurons in the brain by 10 to 50 times [7].

2.2.3 Operation of biological neurons

When a neuron is at rest, before it becomes stimulated, it is said to be polarised. This means that, although the neuron is not receiving any electrical signal from other neurons, it is charged up. The polarisation of the neuron before stimulation is maintained by a constant flow of ions across the cell membrane [8].

Each neuron has associated with it a level of stimulus above which a nerve pulse or action potential will be generated. Only when it receives enough stimulation, from one or more sources, will it initiate a nerve pulse. This threshold level is called the threshold stimulus [9].

The action potential travels down the axon by means of ion interchange. Under normal resting circumstances the extracellular fluid (fluid outside the cell) has a high concentration of Na^+ and Cl^- ions while the intracellular fluid (fluid inside the cell) is rich in K^+ ions, phosphates and proteins [10]. This balance is due to the permeability of the cell membrane to the different types of ions. The result of this is that, when at rest, the membrane has a potential of 70mV across it.

When an action potential travels down the axon, chemicals in the membrane, known as channels, open and allow ion interchange between extracellular and intracellular fluid and the potential difference across the membrane changes by approximately 100mV. It is said that the membrane depolarises. After the action potential has passed, ions defuse back to their normal resting concentrations and the membrane repolarises. The nerve pulse may be shown diagramatically as in figure 2.5.

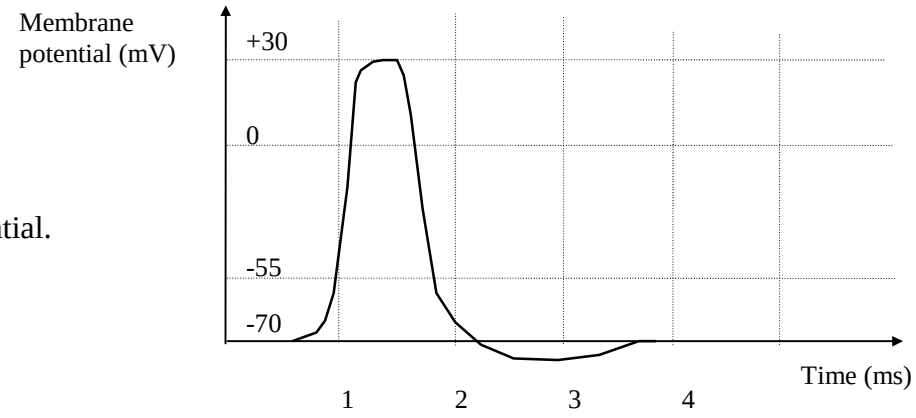


Figure 2.5
A nerve pulse
or Action potential.

Nerve impulses travel at between 12 and 120 metres / second.

The frequency of action potentials is the parameter which carries information about stimulation to the nervous system. The higher the frequency, the more stimulation is being produced. A single neuron in the nervous system can be stimulated by many such pulses. It is only when the stimulation reaches the threshold level (which also depends on the activity of neurotransmitter chemicals - see below) that an action potential is generated.

When the action potential reaches the synapse, it has to cross the small gap between one neuron and the next. There are two methods by which this can happen.

1. Electrical conduction.

2. Chemical conduction.

In the mammalian nervous system, electrical conduction across the synapse happens in cardiac cells and in the smooth muscle of the intestinal tract. The more common form of transmission is the chemical mechanism [11].

Chemical conduction takes place when the action potential reaches the synapse and the synaptic bulb release chemicals known as neurotransmitters. These chemicals cross the short space between cells and carry the neural information.

There are at least 50 neurotransmitters known and there may be over 100 [12]. The complete role of neurotransmitters in the nervous system is not understood, but a large part of the functionality and complex behaviour of the system is due to these chemicals.

Neurotransmitters can excite or inhibit cells and they can stimulate the release of other chemicals, which affect an area of brain in a more general way [13]. Examples include Somatostatin, which inhibits the release of growth hormone and Endorphins which have analgesic properties [14]. The operation of neurotransmitters is therefore very complex and attempts thus far to mimic their effect in artificial systems have not been entirely successful [15]. In many cases, the effect of the neurotransmitters is only understood in an affective domain, an example being Sertonin, which is known to affect sleep, mood and appetite.

2.3 The large scale structure of the biological brain

On a large scale, the biological brain is extremely complex, having many substructures and components.

There are several different ways of looking at the structure and it should be borne in mind while reading the following description that this is just one way of characterising the brain.

The biological nervous system may be split up into the Central Nervous System (CNS) and the Peripheral Nervous System (PNS). The CNS consists of the Brain and Spinal cord and might be thought of as the central control centre of the body. The PNS can be regarded as the communications network which connects the CNS with the rest of the body [16].

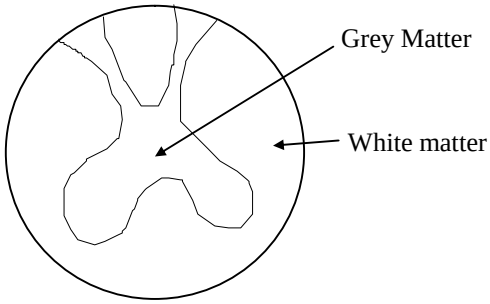
The PNS is not discussed here at length, but consists of Afferent and Efferent Neurons, together with the biological connections (in the form of axons) for the transport of signals [17]. Reference has already been made to these functions in section 2.2.2.

The CNS on its largest scale may be regarded as the spinal cord and the brain. The spinal cord acts rather like the main telephone exchange of the nervous system, the main nerves branch off from it to different areas of the body and it coordinates signals for return to the brain. The spinal cord itself is about 1cm in diameter and (in a human) runs from the base of the brain to approximately the small of the back.

From the spinal cord branch there are 31 pairs of nerves running to the different body systems which require control. The cord itself is protected by a tough membrane called the Dura Mater, Cerebrospinal fluid and the structure of the backbone itself [18].

If a cross section was cut though the spinal cord, one would find that the tissue consists of two types. The first is called White Matter and is the nerve fibres themselves. The second tissue is called grey matter and comprises the cell bodies and dendrites of the spinal cord nerve cells. The brain is also differentiated in this way, the connections (nerve fibres) being kept separate from the processing units (cell bodies). A cross section though the spinal cord is shown in figure 2.6.

Figure 2.6
A cross section though
the spinal cord.



The spinal cord is attached to the bottom of the brain through the brain stem, which apart from relaying messages from the spinal cord to the brain also controls vital involuntary functions such as heart rate, breathing and blood pressure [19].

The brain proper is so complex that only a brief description is possible in the space available. It is a symmetrical structure, having two halves or hemispheres, which are

further divided up into smaller areas called lobes. A description is given below of the major structures and their function.

1. Brain stem.

   This has already been mentioned and its function alluded to; it may be further subdivided into three smaller sections called the Midbrain, Pons and Medulla Oblongata.

2. The Cerebellum.

   A convoluted structure behind the main area of the brain; its functions are concerned with the control of movement, balance and coordination.

3. The Diencephalon.

   This structure lies deep within the brain and consists of the Thalamus, Hypothalamus, Epithalamus and Ventral Thalamus. Its purpose, among others, is to connect the two halves of the brain together.

4. The Cerebrum.

   This is the main mass of the brain and consists, like the spinal cord, of both white and grey matter. It has many functions; for example, it coordinates mental activity and is the centre for all consciousness [20].

2.4 The functions of the biological brain

As mentioned previously, the brain is extremely complex and only since the advent of brain surgery have its modes of operation been explored. In the 1950's, Roger Sperry began a series of experiments with patients who had the right and left hemispheres of their brains separated by cutting the corpus callosum, which is the bundle of nerve fibres connecting the left and right hemispheres of the brain. This drastic action was an attempt to cure epileptic seizures [21].

When, following the destruction of the corpus callosum, a patient was told to hold a spoon in their right hand, without looking at it, they could describe the spoon without difficulty. However, if the spoon was held in their left hand, they could not describe it.

This is because the speech centre is located in the left hemisphere of the brain, along with the processing for the right hand. Without the corpus callosum, which is the communicating link between both hemispheres, information from the right hemisphere (left hand) could not be passed to the speech centre in the left hemisphere.

In another experiment, the patients were shown a different image in each eye. In the left eye they were shown a spoon, and, in the right, a knife. The patients were able to describe what they saw as a knife, but if asked to pick up the object they saw, they would pick up the spoon.

In a further case, the right hemisphere learned how to speak and told the experimenter it wished to be an artist, while the personality in the left hemisphere indicated that its ambition was to be a racing driver. Only basic emotions such as anger and sexual stimulation could be passed between the hemispheres There was effectively two individuals living in the same body.

Experiments like these have shown that the left cerebral hemisphere controls language, scientific and numerical skills, whereas the right cerebral hemisphere controls imagination, musical skills and art [22].

Further illustration of brain function is provided by brain surgery under local anaesthetic. In these situations, it is possible to disable a small area of the brain using electrodes, and it is found that this corresponds with a small loss of function or reasoning. In more modern times, Nuclear Magnetic Resonance (NMR) scans have also provided many clues by illustrating blood flow to active regions of the brain when certain tasks are given to the subject. Examples of these smaller areas of the brain are: Wernicke's area, lesions of which produce a failure to recognise written words, a different section of this area is also responsible for the understanding of the spoken word; and Broca's area, which is involved in the formation of the spoken word [23].

From this work and others, it is possible to see that the biological nervous system is split up into substructures, each of which is responsible for some small subtask which

is integrated, with others, to make a whole. It has been postulated, therefore, that the brain is organised in a hierarchical manner, in which small independent subprocessors have their results integrated into the whole by larger networks of neurons [24].

These insights into the biological neural network give clues to suitable structures and hierarchies for use in the Evolutionary Artificial Neural Network. The more detailed descriptions will also be useful, particularly when considering further work and the generation of more complex systems based on biology, for example in the later chapters (chapters 8 to 13).

Chapter 3
Artificial Neural Networks

3.1 Introduction to chapter

This chapter explains the basic forms of the Artificial Neural Network (ANN). These models provide useful reference points during later discussions of the more complex Evolutionary Artificial Neural Networks.

The chapter deals with the early history of ANNs, the basic neuron model and a summary of the most commonly used ANNs. There are also sections showing some of the lesser used models and ANN limitations.

There are now many different ANNs in use and only the most common can be explored in the space available. Further information on the more exotic types may be obtained from the references, and in particular Simpson [1], who also provides tables comparing the available types.

3.2 The history of the Artificial Neural Network

The ANN is an attempt to model the construction and behaviour of the biological brain using man made technology.

3.2.1 McCulloch and Pitts

It is generally considered that the history of the ANN began in 1943 with a paper by Warren McCulloch and Walter Pitts called: ‘A logical calculus of the ideas imminent in Nervous Activity’ [2]. In this paper, McCulloch and Pitts presented the first descriptions of what they called ‘neuro logical networks’, although they did not say what purposes such networks could be put to [3]. The same authors then followed this in 1947 with a second paper entitled ‘How we know universals’ [4]. This paper describes neural networks capable of recognising patterns [5]. Even at this earliest stage in the development of ANNs, the models had recognisable components such as weights and thresholds [6].

3.2.2 Hebbian learning

Although McCulloch and Pitts had forwarded some interesting models, the subject of how the network would learn had still not been fully addressed. However, in 1949, Donald Hebb in his book: 'The Organisation of Behaviour' [7], put forward the idea that a neural network could learn by strengthening the pathways, within itself, which showed the highest levels of activity. Although his work was an attempt to explain learning in biological systems, it was also read by researchers interested in artificial neural systems [8].

3.2.3 Rosenblatt and the Perceptron

The first really successful attempt at constructing an artificial neural network was made by Frank Rosenblatt and others between 1957 and 1958. These early ANNs are often called Perceptrons. Rosenblatt wrote an influential book on neural networks called 'The Principles of Neurodynamics' [9]. This success encouraged many other researchers into the field, and in the 1950s and 1960s ANNs became a fashionable subject to study [10]. Perceptrons had successes in some interesting problems (for example pattern recognition), but despite much hype, failed to provide a key to the complex behaviour of the biological brain.

3.2.4 The publication of the book 'Perceptrons' by Minsky and Papert

Interest and research in ANNs continued to grow, until 1969, when Marvin Minsky and Seymour Papert published 'Perceptrons' [11]. This book stopped ANN research in its tracks. It showed that a single neuron of the type used in Perceptrons could not even simulate a simple Exclusive OR (XOR) gate. 'Perceptrons' was filled with innuendo against ANNs [12]. Due to the convincing argument of the book and the stature of its authors, most research in ANNs ground to a halt and researchers (except for a dedicated few), drifted off to more respected areas of research. (In the new edition [12], the authors rework many of their original ideas in the light of new research).

3.2.5 Decline and resurgence of interest

During the 1970s, ANN research was at a low ebb. Few individuals were interested after the publication of 'Perceptrons'; however, a few continued to fight the problems, and during this time, important progress was made, such as the design of new training algorithms.

In the early 1980s, ANN research took off once again. This was due to many factors, such as the availability of cheap computing power and the development of powerful learning algorithms such as Back Propagation [13], which overcame the problems that Minsky and Papert had pointed out in 'Perceptrons'.

At the present time, ANNs are one of the largest areas of interest within Artificial Intelligence (AI) research. They have had many successes, but also some failures. In an attempt to overcome these problems, ANNs are now being combined with other modern techniques such as Genetic Algorithms (GA) and Fuzzy Logic.

3.3 The basic Artificial Neuron

To explore the ANN, one needs to start by examining the artificial neuron [14,15,16]. In the biological model, the neuron fires and transmits an action potential, when its received stimulation is sufficiently high. In the artificial neuron, the stimulation is represented by inputs, which are numbers between zero and one (or sometimes minus one and one). These are weighted by a multiplication factor and then summed. If the result of summing these numbers is higher than a threshold, then the neuron fires. To illustrate this, consider the four input neuron shown in figure 3.1.

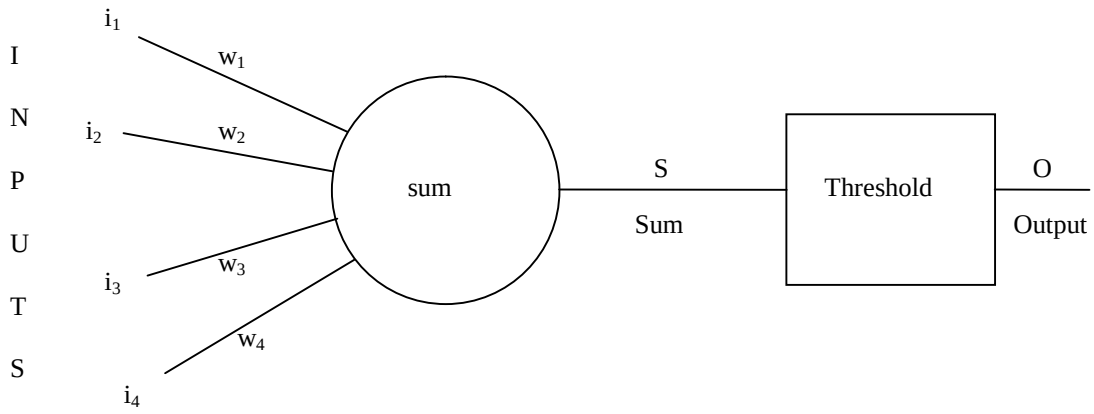


Figure 3.1

The simple artificial neuron.

The above diagram shows that the inputs to the neuron (which in the biological sense are the activities of the other connecting neurons, transmitted through the dendrites), are represented by *i* and the weighting factors of each of the artificial dendrites are represented by *w* . The sum of these inputs and their weights is denoted *S* . In mathematical terms:

$$S = i_1 w_1 + i_2 w_2 + i_3 w_3 + i_4 w_4 \qquad \text{(eqn 3.1)}$$

The threshold (which changes as training proceeds) is a simple binary level:

if S >0.5 then O = 1

if S <0.5 then O = 0

Threshold set at 0.5

The first artificial neural systems used simple binary outputs in this way. However, it was found that a continuous output function was more flexible. One example is the Sigmoid (overleaf):

$$O = \frac{1}{1 + e^{-S}} \qquad \text{(eqn 3.2)}$$

This function produces an output which is always between zero and one, and it is therefore often called a squashing (or activation) function. Other activation functions are also used, including: linear, logarithmic and tangential functions; however, the sigmoid function is the most common.

The preceding example may be formalised for a neuron of *n* inputs:

$$S = w_1 i_1 + w_2 i_2 + \ldots\ldots + w_n i_n$$

Which may be written conveniently as:

$$S = \sum_{x=1}^{x=n} w_x i_x \qquad \text{(eqn 3.3)}$$

Or, if the inputs are considered as forming a vector $\bar{I}$, and the weights a vector or matrix $\overline{W}$:

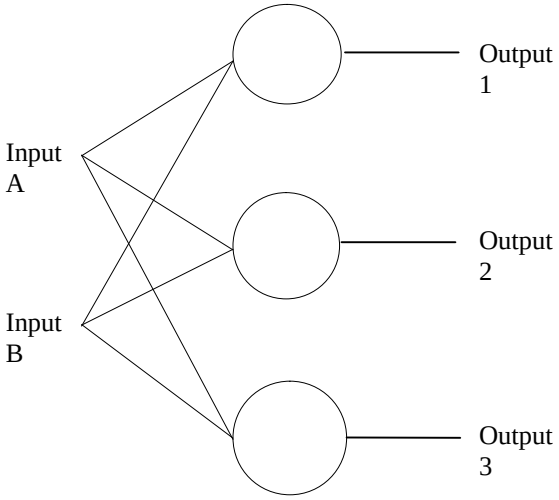
$$S = \bar{I} \,.\, \overline{W} \qquad \text{(eqn 3.4)}$$

In which case the normal rules of matrix arithmetic apply. The output function remains as in eqn 3.2. A further input is sometimes added; this is called the bias and remains constant under all conditions. This model is not the only method of representing a artificial neuron; however, it is by far the most common and is the model used to generate the experimental results in chapter 8. A neural network consists of a number of these neurons connected together.

3.4 The perceptron

Early ANNs, usually consisting of a single layer and using simple threshold functions, were called Perceptrons. A typical perceptron type network is shown in figure 3.2

Figure 3.2
A simple single layer perceptron with several outputs.



The network shown in figure 3.2 has a single layer of three neurons, each of which process two inputs and produce one output. The neurons are simple threshold units and have the same structure as the unit shown in figure 3.1.

Simple Perceptrons of this type were used extensively in the 1950s and 1960s and showed that ANNs could be taught to recognise patterns.

3.4.1 Perceptron learning

Before a network such as the one presented above can be used, it must be taught [17]. This is accomplished by altering the weights in such a way that the output from the network moves toward the value specified by the designer for a particular input.

At this stage it is necessary to introduce some of the terminology used to describe ANN training and learning.

First, before the network is trained, the user applies an input and the network calculates its output; this output is not usually what the user actually wants. The

output actually required is known as the *Target*. The difference between the target and the actual output is called the *Error*. An input and target used for training is called a *Training Pair*. In the training process we apply each training pair and alter the weights in such a way as to reduce the error.

The training method described above is referred to as *Supervised learning* and was developed by Widrow and Hoff [17]. The other form of learning encountered in neural networks is *Unsupervised learning*, in which there is no target, and therefore no error. The network organises itself to produce the required output.

Returning to perceptron learning, an error (δ) can be produced from the network which is the difference between the output (O) and the target (T). If there are several neurons in the output layer, then there will be several targets and therefore several errors. To simplify this example, we will consider a network with only a single output:

$$\delta = (T - O) \qquad \text{(eqn 3.5)}$$

This error is multiplied by the input to the neuron (i_x), and produces a change (Δ) which is added to the weight connecting that input with the neuron. A constant factor, called the *Learning rate (η),* is sometimes introduced to control the process. In eqn 3.7, the symbol w^+ represents the weight after it has been changed by the process described above:

$$\Delta_x = \eta \delta i_x \qquad \text{(eqn 3.6)}$$

$$w_x{}^+ = w_x + \Delta_x \qquad \text{(eqn 3.7)}$$

The threshold level is also adjusted by subtracting the error from it.

3.4.2 The Perceptron and the XOR problem

It was mentioned in section 3.2.4 that the publication of the book 'Perceptrons' by Minsky and Papert caused most researchers in the field to lose interest in ANNs. One

of the main arguments used in the book is that a simple perceptron cannot simulate a two input Exclusive OR (XOR) gate. This is known as the XOR problem [18].

Consider the two input neuron shown in figure 3.3.

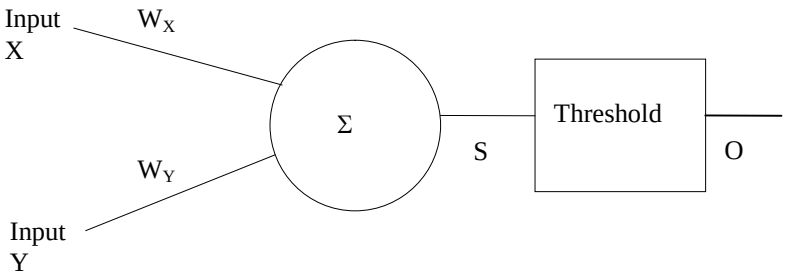


Figure 3.3
A simple two input perceptron type neuron.

The output from the summing stage of the neuron is:

$$S = XW_X + YW_Y \qquad \text{(eqn 3.8)}$$

This of course corresponds to the equation of a straight line:

$$Y = \frac{S}{W_Y} - \frac{W_x}{W_Y} X \qquad \text{(eqn 3.9)}$$

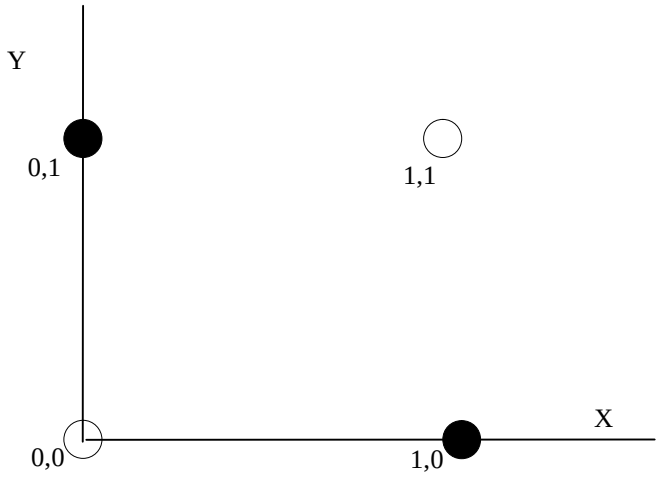
If the neuron is a simple threshold unit of the type shown above, then this line divides the region in which the output is a logic ‘1’ from the region in which the output is a logic ‘0’.

Figure 3.4 shows the output from a XOR gate plotted as a function of the inputs X and Y.

Figure 3.4
A graphical representation of the XOR function.

XOR Truth table.

X	Y	OUT
0	0	0
0	1	1
1	0	1
1	1	0



In figure 3.4, inputs which produce a ‘0’ output are shown as empty circles, and inputs which produce an output of ‘1’ are shown as filled circles. It can be clearly seen, that no matter where a line is plotted on the graph, the ‘0’ outputs cannot be separated from the ‘1’ outputs by that line; and hence, a simple neuron cannot simulate a XOR gate. This class of problem is called *Linear Separability* [18] and we say that the XOR function is not linearly separable by a single perceptron unit.

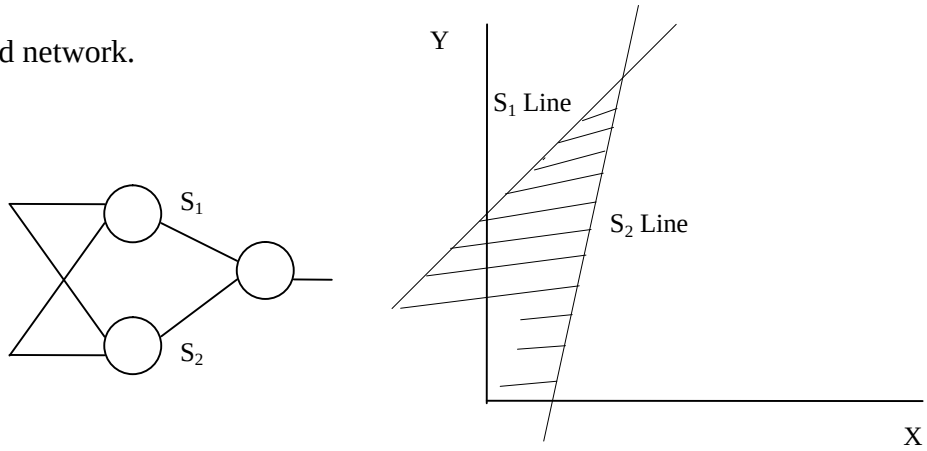
3.5 Feedforward networks

In section 3.4 above, simple networks of one layer were discussed; these networks are a subset of a larger class of network called the Feedforward or Associative network. A generalised feedforward network can have any number of layers. Its distinguishing feature is that it has no signal paths which feed data from the outputs back towards the inputs. In other words, data only flows in one direction: from the inputs to the outputs. This is in contrast to the Feedback or Auto-Associative network in which data can flow back from output to input.

3.5.1 Overcoming the XOR problem

A Multilayer Network is the key to solving the XOR problem. By adding neurons in other layers, it is possible to linearly separate functions which can not be separated using only one layer. An example of a two layer network is shown in figure 3.5.

Figure 3.5
A two layer
Feedforward network.



By adding extra neurons in new layers, it is possible to enclose any region of space and separate it from any other. Hence, by adding another layer of neurons, the network overcomes the problem of functions, (like the XOR), which are not linearly separable. In fact it can be shown that a two layer network (the inner layers are called *Hidden* layers), can solve any neural classification problem in which the outputs can be separated by two lines [19]. A three layer network can separate enclosed regions of space [20] . So a three layer network is all that is required to separate solutions in any arbitrary area of space; this is known as the *Kolmogorov theorem* [21].

The main problem with this approach was that when ‘Perceptrons’ was published there was no reliable method of training the hidden layers of a network. This is why the invention of algorithms, like Backpropagation (which could do this), gave ANN research fresh impetus in the 1980s.

3.5.2 Back Propagation

Back Propagation (BP) is a method for training multilayer feedforward networks. It works by training the output layer in the same way as was shown for the perceptron, and then propagating the error calculated for these output neurons, back through the weights of the net, to train the neurons in the inner (hidden) layers.

Back propagation was invented several times by several different researchers (ANN research is filled with such duplication of effort). Its basis is that the state of the network always changes in such a way that the output follows the error curve of the network downwards: that is, the error always decreases . This idea is called *Gradient descent;* it was first outlined by Amari in 1967 [22].

Back propagation proper was discovered independently, first by Werdos in 1974 [23], then by Parker in 1982 [24], and by Rumelhart, Hinton and Williams in 1986 [25]. None of these researchers were aware of the work that the others had done.

Back Propagation was not the first multilayer training algorithm to be used: that was the Boltzmann [26], outlined in section 3.5.3; however it is the most popular.

To understand how BP works, consider the network shown in figure 3.6 below.

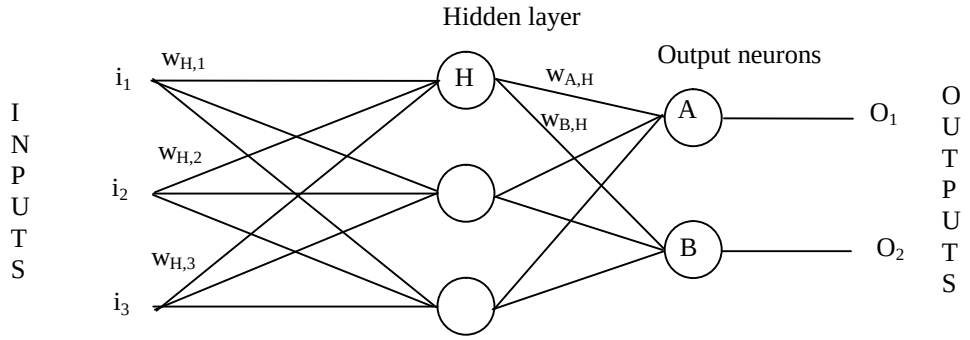


Figure 3.6 General Multilayer Feedforward network.

To show the general technique of BP training, consider the neuron in the hidden layer H. This neuron has three weights feeding into it: $w_{H,1}$ $w_{H,2}$ and $w_{H,3}$. The training technique applied to this one neuron can be applied to all the other neurons thoughout the network.

First, an input is applied to the network and the output is calculated. The output is then compared with the target and an error is calculated, as with the perceptron. In the perceptron case the neurons were considered to be simple threshold units; however, if they have a more complex activation function, then the error must be multiplied by the derivative of the activation function (for example the sigmoid function).

$$\frac{\partial O}{\partial S} = O(1-O) \qquad \text{(eqn 3.10)}$$

so:

$$\delta_A = O_A(1-O_A)(T_A - O_A) \qquad \text{(eqn 3.11)}$$

This can be used to calculate Δ_A , and therefore the new weights for neuron A, in exactly the same way as was done in the perceptron, by using equations 3.6 and 3.7 in section 3.4.1.

Where BP differs from simple perceptron learning is in the training of the hidden layers. The idea is to propagate the value of δ calculated for the output neurons, back through the weights, to the neurons of the hidden layer, and hence calculate a value of δ for these. This is done by multiplying the value of δ from each output layer by the weight connecting that output layer to the hidden layer neuron, and adding all the contributions together. Again, if we have an activation function, we must multiply by its derivative.

For the neuron H:

$$\delta_H = O_H(1-O_H).(\delta_A.w_{A,H} + \delta_B.w_{B,H}) \qquad \text{(eqn 3.12)}$$

and hence, having calculated δ for the neurons in the hidden layer, one can use equations 3.6 and 3.7 to calculate the weights associated with this neuron ($w_{H,1}$,$w_{H,2}$ and $w_{H,3}$). The process is then repeated for all the hidden layer neurons.

Although the process outlined above shows the method for discrete inputs and signals, the theory also applies to continuous functions [27].

Back Propagation has several problems associated with it, some of the more important of these problems are [28]:

Training Time. BP can take many iterations to train a network. There are now several new algorithms which aim to speed this process up; these are called advanced BP algorithms.

Network Paralysis. Sometimes the weights converge, during the algorithm, to large numbers; if this happens, training can slow down and stop; this is called Network Paralysis.

Local Minima. In more complex problems, the possible error surface of the outputs can have many bumps and valleys. Since BP always follows the error downwards, the network may not be able to get out of these valleys. This is known as the local minima problem.

Non Convergence. In certain circumstances, with many different training pairs, the network may fail to converge to a low error and oscillate between weights.

3.5.3 Statistical Methods

There are several methods of training neural networks which come under the title of *Statistical Methods* [29]. These methods involve selecting a weight from the network and changing it by a random amount. If the result of this reduces the error, then the change is retained. If the change makes the error worse, then it is discarded. When this is repeated over many cycles, the error decreases. This process may be written as an algorithm:

Apply an input and calculate the error of the network.
Select a weight at random from the network.
Add a small, random number to that weight.
Recalculate the error using the new weight.
If error is greater *then* discard weight and try again.
If error is smaller *then* keep weight and start again.

The training is simple to understand and therefore requires no further explanation. However, when such a algorithm is applied, it is found that the results of training are often disappointing. One reason for this is because the outputs of the network can get stuck in a local minima as described in section 3.5.2 (for a more in-depth description of this see reference [29]). Therefore, other, more complex variations of the basic statistical method have been invented. The two most popular are Boltzmann training (BT) and Cauchy training (CT).

1. Boltzmann Training.

The BT algorithm is similar to the simple algorithm outlined above, the differences being that BT sometimes allows a change which will cause an increase in error and decreases average size of the random adjustment in each training cycle. The reason behind this is that it allows the outputs to escape from local minima should they

become trapped, (unlike gradient descent, in which the error always decreases). A simple BT algorithm is shown below:

Define a variable T which will decrease as the algorithm runs

Apply an input to the network and calculate output and hence error

Make a random change to a weight (see below)

if change improves error *then* keep change, start algorithm again

else Calculate the function shown in equation 3.13 below

Calculate a random number

if mapping of random number is greater than P(c) then keep weight change, start algorithm again

else discard weight change, start algorithm again

$$P(c) = e^{-(\frac{c}{kT})} \qquad \text{(eqn 3.13)}$$

Equation 3.13 allows a small probability that the error in the network will increase. The variable T gives a higher probability of an error increase at the beginning of the algorithm, which decreases as time advances. Variable c is the change in weight, and k is a constant.

As the training runs, the size of the random weight change becomes smaller (allowing large changes at the beginning of the algorithm means that it is easier to escape from large local minima). The size of the weight change is calculated by integrating eqn

3.13 and using the result to map onto a random number (for further detail see reference [29]).

These methods come from the study of thermodynamic systems in physics, where it is possible to calculate the probability that a particle can have a certain kinetic energy. The average kinetic energy decreases as the temperature decreases, (which is simulated by the reduction of the variable T, above).

A closely related algorithm uses the Gaussian distribution to choose the weight change size:

$$P(c) = e^{-(\frac{c^2}{T^2})} \qquad \text{(eqn 3.14)}$$

The variable T can be calculated from:

$$T = \frac{T_0}{\log_{10}(1+t)} \qquad \text{(eqn 3.15)}$$

which represents cooling rate in a Boltzmann system (T_0 = Initial Temperature, t = Time).

It is possible to simulate the cooling of a metal using these formulas as they model the particle kinetic energy probability distribution in the system; such a process is called annealing, and the computer model is called simulated annealing. More complex operations, such a phase changes, can also be included in the system. It should be noted, however, that Boltzmann training is often extremely slow.

2. Cauchy Method.

Another important method uses the Cauchy distribution. This is, in operation, very similar to the Boltzmann distribution. However the probability function is expressed as:

$$P(c) = \frac{T}{T^2 + c^2} \qquad \text{(eqn 3.16)}$$

and the temperature function:

$$T = \frac{T_0}{1+t} \qquad \text{(eqn 3.17)}$$

Cauchy training is easier to implement than the Boltzmann since eqn 3.16 may be integrated by normal methods whereas eqn 3.13 or eqn 3.14 require numerical integration. Integrating eqn 3.16 gives the following expression:

$$p = \eta(T\tan(x)) \qquad \text{(eqn 3.18)}$$

Substituting a random number for x yields the probability of that weight change. The symbol η is a constant corresponding to the learning rate of the system.

Such systems are useful in the study of interesting ANN topologies, such as the ones encountered in evolutionary systems; this is because they can train most structures of network, which is not the case with BP.

3.6 Feedback networks

It has already been noted, in section 3.2.5, that the resurgence of interest in ANNs during the 1980s was caused by several factors. One of the factors often cited, was the publication, by John Hopfield in 1982, of a paper in which a new type of network was presented. The paper was entitled ‘Neural Networks and physical systems with emergent collective computational abilities’ [30]. These networks have feedback paths from the output to the inputs, and are termed Recurrent, Feedback or Auto-

Associative. The simplest are called Hopfield nets [31,32]. They differ from the feedforward networks in that, whereas a feedforward net can classify a input, (for example in the case of character recognition), a Hopfield net can reconstruct the desired output from a corrupted input. That is, it can *associate* a structure stored in the values of its weights with an input.

3.6.1 Hopfield networks

Consider the network shown in figure 3.7.

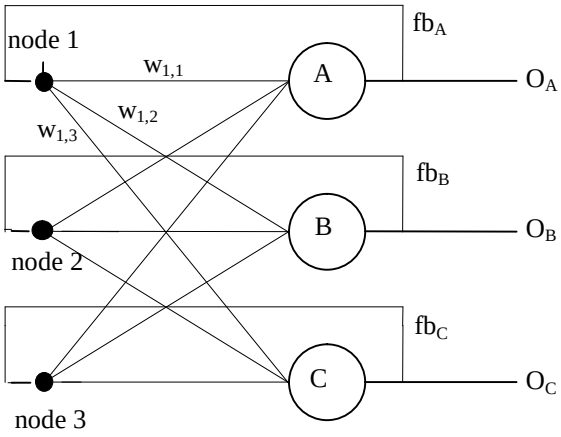


Figure 3.7 A Hopfield Network.

This network has three neurons A, B and C. The output from these neurons is fed back by the paths fb_A, fb_B and fb_C to the inputs (marked node 1, node 2 and node 3). The nodes are just distribution points for the weights; they do not perform any arithmetic operation on the signals. The first weight leaving node 1 is labelled $w_{1,1}$, the second weight $w_{1,2}$, etc.

The two important operations on the Hopfield net are, of course, extracting information from it and training it.

1. Extracting information from the Hopfield net.

The neurons in the Hopfield net operate in exactly the same way as the basic neurons which were outlined in section 3.3, and like the basic neurons, they also come in both a binary form and an activation function form. Signals are also propagated around the network in the same way as for any of the other networks which have thus far been discussed.

The network functions as follows:

Force the output vectors (O) to the starting state (inputs)
Allow the network to recalculate its outputs
When the network outputs stop changing, read off outputs

So the input vector is forced onto the outputs of the network and the network cycles round with the values generated by these inputs, until it settles: it is said to *relax*.

2. Learning in a Hopfield net.

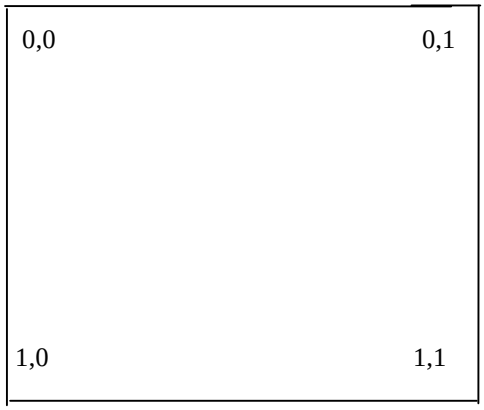
The Hopfield net has no target vectors; the training is therefore unsupervised. The weights are calculated directly from the required outputs using the formula:

$$w_{y,z} = \sum_{x=1}^{x=m} O_{y,x} O_{z,x} \qquad \text{(eqn 3.19)}$$

Where m is the total number of desired output vectors. $O_{\psi,x}$ is the ψ th component of the desired vector for that output (output x). This training is sometimes refered to as *one shot* because there is not iterative element.

The outputs from a binary Hopfield network may be represented as a square if the network has two output neurons, a cube if the network has three, and a **n** dimensional hypercube if the network has n outputs. Each vertex of the figure corresponds to a possible output of the network. This situation is illustrated in figure 3.8.

Figure 3.8
A square representation
of the outputs from a two neuron
binary Hopfield network.



As the network relaxes, it shifts through each vertex until it comes to rest at its stable point.

Being a recurrent system, a Hopfield network is not unconditionally stable as a feedforward network is. It can, under some circumstances, display saturation, oscillation and chaotic behaviour. S Grossberg and M Cohen, in a paper called 'Absolute stability of global pattern formation and parallel memory storage by competitive neural networks', published in 1983 [33], showed that if the weights in the network are symmetrical, that is: $w_{a,b} = w_{b,a}$ and $w_{a,a} = 0$, for all **a** and **b,** then the network would be unconditionally stable. If equation 3.18 is studied, it will be found that it produces just such a set of stable weights. Another way of looking at this is to say that the weight matrix must be orthogonal.

Hopfield was a Physicist, and used the networks to simulate quantum particle states. The total network state is therefore sometimes expressed as an energy.

The stable points in the network output can be shown to correspond to low points in the energy function of the network. The energy of the network may be expressed as:

$$E = -\frac{1}{2}\sum_{x=1}^{m}\ \sum_{y=1}^{m} O_x O_y w_{x,y} + \sum_{x=1}^{m} O_x \theta_x \qquad \text{(eqn 3.20)}$$

where **m** is the total number of output neurons and θ is the threshold function of the neuron. It should be noted that some authors use other expressions for the energy.

The Hopfield network may be integrated with the statistical methods outlined earlier to produce a network known as the Boltzmann machine (BM). The BM uses statistics to calculate the outputs of the network. It is important because generalised networks can be modelled using this technique.

3.6.2 Other feedback networks

Several other important feedback structures now exist within ANN technology. These networks are many and varied; one which is worth noting is the Bidirectional Associative memory or BAM [34].

It has already been noted that an important difference between the feedforward network and the recurrent network is that the feedforward net classifies inputs, whereas the recurrent network can reconstruct the input. This means, that in a real sense, the feedback network has a memory and feedback networks are sometimes called *Associative Memories*. The two different networks presented, therefore, have very different modes of operation.

The BAM adds another layer of neurons between the input nodes and the output neurons, (like a hidden layer in a feedforward net). This allows the network to be able to reconstruct a different memory from the input vector. That is, the network can be programmed to associate a unrelated output with its input. The action is known as *Hetero Associative* (in different literature these names are often used interchangeably - leading to much confusion). It has been pointed out [34], that the action of a BAM is similar to the human action of receiving a stimulus and remembering a forgotten incident; for example, smelling roses and remembering a summer's day. The BAM is trained in a similar way to the Hopfield net.

3.7 Other important ANN structures

In section 3.1, it was mentioned that there are many different ANNs now in operation. There are, however, several topologies, other than the ones covered thus far, which should be included for completeness by virtue of their importance and/or widespread use.

3.7.1 Competitive Learning

In Competitive Learning [35], an input is applied to a layer of neurons and the output of these neurons calculated. The neuron with the largest output 'wins' and its output is made to be one, while the others are made zero. The weights of this neuron are then changed so that the neuron has an even greater output from the given input. This idea is based on Hebbian learning mentioned in section 3.2.2. The idea is that each neuron in the network can classify a certain type of input. Neurons which behave in this way are sometimes known as *Kohonen* or *Self Organising* neurons and a popular type of network which uses this principle is the *Counterpropagation network*.

3.7.2 Adaptive Resonance

The Adaptive Resonance network [36], (also known as ART), is quite a complex structure compared with the other networks illustrated thus far. ART networks consists of two layers, the *recognition layer*, and the *comparison layer*, as well as other support structures. The recognition layer classifies inputs and operates as a competitive network. The comparison layer compares the output of the recognition layer with the incoming vector and changes the weights if they are not sufficiently similar. The major advantage of ART networks is that they can learn throughout their use, unlike the other ANNs mentioned so far, which have to be trained in a separate phase from their use.

3.7.3 The Cognitron

The Cognitron [37] is based heavily on brain anatomy and physiology and in particular on human visual pattern recognition. It is a multilayer network and contains competitive type neurons. The major difference between this and the networks mentioned so far is that the cognitron contains neurons which function in both inhibitory and excitatory modes, (see section 2.2.3 for an explanation of this behaviour in the BNN).

3.8 Implementations

There are many implementations of neural networks now in use. Again, Simpson [1] is a good source of information on this subject, using tables to compare the various implementations available at the time of publication. Aleksander and Morton [38], is also an interesting source of information.

Although methods such as hydraulics [39] have been used in the past to implement neural-like networks, the most obvious and widely used implementation is by software. Since most of the rest of this work uses that method, it will not be further elaborated on at this point. Two other popular approaches are worth mentioning and these are described below.

3.8.1 Hardware

The first ANNs were implemented in hardware and this continues to be a popular approach. Analogue [40] and digital [41] circuits are both used, although digital circuits can be rather complex compared with the analogue approach. In recent years user specified integrated circuits have come into popular use, these are known as ASICs (Application Specific Integrated Circuits). These circuits allow the designer to program the required circuit. Especially popular are FPGA's (Field Programmable Gate Arrays) [42,43]. These devices have a special and important place in the evolutionary ANN implementations as presented here. The use of such circuits allows real time, true parallel ANNs to be implemented, (rather than serial types, as in the case of the computer program).

3.8.2 Optical

Less popular than the electronic hardware ANNs, but again important, is the use of optical hardware for implementing ANNs [44]. The allows the ultimate in speed and high density. Holographic methods are a popular research topic. However, since many optical systems are often difficult or impossible to reconfigure in real time, they present major problems with training.

3.9 ANN Limitations

The three networks listed in detail above:- Feedforward, Hopfield and BAM, have three very different modes of operation. The Feedforward classifies inputs, the Hopfield reconstructs inputs and the BAM associates one input with another data set. It is therefore difficult, with such a variation in operation, to generalise about the behaviour of networks. Other types exist which behave in a different way to the examples listed above. However, there are several points which should be made on the limitations of these networks in real world problems.

1. The application of networks to non-specific problems.

When the ANN is presented with specific constrained problems, it performs very well. For example, in character recognition tasks it can out-perform a human operator. However, the human brain can sort out problems with many variables and distractions. This is currently beyond the capability of the ANN. This problem will be discussed in later chapters.

2. Training.

Using the most popular training method (BP), networks will sometimes not train. It is largely a matter of experience to set up the training in such a way that the network will train successfully.

3. Network design is a matter of experience.

Designing a network requires the operator to make judgements on the network structure based on earlier results; for example, the maximum number of memories a Hopfield network can store is theoretically 2N, where N is the number of binary

neurons in the network [45]. However the actual number is found empirically to be much less. There are no readily available formulas for network structure design. This is a specific point which the research presented here addresses; it is also the subject of much modern research.

4. Time series modelling.

Although the networks are very good at modelling and classifying spatial data, they are very poor at interpreting time varying data. Much current research is directed at this problem.

Listed above are some of the problems which are currently associated with ANN technology. These problems are perhaps the most important, but there are many others also. It is to address these points that this research is presented.

Chapter 4

Biological Evolution and Embryology

4.1 Introduction to chapter

The ideas outlined in this chapter underpin the operation of the algorithms used to design the ANN structures presented in the results. To understand the operation of the artificial computer models of these biological systems, it is first necessary to appreciate the natural processes themselves. An in-depth knowledge of these systems also allows the critical discussion of artificial evolutionary algorithms, which are presented in later chapters.

Evolution is the gradual change of one species into another over a period of time. The chapter gives a history of Evolution, then an outline of the main ideas inherent in the theory. The modern aspects of Evolutionary theory are also highlighted. The same structure is used to outline the history and theory of Embryology. Finally, Evolution and Embryology are compared and contrasted. This is followed by a discussion of the Evolution and Embryology of the nervous system and brain.

4.2 The History of Evolution

Contrary to popular belief, Charles Darwin did not invent the concept of Evolution; the weight of evidence from Fossils, Embryology and general observation of animal communities had long been acknowledged. The idea has been used and abused by thinkers, scientists and philosophers from Greek times.

4.2.1 Evolution before Darwin

The first recorded writer on the subject was the philosopher Empedocles of Agrigentum, who lived from 495 to 435 BC. Although his ideas seem strange by modern standards, he never-the-less believed in a form of evolution. Many others in the Greek philosophical tradition also toyed with it, including Aristotle. So popular was the concept that animals were not created but evolved one from another that the

early Christian church also taught the idea, and in particular Saint Augustus who made it a cornerstone of his philosophy [1].

During the Dark Ages, most scientific development in Europe ground to a halt; although it continued in the Arab world. It was in the Sixteenth Century that scientific progress restarted in Europe. During this period Evolution again became a topic of debate. A new philosophy was being practised; this advocated experiment and observation of the natural world. One of the founding fathers of this new breed of natural philosophers was Sir Frances Bacon, who lived between 1561 and 1626. Bacon published a book called ‘Novum Organum’ [2], which contained some of the first speculations on variations in the animal and plant kingdoms. Between Bacon’s time and Darwin’s many authors published accounts which pointed to some form of evolution in the natural world. These attempts at explaining the relationship between plants and animals culminated in the work of Charles Darwin’s grandfather, Erasmus Darwin. Erasmus was one of the leading scientists and thinkers of his time, and his adventures are at least as interesting as those of his more famous grandson. In 1794 he published a book called ‘Zoonomia’ [3], which contains the strongest ideas on evolution before Charles Darwin [4]. An example of his ideas are clearly seen from this poem which he wrote:

Organic life beneath the shoreless waves
Was born and nurs’d in oceans pearly caves;
First, forms minute, unseen by spheric glass,
Move on mud, or pierce the watery mass;
These, as successive generations bloom,
New powers acquire and longer limbs assume;
Whence countless groups of vegetation spring,
And breathing realms of fin and feet and wing.

4.2.2 Charles Robert Darwin

Charles Darwin was born in 1809 in Shrewsbury. His family were medical doctors, and Charles studied medicine at Edinburgh. He soon found that he had little interest in the subject, and transferred to Cambridge with the intent of entering the church. He was not a distinguished scholar.

In 1831 he jumped at the chance to become a naturalist on a surveying ship called the 'Beagle'. The trip aboard the Beagle took three years and during this time Darwin had the chance to observe much of the natural world. It was at this time that he started to consider the relationship of the plants and animals to their enviroment, and set out along the road which would lead him to his theories on the origin of species.

Darwin was both a skilled geologist, (he did important work on South American geology) and a first rate naturalist (his books on Barnacles are still the standard texts). It was connecting these two disciplines which brought him to his ideas on evolution.

As mentioned in section 4.2.1, the idea of evolution had been around for a long time before Darwin: however, Darwin's contribution was to put these ideas in a solid theoretical framework and add one more idea of his own, that of *Natural Selection.*

It was the idea of Natural Selection (commonly called survival of the fittest) which caused Darwin's initial friction with the church and it is also this aspect which is still argued about among biologists today. Evolution was obvious; natural selection was Darwin's attempt to explain the mechanism behind it.

It should be noted that there were other researchers working along the same lines as Darwin at this point, but Darwin was the first in print. He just beat another worker, Alfred Russel Wallace into publication (they later produced a joint paper on evolution) [5]. The book which outlines Darwin's theory is: 'The origin of species by means of Natural Selection, or the preservation of favoured races in the struggle for life' [6].

4.2.3 After Darwin

When Darwin published his theories, few facts were known about the science now known as Genetics. It was even assumed that animals could pass on characteristics to their offspring, which had been acquired during their lifetimes.

Genetics had its origin with Father Gregor Mendel who was an Abbot of a monastery at Brunn in Moravia. Mendel lived between 1822 and 1884 and actually published his results three years before the publication of *Origin* but they were passed over until their rediscovery in 1900. It is thought that his experiments stopped when his superiors pointed out that research into Genetics and Evolution might not be beneficial to his promotion prospects within the church. Mendel experimented with pea plants and uncovered how genetic traits are passed on from one generation to another.

In the 20th century much more work has been done, both investigating evolution and in genetics, leading in 1953, to the discovery of the structure of DNA, the carrier of the genetic information itself, by Watson and Crick at Cambridge [7]. Some of the more modern theories which were proposed for evolution are outlined in section 4.4 on recent work [8].

An understanding of the process of biological inheritance and genetics is very important for comparison with the artificial Genetic Algorithm, to understand the mechanism of Evolution and Embryology and also in the discussions which will be presented in the latter part of the thesis. However, this subject is too large to cover in the main text, and since it is refered to several times in the course of discussion, it is included in appendix 6 for reference by the reader when necessary (see also reference [8]).

4.3 The formation of the theory

The best way to illustrate the formation of the theory is to outline the evidence which led Darwin and Wallace to their conclusions. This evidence is conveniently divided into, firstly, that for evolution of animal species, one into another, which as mentioned in section 4.2.2 and was obvious long before Darwin’s time; and secondly, the

evidence which led to the formation of the theory of Natural Selection as the cause of evolution.

4.3.1 The evidence for evolution

Before the era of DNA finger printing and other modern techniques, evidence for evolution came from three main sources:

- Anatomy and Embryology
- The Fossil Record
- Geographical evidence

1. Anatomy and Embryology.

The evidence for evolution in anatomy consists mainly of two different points. Firstly, there are obvious similarities between groups of animals. Comparative study shows that some animals are more closely related than others. Similarities are also found between animals which seem at first sight quite different, for example, the horse and the rhinoceros. Secondly, many animals have the remnants of limbs which are no longer used. Examples include, the hind limbs of whales which are buried deep with in the body and the limbs of snakes, which are also internal in the animal. This infers that these creatures once had limbs which have now become so small that they are useless remnants within the body.

The evidence from Embryology requires more explanation and is vitally important to the philosophy of the research presented within this thesis. Embryology has a deep and profound connection with evolution which may not be at first apparent. When a animal develops, it starts as a single cell (the fertilized egg), then divides to become a multicelled organism. As it grows it passes through stages of more and more complexity, until at birth, it is a recognisable member of the species. The parallel with the fossil record is obvious. However, the connection is much deeper than even this: as an human embryo grows, it develops gill pouches like a fish. These disappear as development continues. The heart is also an example; as development progresses, the embryo develops a heart which is partitioned like that of an amphibian. This later

develops into the multi-chambered heart of a mammal. As a final example, the human foetus has a tail which is reabsorbed into its structure during development (for further comparison refer to figure 4.2). It is as though the genetic sequence which causes an animal to develop in the womb does so by replaying the evolutionary development of the organism.

2. The Fossil Record.

The most direct evidence for evolution comes from the record of animal life contained within the fossil record. The oldest strata contain the simplest organisms, and it is possible to follow animal development through invertebrate forms to fish and onto amphibians, reptiles and mammals. In addition, several lines of animals can be traced from their origins to their present day forms with few breaks in the record; examples include the horse and elephant families, both of which are quite different today to their directly traceable ancestors (the original horse had three toes and was the size of a dog; all the intermediate forms are present in the fossil record between this and the modern horse).

3. Geographical evidence.

The final evidence for evolution comes from the geographical distribution of animals. Large groups in the classification of animals such as Classes and Orders of animals are widely distributed over the surface of the globe. Subgroups such as Families are less widely separated and small groups such as Genera are localised. This gives strong evidence as to the changes which occur in animals as they radiate outwards from their points of origin.

Strong evidence also came from variation in domestic animals. It can be shown that all dogs are descended from wolves, and cattle from the ox. This is an example of man acting as the mechanism of selection in the animal [9]. Darwin pointed this out in several of his books (although its significance was debated by other authors).

Outlined above are the main points which could be considered evidence for evolution during Darwin's lifetime. Since these early days, biologists have had the time to look for evidence of actual evolution in the wild. Advances in genetics have meant that mutation and the nature of inheritance is now well understood and computer models have shown how evolution operates in populations.

4.3.2 Natural selection

During his travels aboard the Beagle, Darwin was profoundly struck by his visit to the Galapagos Islands. He noticed how various species filled different ecological niches on different islands and conjectured that these animals had evolved to fill the varying environments available to them. Because the islands were close together and similar animals existed on them, it was still possible to trace the relationship between various species. The best known examples which display these facets are Darwin's Finches (Geospiza). These birds do not exist on the mainland and on the Galapagos they have evolved into several different sub-species occupying different habitats and with different habits, but still with a traceable common ancestry.

From this evidence and much thought, Darwin concluded that if a species exists in a changing environment, those members of the species best suited to survival in that environment will survive in larger numbers and be more successful than others of the species. These 'fit' members will therefore have a better chance of passing on their genes (and therefore traits) to the next generation. This is Natural Selection [10].

An example of this process at work is the tortoise (of which there are several species on the Galapagos). During periods of drought the lower leaves on the trees tend to die before leaves higher up the tree. Only tall tortoises with long necks can reach these higher leaves and so they survive to pass their genes to their offspring. These next generation tortoises are said to be 'fitter' (hence survival of the fittest).

4.4 Modern perspectives

'I am convinced that Natural Selection has been the main but not exclusive means of modification'.

So wrote Darwin at the end of his introduction to the *Origin*.

Thirty years before Darwin published his masterpiece, a Frenchman named Jean Baptist Lamerck had published his views on evolutionary mechanism. Lamerck suggested that the process was due to the adaptation of animals to their environment, using acquired traits. Starting at that time and continuing until the present day arguments about the details of the mechanism have raged among biologists. Listed below are some of the other, more modern models, used to explain evolutionary mechanism.

1. Selfish genes.

This idea revolves around the preservation of the gene as the mechanism behind evolution. Each animal competes with others to pass on its genetic material to the next generation. G C Williams first published the idea in 1966 [11]. The concept was made famous by Richard Dawkins in his book 'The Selfish Gene' [12].

2. Group Selection.

This theory models a group of animals in an environment which has limited resources. The animals form a hierarchy with those at the bottom unable to breed. The theory was forwarded as a possible evolutionary mechanism in 1962 by V C Wynne-Edwards [13].

3. Kin Selection.

In 1964, W D Hamilton extended Wynne-Edwards ideas mathematically to include a degree of 'relatedness' of animals in a group and therefore managed to model their chances of sharing genes (and therefore similar behaviour). Hamilton went on to describe the behaviour of social insects using his model (with some success).

4. Punctuated Equilibra.

Geologists had long noticed that animal species evolved slowly in the fossil record and then seemed to totally disappear, while other, new species suddenly appeared

apparently from nowhere. Niles Elredge and Stephen J Gould [14] suggested in 1972 that this situation was the norm: slow change, with points of very rapid development. These periods of fast development where thought to correspond to periods of rapid environmental change. Recent computer simulations have borne this idea out. In 1975 Steven Stanley further developed this idea into his theory of species selection.

5 Extinction.

The fossil record shows several events which were catastrophic for life on earth. These represent periodic mass extinction. The most drastic of these happened at the end of the Permian era, 250 million years ago. It is estimated that 95% of all life on earth was wiped out. Only simple organisms were left, and nature practically had to start all over again to build new populations of plants and animals. The cause of these episodes is not fully understood. One possibility is vulcanism; another (currently the most popular) is an extra terrestrial source, such as a large meteor, supernova or comet.

It should be apparent from the above points (and there are many others) that the exact mechanism of evolution is not fully understood. Natural Selection, has been shown to play a part in it, but other influences are open to debate. The truth is possibly that all these mechanisms have a role to play in the natural world [15] (reference [15] is also a good general overview of evolutionary mechanisms).

4.5 The History of Embryology

The following section on Embryology is shorter than the preceeding section on Evolution. This is because much of the important material has already been covered in the discussion of Evolution.

Embryology is the science dealing with the development of the embryo and foetus. It has proved less contentious than Evolution. This is perhaps unusual, as its findings are equally revolutionary. As mentioned eariler, Evolution and Embryology share certain deep connections.

Early natural philosophers considered that the egg or sperm contained a miniature adult (called an Homunculus). This then grew in the womb, to become a baby at birth. Such a view is known as *preformationist.*

At the end of the eighteenth century, it became obvious (thanks to new scientific methods) that this was not the case. By the nineteenth century scientists had begun to view the process as one of evolution (not, of course, in the Darwinian sense). This was not the first time such a option had been aired. Aristotle had considered embryological development in much the same light.

The modern explanations of foetal development are descendants of this view. The embryo passes through a series of different stages of development, becoming more complex as it grows. In modern times, the role of DNA and the operation of genetics has been discovered, and the understanding of the process is now a good deal more sophisticated [16] (this reference also gives a useful overview of embryology).

4.6 Embryological development

Not all animals reproduce sexually. Protozoa and other primitive groups do not have sexual processes (reproduction is by division of the adult). Others, such as some insects reproduce *parthenogenetically*, that is, their eggs develop without being fertilised by sperm. This process can also be stimulated artificially in other groups of animals. The egg and sperm are known as *gametes*. Once development starts in

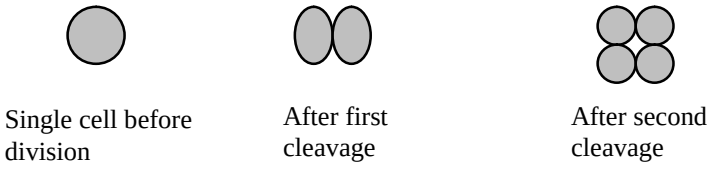
organisms which reproduce parthenogenetically, they develop in a similar way to those with gametes [17].

In mammalian species, the unfertilized egg is about 0.1mm in diameter, it is just on the scale of visibility to the human eye. This makes it one of the largest cells in the body; the male sperm is, on the other hand, one of the smallest.

Only the physical development of the embryo will be discussed here. For a overview of the role of genetic material refer to appendix 7.

Once the sperm has fused with the egg, development proceeds by cell division. Mammalian development proceeds slowly compared with other animal forms. Goldfish eggs cleave (divide) every 20 minutes, frog eggs each hour but mouse eggs only cleave for the first time after 24 hours and thereafter every 12 hours. In mammals after four days, the embryo may consist of fewer than 100 cells. The reason for this wide variation of cleavage time is not known [18]; the process is illustrated in figure 4.1.

Figure 4.1
The beginning of development of a mammalian embryo



As can be seen from the diagram that, each time the cells divide, their number doubles.

After only four days, even although there may be less than 100 cells (as noted above), important changes have begun to take place. These include:

- The cells are approximatly the same size as adult cells.
- The operation of RNA and therefore the capacity to make protein is present
- The metabolic rate has increased.

- The genes of the embryo have begun to function.
- Cell differentiation has begun.

This last point is important since differentiation gives rise to the different tissue types within the body.

Within the solid ball of cells, formed by cell division, a cavity appears; this is known as the *blastocoele*. As it enlarges the embryo resembles a hollow sphere. Within the cavity, sticks out a knob of cells which will become the adult animal, the surrounding cells becoming the placenta. Near this time the developing embryo becomes implanted in the uterus wall [19].

Development continues by cell division, differentiation and the flow of material within the embryo. The cells first become differentiated into what will become the skin layer of the animals, called *ectoderm,* the middle layer (which will consist of muscle, blood system, etc) called the *mesoderm* and inner layer, which will line the digestive tract, called the *endoderm.*

As development proceeds, specialised tissue such as neural material (with which we are specifically concerned) becomes differentiated and migrates to what will become the nervous system of the animal [20] (as will be discussed latter, neural material has its origins as part of the ectoderm).

This process continues until the foetus reaches term and is born. In section 4.8.2, the specific development of the nervous tissue and system will be considered.

4.7 Comparison between Evolution and Embryology

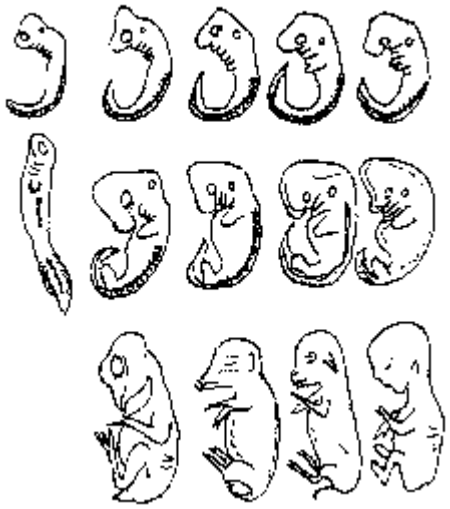
During the preceding sections of this discussion, it has been pointed out several times that Evolution and Embryology have many striking similarities. In the fossil record, the first primitive life appeared around 3.5 to 4 billion years ago. These were simple single celled organisms. The next major step was the development of multi-cellular

organisation. Much later came animals with backbones, fish, amphibians, reptiles and Mammals.

This of course bears a striking resemblance to the development of the embryo, further strengthened by the appearance of primitive features such as gill slits and a tail. The comparative development of five embryos is shown in figure 4.2.

Figure 4.2

The embryos of a fish, chick, pig, rabbit and man at different stages of development.



After diagram in: 'The penguin book of the Natural World", edited Martin et al

Notice that the most primitive form (the fish) diverges from the others first. The reasons for embryology following this evolutionary pattern so closely are not known [21].

There are, of course, many differences between Evolution and Embryology. For example, there is no analogy in evolution to the development of the embryo by cell cleavage. The algorithm which is outlined later in this work and used to grow ANNs is derived both from evolution and embryology and shows features of both.

4.8 The Evolution and Embryology of the nervous system

The final section of this chapter is concerned with the specific development of the nervous system, both in terms of evolution and embryology. The reader should again note the similarities between both processes. The study of these processes give clues both in terms of nervous system architecture and the growth algorithms themselves which can help in the design of artificial systems.

4.8.1 The Evolution of the nervous system

In a single celled animal, the various parts of the organism can communicate by chemical means. However, in multicellular organisation, communication becomes more difficult. Some primitive multicellular animals still employ chemical means, but this is slow and becomes increasingly difficult if the cells within the animal are specialised.

It is thought that the origins of the nervous system were specialised sensory cells in the skin of the animal (this also ties up with embryology; see section 4.8.2). These cells would have been able to give the animal some limited feedback from its environment (for example, to detect food etc). Some neurons in higher animals have retained their positions as sensory cells, others have migrated inwards in become interneurons. An example of a primitive animal showing a simple nervous system is the Hydra, in which the neurons are equally distributed throughout the body [22].

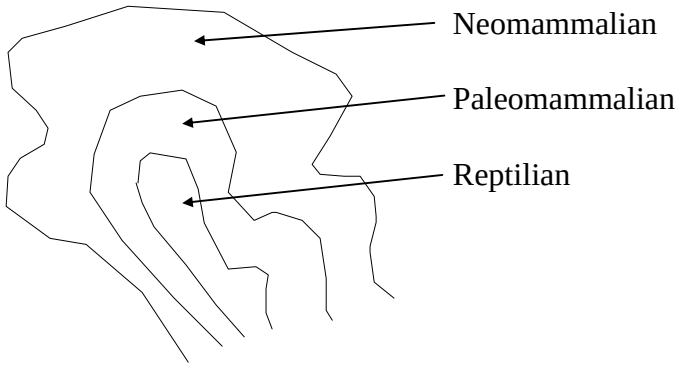
As process intensive tasks such as vision evolved, neurons clustered into small groups known as *ganglia*. Each of these little clusters is a sort of primitive sub-brain, processing tasks for a localised area. In the earthworm, there is a pair of ganglia in each segment. Indeed, the human spinal cord still shows evidence of segmentation and is capable of primitive processing.

Although the subsequent evolutionary development of the brain is not fully understood, it may be supposed that one of these ganglia at the head of the animal had to enlarge to accommodate the processing for sight, smell, etc (there are obvious reasons why the senses should be located at the eating end of the animal).

The brain shows evidence of later development being built on top of the primitive structures. Dr Paul MacLean of The Laboratory for Brain Evolution and Behaviour at the National Institute for Mental Health in the USA [23] considers that the brains of higher animals are organised as shown in figure 4.3.

Figure 4.3

MacLean's theory
of brain organisation.



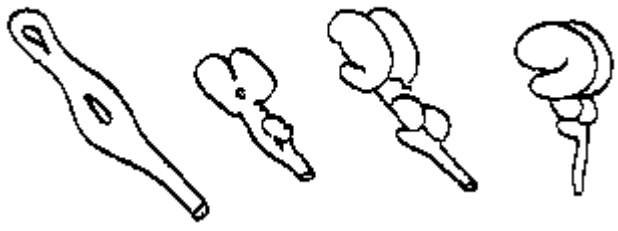
4.8.2 The Embryology of the nervous system

The nervous system is derived from a section of the ectoderm (see section 4.6). A narrow strip of this material which runs along the back of the embryo is called the *Neural plate*.

As the embryo grows, a groove forms down the middle of the plate and this deepens as development continues; eventually reaching a point where the edges meet. The neural plate has now formed a tube. Sensory fibres from the skin attach themselves to the back of this tube; fibres which will become the motor nerves attach themselves to the front. This structure will become the spinal cord [24].

The end of the tube subsequently enlarges into two swellings. These two swellings will eventually become the cerebral hemispheres [25]. The stages involved in this transformation are shown below in figure 4.4.

Figure 4.4
Stages of brain
embryology
(neural tube on
left)



After: Carpenter, from the book 'Neurophysiology'

From the proceeding descriptions we can see that Evolution and Embryology are both similar processes. These similarities will be exploited in later sections.

Chapter 5
Artificial Evolutionary Methods and their application to Neural Networks

5.1 Introduction to chapter

In this chapter, both the Genetic Algorithm and Artificial Embryology will be discussed.

These techniques are search and optimisation methods based on biological evolution. The Genetic Algorithm is based on the simulation of a large number of organisms. By selecting important traits, using a computer model of natural selection, it gradually evolves a solution which is close to optimum for the problem. The Artificial Embryology works by growing the system from a simple form to a complex form. In biological terms, this is similar to the growth of the foetus or the development of a single evolutionary line through geological time.

The chapter first outlines the Genetic Algorithm, then illustrates how it is applied to ANNs. The Artificial Embryology is outlined next. Finally, the two methods are compared and contrasted.

The development of the Embryological Algorithm shown here and its application to ANNs is a major part of the original work in this project. The concept and application of this algorithm are original.

5.2 The history of the Genetic Algorithm

Like many of the developments in computing, the earliest ideas about evolutionary algorithms originate with John von Neumann. In the 1950s and 60s he suggested that evolutionary processes could play a part in computational systems [1].

The Genetic Algorithm (GA) proper, had its origins in a 1975 monograph by John Holland, called: *Adaptation in natural and artificial systems* [2]. Holland worked at

the University of Michigan and both his students and colleagues were involved in the development of the algorithm. Holland is considered the father of the technique.

There are two other techniques which are closely related to the GA, these are: *Evolutionary Strategies* (ES) and *Evolutionary Programming* (EP) [3]. They were invented around the same time as the GA, but independently. They are not covered here as they are very similar to the GA, the main difference is that they do not use a crossover operator, relying instead on mutation to perform the search and they are not nearly as widely used.

Unlike the ANN, which had to go through many disparate variations before developing useful applications, the history of the GA is considerably simpler. Since the publication of Holland's work, many papers have been published on the subject, which have demonstrated its usefulness in spheres of optimisation, search and artificial intelligence.

5.3 The operation of the Genetic Algorithm

The basic operation of the GA is relatively simple. The method outlined below is one way of implementing it; there are several others. There are also additions to this basic algorithm which may be applied to the scheme (some of these extra operators are outlined later in the text).

5.3.1 The basic algorithm

Given below is the basic algorithm [4] (compare this with the biological equivalent outlined in appendix 6).

1. Start condition.

   The GA starts with population of random strings called Chromosomes. These strings usually consist of a binary sequence (known as the Genotype). The bits in the sequence (called the genes) contain the features of the system to which the GA is being applied. For example, in a neural net, a chromosome might represent the topology of a particular network or the weight values used in the network. Therefore one chromosome corresponds to a particular

implementation of the network. To assist in the understanding of the process an example will be included. Figure 5.1 shows four typical strings.

Figure 5.1
Typical population of GA strings.

1010010 *1*
1001001 *2*
1010100 *3*
0100100 *4*

2. Fitness testing

The system set-up which is coded in each string (called the phenotype) is tested and given a mark of merit called the Objective Function or Fitness. In the case of an ANN, if the chromosome is coding the weights in the network, then the fitness might be the corresponding error. If a chromosome were coding network topology, then the fitness might include parameters related to how well that topology worked for a particular problem. How fitness is measured depends on the system to which the GA is being applied. In this example, it is arbitrary. The phenotypes are tested and, according to their performance, assigned a mark of fitness as shown in figure 5.2

Figure 5.2
Fitness testing

String	Genotype	Fitness
1	**1010010**	34
2	**1001001**	101
3	**1010100**	10
4	**0100100**	61
Total		206

The fitness can be normalised to provide a percentage; this is shown in figure 5.3. The probability is the percentage expressed as a number between zero and one.

Figure 5.3
Assigning a percentage.

String	Genotype	Fitness	Percentage	Probability
1	**1010010**	34	16.5	0.165
2	**1001001**	101	49	0.49
3	**1010100**	10	4.85	0.0485
4	**0100100**	61	29.65	0.2965

3. Copying

Strings are now copied pseudo-randomly; that is, a string is picked with a probability as expressed in figure 5.3. This means that string 2 is the most likely to be picked, followed by string 4. Each time a string is picked it is copied into a *mating pool*. In the simple algorithm above, this is done four times. As they are picked, the strings are paired. Having completed this stage, one might end up with a mating pool as shown in figure 5.4.

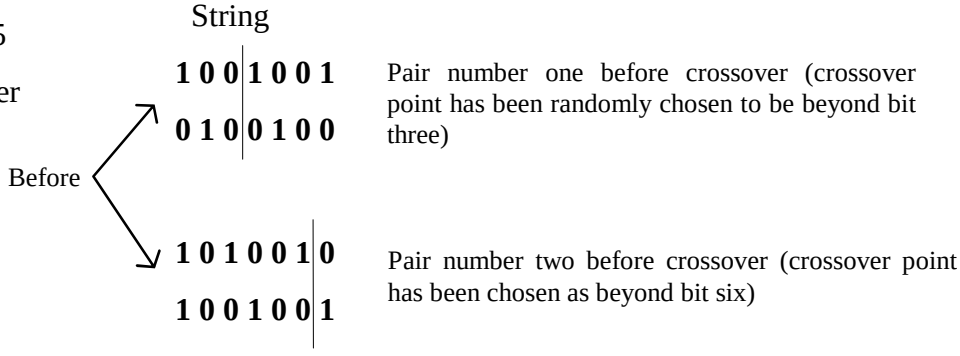
Figure 5.4
The mating pool

String	Number	
1001001	2	} Pair number 1
0100100	4	
1010010	1	} Pair number 2
1001001	2	

4. Cross Over

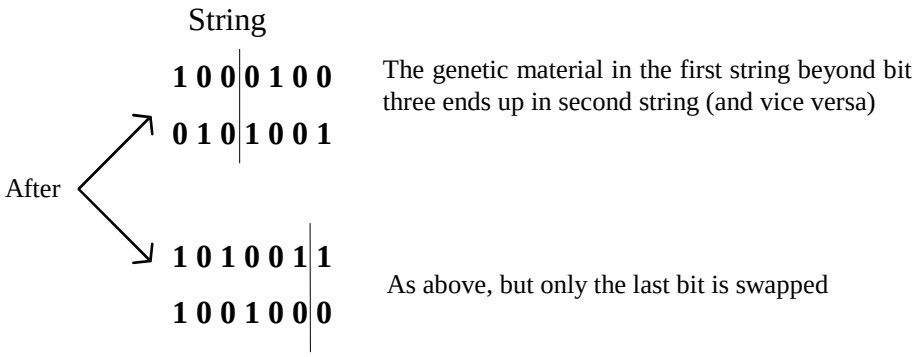
In the next stage of the algorithm, the genetic material of the strings is swapped between the pairs. The point at which the swap is made is chosen randomly. This operation is shown in figure 5.5.

Figure 5.5
Cross Over



Pair number one before crossover (crossover point has been randomly chosen to be beyond bit three)

Pair number two before crossover (crossover point has been chosen as beyond bit six)



5. Mutation

Finally, a small number of randomly chosen bits in the strings are inverted. This is a method of introducing different genetic material into the system and ensures that the whole solution space is searched.

5.3.2 Additions to the basic algorithm

Having outlined the basic algorithm, it should be noted that there are several more complex operators which may be added to it. These are meant to simulate some of the other aspects of the biological system. Some of these alternations to the basic algorithm are listed below.

1. Multiple crossover is sometimes added to basic algorithm. This allows the chromosomes to exchange genetic information in several places at once and hence spreads good genetic material more rapidly through the population [5].

2. Another popular addition to the basic algorithm is the use of continous numbers rather than binary representations. In many systems this aids parameter representation in the system [6].

There are also several other operators which can be added to the model also. These include the introduction of multiple chromosome genotypes (often extending the chromosome length), the addition of special coding to the chromosome (for example to reduce problems with the crossover operation removing good genes) and provision for the evolution of species which cannot interbreed [7].

5.4 Genetic Algorithms and Artificial Neural Networks

Researchers have developed several ways of combining the GA and ANN. GAs may used to support the operation of ANNs, for example to pre-process data. Alternatively, they may be used as part of the ANN itself, for example as a training algorithm [8][9]. It will be helpful to look at each of these applications in turn. Should further references be required, the reader is directed to review papers [8] [9].

1. GAs used in support of ANNs.

GAs may support ANNs in three broad areas:

- Prepossessing of input data. In particular, the use of the GA to select features in the input data, or to transform feature space [10].
- Using the GA to pick an appropriate training method for the ANN, or to pick training parameters such as learning rate [11].
- Using the GA to analyse the neural network [12].

2. GAs used as part of the ANN itself.

Again, GAs can be used as part of the ANN algorithm in three basic ways:

- As a training algorithm for ANNs.
- To select ANN topology (of particular interest in this thesis).
- Some algorithms are a combination of the two above.

It should also be noted that some workers have used ANNs to assist GAs [8]. It is the second of these two basic categories (GAs as part of the ANN), with which this research is chiefly concerned. This is directly relevant to the Embryological Algorithm outlined later. These applications of the GA will be outlined in more detail below.

5.4.1 Using GAs to train ANNs

Most authors consider that using a GA to train an ANN is not as efficient as the best gradient descent methods (that is, BP type methods) [8]. However, when error or gradient information is not available, GAs may hold much more promise. They also less prone to local minima problems [13].

There are several different schemes for training ANNs using GAs. In a typical system, each weight is expressed as a substring, figure 5.6.

weight 1 = **0010** substring 1
weight 2 = **1100** substring 2
weight 3 = **1001** substring 3
etc .....

Figure 5.6 Network weights expressed as substrings..

Each of these substrings is just a binary representation of the weight. The substrings are then concatenated into one long string which represents the weights of the whole network, figure 5.7.

001011001001.....etc

Figure 5.7 Concatenated string.

This string is then used in the GA as a member of the population. Each string in the GA represents the weights of a whole network.

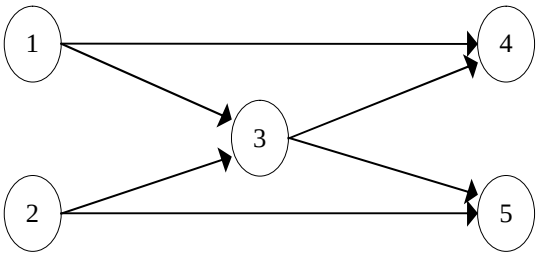
There are several variations on this theme (which may be found in the references); for example, the approach is sometimes combined with BP or statistical methods to produce a hybrid training method [15].

5.4.2 Using GAs to select ANN topologies

The use of GAs to select ANN structure has a direct relevance to the research contained in this thesis; this is the purpose of the Embryology outlined later. The GA therefore offers a benchmark both in terms of results and philosophy with which the Embryology can be compared.

Several different ways of representing network structure for use with the GA have been reported. Figure 5.8 shows a simple network.

Figure 5.8
A simple Neural
Network



First, consider the connections from neuron 1. These may be represented by the string shown below:

0 0 1 1 0

The first zero represents the fact that neuron 1 is not connected to itself. The second zero means that neuron 1 is not connected to neuron 2. The third digit, which is 1, means that neuron 1 is connected to neuron 3; and so on. The complete network may be represented by the matrix shown in figure 5.9.

Figure 5.9
Network Matrix

0 0 1 1 0
0 0 1 0 1
0 0 0 1 1
0 0 0 0 0
0 0 0 0 0

where matrix element $\mathbf{M_{jk}}$ is 0 if there is no connection between neuron **j** and **k**; if the matrix element is a 1, then there is a connection.

It is possible to concatenate the matrix in figure 5.9 into one string as shown in figure 5.10

0 0 1 1 0 0 0 1 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0

Figure 5.10 Concatenated string.

This string can then be used as part of the GAs population, each string represents the connection of a whole network.

One of the obvious disadvantages of this method is that, as the network grows, the size of the string also becomes larger. Several workers have tried to overcome this, either restricting the structure of the network to certain defined types [16], or by finding abstract representations for network structure; for example, using LISP s expressions [17] or using addresses to encode connections [18]. Other researchers restrict themselves to removing connections (network pruning) or adding / removing neurons (but maintaining complete connectivity between layers).

5.5 Incremental evolution

The Genetic Algorithm intelligently operates on a population of random strings. It therefore represents a structured search through random data space. This is in stark contrast to the evolutionary development of the biological nervous system, which produces a highly structured result. Considerations such as this led the author to consider other approaches to network topology definition. The result is the Incremental Evolution approach, also called Embryological Algorithm (EA), presented here. This algorithm is a major part of the original research in the thesis. As explained in chapter 1, the better name for the algorithm is *Incremental Evolution;* however, the term Artificial Embryology had already been coined by Holland and Snaith.

Although a subsequent literature search yielded some papers which outlined similar ideas (for further details on other papers, see chapter 6), none of the workers had developed algorithms similar in operation to these.

The basic idea behind the EA came from ‘The Blind watchmaker’, a book by Dr Richard Dawkins [19]. In his book, Dawkins explained the operation of evolution in animal populations. To aid his explanation of the evolutionary process, he developed a simple computer program which produced branching structures which he named Biomorphs. Each time the program ran, the Biomorph grew; that is, it added another

layer to its structure. These Biomorphs are a primitive form of EA. Dawkins encoded a sequence of numbers which represented the structure of the Biomorph as a string (that is, as a single string - hence the phrase ‘single string techniques’ in the title of the thesis). Figure 5.11 shows how a typical Biomorph grows.

Figure 5.11 Typical Biomorph development.

In the original program, the Biomorph was represented by nine genes. The genes influence parameters such as the height and width of the Biomorph. The Biomorphs always grew by branching into two segments (as a biological cell divides in two during reproduction) [20].

The process has several important properties:

- It represented a structured search and produced a structured result
- It allowed growth from simple to complex structures
- It was based on sound biology (although the final algorithm is NOT a direct model of nature).
- It appeared to be adaptable to ANNs

The system, as presented by Dawkins, is not in itself suitable for use with neural networks; however, it is possible to use the essence of his algorithm (that is, growth from simple to complex) as the basis of an algorithm suitable for use with ANNs (see paper 1 in appendix 1 for further details of how Dawkins algorithm came to form the early basis of this method). In its final form, the EA bears little resemblance to the Biomorph algorithm, from which it drew its inspiration.

5.6 Incremental evolution and Neural Networks

The adoption of a Biomorph-like algorithm to ANNs means that properties within the network which can alter performance must be identified. As far as topology is concerned, these parameters involve the network shape, size and configuration. In this project, they have been named *Growth Strategies*. Although network learning and neural functionality could also be included in the EA, these have been excluded from the investigation for the following reasons:

- Many successful algorithms are available for network learning and there is therefore no real need to consider this aspect, as an 'off the shelf ' algorithm such as BP may be used (for further discussion, see chapter 9) .
- In changing neural functionality one would also have to study the problems with learning algorithms which this caused; this is a project in itself. Further, it has been shown [21] that standard neural functionality can fulfil all classification tasks (for further discussion see chapter 12).

The growth strategies typically involve alterations such as: changing the number of layers in the network or changing the number of neurons in a particular layer; further details will be given in chapter 7.

The string for use with the EA is not in binary form like the GA; instead it is multi-valued, with each gene (number) holding information on one strategy. An example string is shown in figure 5.12.

a b c d

Figure 5.12. An example string for use with the Embryology.

In this case **a** is a number (gene) which might represent the number of layers in the network, **b** the number of neurons in the current layer, **c** the connectivity of the current layer and **d** the introduction of bias units into the network.

Not all strategies are required for growing every network; for example, there is no need to add feedback paths or skip layers in a pattern classifier. Usually, only four or five strategies are required for a particular network. The algorithm can, however, be run in an unsupervised mode in which all the available strategies are tried. This is useful when the nature of the problem is unknown (for further details see chapter 10).

The basic algorithm is operated by selecting one of the genes from the string and changing it, thereby producing a new network. If the performance of the new network is better then the original (if it is fitter), then the new gene is kept, otherwise it is discarded and the algorithm is run again. Note the similarity of the algorithm to the statistical methods explained in chapter 3. A measure of network fitness must also be chosen. Generally this is either: speed of learning, simplicity or lowest obtainable error. The algorithm stops when it reaches some performance target in one of the fitness measures. The basic form of the algorithm is shown below:

Select gene at random from string

Change (mutate) it by a random amount

Test network produced by string

If network is better than before *then* keep change: Start algorithm again.

If network is worse than before *then* discard change: Start algorithm again.

There are several different ways of running the EA; for example, when the first tests on this method were conducted, the algorithm was run by selecting a strategy at random and mutating it by a small random number, thereby producing a new network. In this mode the algorithm is similar to an ES or EP type approach (see section 5.2). It was found, however, that a much more effective method (in terms of time) was to prioritise the strategies in order of most effective to least effective and mutate the most effective first. It was also found, that it is more effective to change the gene by

only one unit in either direction and that using different priorities of strategy yielded different topologies, which were capable of producing the same result (this is related to the 'local minima' problem and may be solved by combining the algorithm with a Boltzman type approach). These observations are further discussed in chapter 8. This improved version of the algorithm is shown below:

Pick strategy which effects network performance most (highest priority strategy)

Decrement gene effecting that strategy (this makes network more simple, thus ensuring that the network always tries to become simple before it tries to become complex)

Test network produced by string

If performance has improved *then* keep change: Start algorithm again

If performance has decreased *then* discard change

Increment gene affecting the strategy (this makes the network more complex)

Repeat test procedure above

If this strategy has not affected the performance of the network *then* start algorithm again, this time with next strategy in priority list

In the most successful algorithm used here, the string is encoded as described above. It is also possible to encode the network as an abstract form in the string. For example, one gene could represent the 'flatness of the network' and another its 'largeness'. This is similar to the way the original Biomorphs were encoded in Dawkins program. This method has in general been avoided, as it means finding a mapping between the 'largeness' factor in the string and the *actual* size of the network, whereas using a direct coding avoids this problem.

The issues outlined above will be discussed in more detail in chapters 7 and 8 which contain results.

5.7 Comparison between GAs and EAs

The basic operation of the Genetic Algorithm and the Embryology differ a great deal. The GA is an analogue of the natural situation of having a large breeding population of animals. The EA, on the other hand, is an analogue of the evolution of a single fossil line through time.

Although, at first, there may seem to be many philosophical differences between these two approaches, they are in fact more closely related than is apparent. In the fossil record, simple, single celled forms appear first and these follow a development from simple to more complex forms. This development, from simple, single cells to complex organisms is also exactly what happens in the development of the foetus. The two processes share more than just analogy. The foetal development of higher animals (such as man) pass through stages of having gill slits and a tail; these primitive features disappear in later stages of development. The Embryology is, therefore, like the Genetic Algorithm, an Evolutionary or Genetic technique.

Since the GA represents a ‘snap shot’ of a population of animals evolving slowly over time and the EA represents the Evolution of just a single species through a large time period, the two can also be considered complementary in some systems, and used together.

The major advantage of the Embryology is that the network grows from a single neuron (or other, simple form) to a complex system. As each stage is reached, the network is tested for fitness and the algorithm terminated when a suitable level is achieved. The method can, therefore, be programmed to settle on the simplest form which can fulfil the task. This is in contrast with the GA which always works with large numbers of complex networks. The GA may find, therefore, the best network but not always the simplest (the EA may also be programmed to find the best network, rather than the simplest).

Speed is another advantage of the Embryological approach. Since it only uses a single string to encode the information, it only has to assess one network per cycle, as opposed to many, in the case of the GA. It is therefore an efficient search strategy. This efficiency can be further enhanced by structuring the algorithm, achieved (as mentioned earlier) by grading the effectiveness of the growth strategies and always trying the most effective first. Such techniques further enhance the efficiency of a already structured approach. Again, this is in contrast to the randomness of the Genetic Algorithm approach.

The EA approach is ideal for growing networks in Field Programmable Gate Arrays (FPGAs). Each network can take up minimal space on the chip and several can be grown at once. The GA is not well – suited to this appraoch.

In later chapters, results will be presented to substantiate the observations outlined in the sections above.

Chapter 6

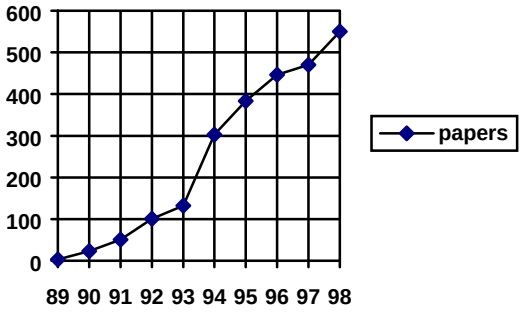
Review of previous work on ANNs which grow

6.1 Introduction to chapter

The aim of this chapter is to place the research contained within the thesis into a wider context. The chapter reviews papers which contain ideas or research on networks which grow, that is, have an embryological component.

The scale of the search which has produced these results should not be under-estimated. Figure 6.1 shows how the number of papers containing the search terms: *(genetic or evol*) and neural* has grown since 1989, in the INSPEC database (the 1998 figure is an estimate, based on half yearly results).

Figure 6.1 Neural Network papers with Genetic or Evolutionary terms.



The graph shows that this field of study has experienced high rates of growth in the 1990s. The search and review presented here is the result of reading approximately 2000 abstracts and 210 papers, as well as related material, over the course of the project.

The next sections will outline sources and search techniques. They then describe the work contained in the most important papers, in the opinion of the author. Finally, a summary is included.

6.2 Sources and search techniques

There are now several media holding information on publications. Printed sources are the most important source before 1990; these must be searched laboriously by hand. After 1990, material becomes available on electronic media such as CD - ROM and on-line database; these may be searched with keywords. Fortunately, interest in Evolutionary Techniques - applied to ANNs - started at around this time; so searching for this subject is relatively easy. Listed below are the sources used:

Paper abstracts, citations and indexes:

- Index to theses
- Pre 1989 INSPEC
- Electrical and Electronics abstracts
- Engineering index
- Applied sciences and technology index

Some interesting references were picked up from these printed sources. However, they were not as useful as the electronic searches.

Sources on CD-ROM:

- INSPEC
- Compendex Plus
- Science Citation index

INSPEC is the most useful of all the primary sources. It contains abstracts from all the important journals in the field and is available from 1989 on CD - ROM.

Internet resources:

- NEuro-Net
- GA- Net
- IEE
- Edinburgh University AI dept
- WWW searches

These Internet sources turned up some references, but generally proved of little value. The Edinburgh University AI Department is an exception to the rule and is useful (being more extensive and having good links and on-line papers). It also leads to other interesting Internet links.

Other useful resources:

- Review papers
- Reference trails

These two sources are extremely important and useful. They have about equal value to INSPEC as a source of publications. Reference trails involve finding papers from references in other publications. Review papers are also a vital source of information.

Listed below are the search words and phrases used for searching electronic databases. The abstracts of all papers turned up under these searches were read. The ***** symbol is a wild-card; that is, a character which can be used to find several word-endings (for example, *grow** will find *grow, growth, grows, growing* and *grown*). These terms were used in all searches through electronic publications databases.

Search terms used in electronic searches:

- *(genetic or evol*) and neural*
- *embry* and neural*
- *grow* and neural*
- *Biomorph*
- *Dawkins*

The result of this comprehensive search was the following:

1.	Papers mentioning Dawkins Biomorphs and Neural Networks	2
2.	Papers outlining Embryologies for Neural Networks	6
3.	Papers outlining Neural Networks which could grow	81

In the sections below, these papers and their importance is reviewed.

6.3 Review of important papers, authors and techniques

In the section above, 89 papers of some relevance to the project were found. Many of these contain ideas or schemes similar to the ART network (section 3.7.2). These schemes involve adding or removing units, connections or layers to a network, the object being to improve network performance in terms of training time or learning ability. Such papers are not aiming to provide a basis for truly intelligent systems but rather to improve ANNs for applications such as pattern recognition.

Another philosophy is also encountered and that aims to grow networks into brain-like structures capable of solving difficult real-world problems. These systems are generally more powerful, usually have a strong biological basis and often the purpose of the work is to simulate part of the nervous system. They are usually a random algorithm able to add neurons in a incremental, but unstructured way. The work presented in this thesis advocates a middle route, not blindly following biology (since this can produce a high degree of redundancy, and an unstructured network, which is generally impossible to understand) but neither is it one of the simple schemes, which are only able to grow in very restricted ways. The next sections consider these papers in detail.

6.3.1 Combinations of GAs and ANNs

Before proceeding to networks which grow, a quick survey of the other dynamic topology changing algorithms is useful. This is included so that the researcher can use these results to generate benchmarks against which other algorithms may be judged (later, in the results section, a GA will be used as a benchmark) and also as a

indication of the state of the art where combinations of GAs and ANNs are concerned. However, since this is not the main thrust of the chapter, references will be confined to review papers and a small number of illustrative applications.

For a comprehensive and easy to understand overview of ANNs and GAs see the excellent (but now somewhat dated), review paper, 'Combinations of Genetic Algorithms and Neural Networks: A survey of the state of the art' by Schaffer et al [1]. Several other review papers are also worth reading [2] [3]. A typical application of GAs to optimise a large feedforward network is covered by Braun [4]. Another, less typical, example is the evolution of a control network for the classic pole balancing problem (a network designed to balance a pole upright on a wheeled cart) by Wieland [5].

6.3.2 ART and GAL: Growth based on competitive networks.

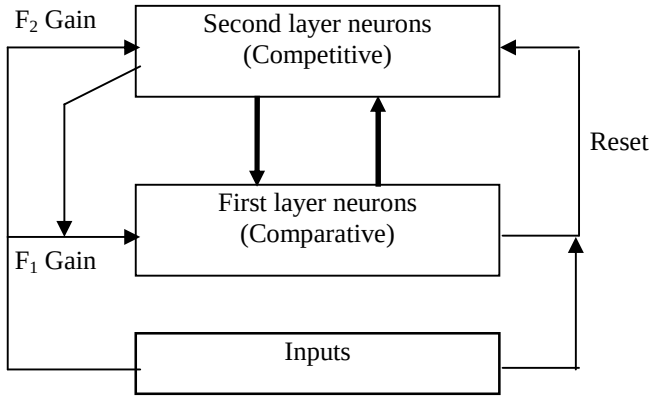
Adaptive Resonance Theory or ART (see section 3.7.2) is a radical departure from the simple neural networks which have been considered thus far. Among other attributes, it features a network which grows and was one of the first *practical* ANNs which did so. However, this is not the only feature of ART which is of interest in evolutionary ANNs.

ART aims to solve the so called *Plasticity-Stability* dilemma [6]. This occurs because retraining a network often erases its existing capabilities. For example, if we present a fully trained network with a new training pair, it often disrupts the network so badly that the network stops working completely. Obviously, this is not what happens in the BNN where patterns can be learnt at any time.

ART is an invention of Grossberg and Carpenter [7][8]; its theory and capabilities were later extended by them [9]. Since then, it has been used in variety of different forms, some bearing little resemblance to the original, for solving several different problems. Typical examples include a high speed learning network designed for pattern recognition, reported by Palmer Brown [10] and the combination of ART and Fuzzy Logic applied to control networks in a series of papers by Lin [11][12].

Simple ART , known as ART -1, shown in figure 6.2 works in the following way:

Figure 6.2 ART -1.



An input to the network is applied to a layer of competitive neurons (see section 3.7.1). Whichever one of these neurons has a weight vector which best matches the input (the similarity calculated by working out the dot product of the input and weight) is forced to have an output of one. The others are set to have an output of zero.

Another layer of neurons (the comparative layer), compares the input vector with the weight vectors of the neuron which fires in the competitive layer. If the difference between the input and weight vector is small (within a tolerance called the vigilance), the network trains by altering the weights so that they are an even closer match to the input vector. If the input does not match the weights of the competitive layer within the vigilance, the other neurons are again searched to find if any of them are a better match. If not, then a new neuron is allocated (this is how ART grows). All this is mediated by three control signals: two *gain* signals and a *reset* signal. The detailed operation is complex but well covered in the literature [13][14].

It is reasonably obvious why ART solves the *Plasticity-Stability* problem. It will not try and train any neuron if the input is too different from the present state of the neuron; it will simply allocate a new neuron.

Another competitive type network is Grow and Learn (GAL). This was first proposed in 1988 by Alpaydin [15] (this 1995 paper is also an excellent review paper of ANNs which grow), based on ideas from Baum [16]. Alpaydin's explanation has then been built on in a number of papers. He and other workers have used the network (or modifications of it) in applications including mobile robot control [17] and recognition of Phonemes [18]. The GAL network incrementally grows by adding units as it learns, but can also contract by losing units which are no longer used, during a 'sleep' mode. GAL builds on work by several other researchers (references listed in [19]). Interest in GAL has been limited compared with ART.

While discussing competitive type systems, it is worth noting that several researchers have applied incremental growth to Kohonen maps. These include Fritzke [20], Datta (who refers back to Sabourrn [21]), Parui [22], Lee [23], Weng et al [24] and Lee and Peterson [25]. These systems grow through the addition of units when learning requires it, in a similar manner to those described earlier.

6.3.3 Papers on general ANNs which grow

Although competitive networks such as ART are interesting because they can solve the fundamental growth - training problem and they also represent an early attempt at network growth; of more interest to this project are growing networks using the McCulloch Pitts model (see section 3.2.1 and 3.3). These systems provide two interesting pieces of information.

a. Theory and models of network growth which can be used to define growth strategies.

b. A survey of other work with which to compare this project.

Surveying the papers found during the literature search, the growth methods found fall into three broad categories:

a. Networks which grow by adding layers.

b. Networks which grow by adding neurons

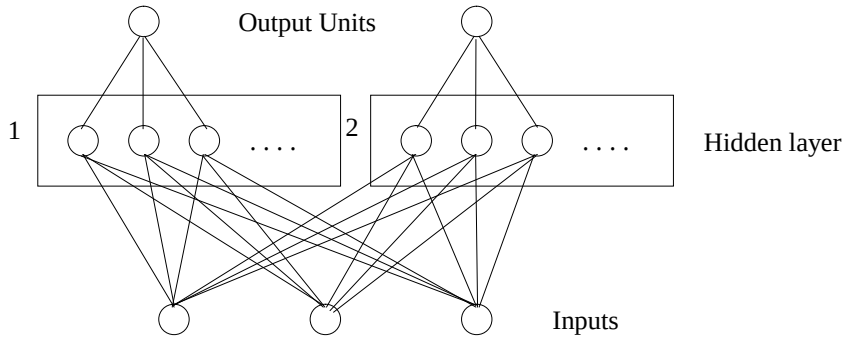
c. Networks which grow by adding links

Strategies **a** and **c** are always combined with the addition of neurons, so that is the logical area to start the survey.

Several of the papers in the area refer back to a 1989 paper by Ash [26]. Ash outlines a network, with one hidden layer, which grows by adding another neuron to that layer when necessary. Other work in a similar vein can be found in references [27][28].

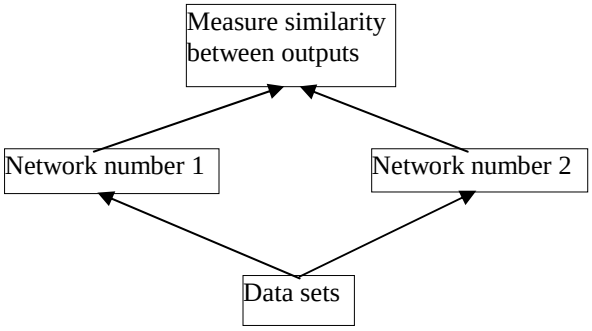
Chakraborty et al [29] takes this idea a step further The network structure used is shown in figure 6.3.

Figure 6.3 Network Structure



The network grows by adding units to its hidden layer. This is accomplished by starting with two networks and comparing them, see figure 6.4. The algorithm assumes that at the border of network size, as the networks start to over-fit, they will diverge. At the point of divergence, the algorithm stops and amalgamates both networks. This strategy is in effect the opposite of Occam's Razor (simplest is best).

Figure 6.4 System components.



Vinod [30], in his 1996 paper, outlines an algorithm which grows one neuron at a time. The unique aspect of his approach is that the neuron is not added in an arbitrary position but in the position calculated to give the maximum reduction of error. Only those links necessary to do this need be included and so the final algorithm produces a non-uniform, sparse network.

Ferran and Perazzo [31] investigate different sizes of networks and their capabilities by adding layers and neurons to the network structure. Their paper provides a extensive account of the performance of different network architectures. Although they do not demonstrate an actual growth algorithm, the paper presents many ideas on the subject. Other, similar algorithms are given in the references [32][33][34].

Obradovic and Srikumar [35] add hidden units to their network but use a GA to partition representation.

Turning now to the reduction of connectivity of the network, an interesting and important paper on the subject is by Kozma [36]. In this paper, he considers a network used for Nuclear Plant Monitoring. The network uses a BP learning algorithm. His algorithm allows weights, which are not being re-enforced through BP weight changes, to decay to zero. This produces a skeleton network. Other papers in a similar vein are included in the references [37][38][39].

At this point, attention is drawn to a small number of papers which concentrate on the growth of Hopfield or Recurrent networks. Of these papers, one very important contribution comes from Anderle [40]. The importance of his contribution is that he considers networks which are inherently stable and grow in such a way that stability remains assured. Anderle's method starts with an unconnected network and grows connections, one by one, until the desired result is achieved. Other work, worth mentioning on Hopfield nets is Brouwer [41], who grows his net by adding neurons, and Sriram and Kang [42].

6.3.4 Biologically inspired systems and tree like networks.

Several researchers are looking into networks which are inspired more directly by biological growth. Many such networks have a different objective from those illustrated thus far. They are generally grown for research into biological systems or systems with emergent behaviour. These networks often display a high degree of asymmetry and they usually form tree like structures [43], rather than the more familiar fully connected networks.

A good example of this type of network is that of Nolfi et al [44]. Here a network is grown as a robot controller.This particular system uses a GA; other systems, however, do not. Several systems 'seed' the network with starting points [45] and allow the topology to grow from these. Other networks form a random or semi-random structure [46] [47] [48].

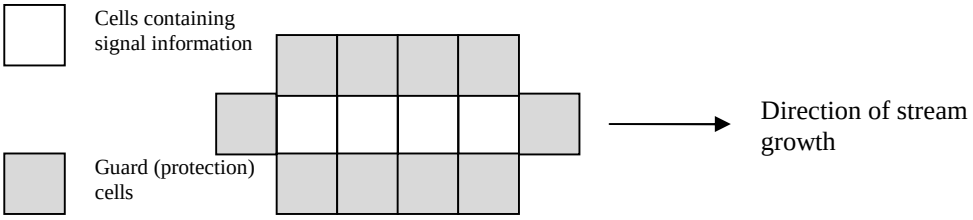
Such systems also yield to implementation using Kohonen maps [49][50][51]. Tree-like structures, are also evident in attempts to combine decision trees with neural networks [52][53]. Some of these systems have also been adapted to grow [54].

6.3.5 Hugo de Garis

One of the most important workers in the field of growing networks is Hugo de Garis. His ideas are often very original and sometimes come under fire from the ANN research community for this reason. The object of his research is quite simple; he wants to build a brain.

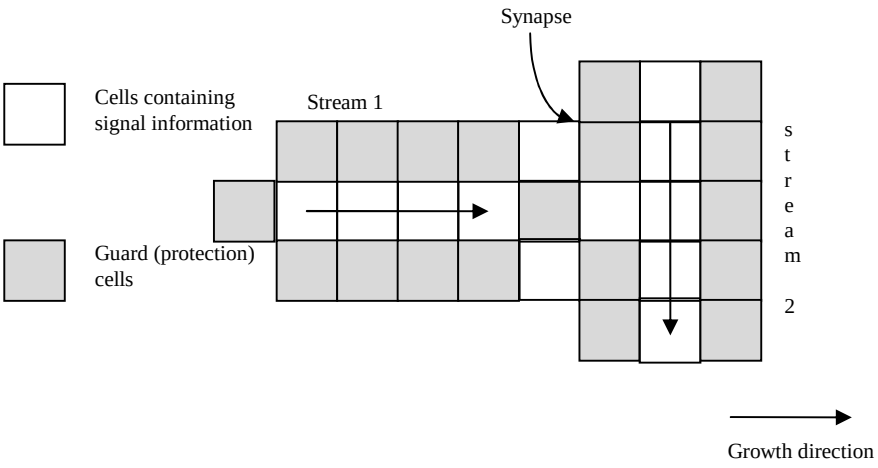
The networks which de Garis uses, are based on cellular automata [55]. The networks grow in long linear streams, figure 6.5.

Figure 6.5 Network grows in linear ‘streams’.



The streams can continue to grow in a straight line, turn right or left, split, branch, etc. What they do is controlled by 'signals', which proceed down the middle of the stream. The sequence of signals is defined by a GA (which therefore defines the growth pattern). When one stream hits another, it forms a 'synapse' which transmits information, figure 6.6.

Figure 6.6 ‘Synapse’ between two streams.



Once the network has grown, its fitness is tested on an application. The information gained is fed back into the GA and the whole process repeats. Through this process, useful networks are formed. The main criticism of de Garis’s work, is the inherent complexity of the system and the amount of redundancy in the network. However, this complexity adds dynamics to the system, in a similar way to Thompson’s definition of Digital Circuits using GAs (see section 9.3.1).

De Garis is a prolific author. Possibly his most accessible publication is his 1993 paper, Neurite Networks [56]. There are too many other papers to note individually (he produces numerous updates and speculative papers); a selection are included in the references [57]. His work is interesting in that it is an example of an alternative approach to that put forward here (and also an alternative definition of ‘growing’).

6.3.6 Holland and Snaith

When the project was first conceived, a search was initiated to find other workers who had considered using the Dawkin’s Biomorph in Neural Networks. Two papers came to light, both by Holland and Snaith [58][59].

Holland and Snaith point out that the Biomorph is suitable for specifying connections of a neural network. They claim that there are Biomorph attributes which are useful in neural networks; these include symmetry and segmentation. An interesting point which they make, is that segmentation and related behaviour may make it possible to define a modular network.

Having made these points, the papers do not suggest how the detailed implementation of their system could work. Examples are not given of how a network might be encoded. The work seems to suggest using the Biomorph in its original form. This is a different approach to that outlined here, since this approach uses the Biomorph *idea* but not its structure.

More critically, Holland and Snaith do not consider *growing* the network. This is an integral part of this work. These workers coin the term 'Constrained Embryology' to describe the method.

6.3.7 Attempts to grow modular networks

Growing neural nets is all very well. However, the brain is organised in a modular hierarchy (see section 2.4), with many functions being localised. Some authors have tackled the problem of how a modular system might be evolved; however, none (as yet) GROW a modular network.

A GA solution is proposed by Happel and Murre [60]. The design algorithm, known as CALM, is biologically inspired. A simple case of hierarchical networks for a practical system is demonstrated by Ceravolo et al [61]. The modular aspect of networks will be further discussed in chapter 10.

6.4 Summary and comments on originality of this research

Research in this area is split into three categories. Firstly, the type of work illustrated in sections 6.3.2 and 6.3.3. This involves growing or incrementally changing one aspect of the network: layers (rarely), connections (occasionally) or number of neurons (usually). The aim of this is always to produce a network which is optimised for a given task and, in particular, to apply Occam's Razor to the network (to produce the simplest network which will satisfy a task).

The second type of research relates to more biologically inspired networks of the type discussed in section 6.3.4. These are often conceived as illustrations of biological growth (in some ways they resemble neuron growth in the foetus). They are more general than the previous examples, but are still usually feedforward networks. They are mostly not homogeneous.

Finally, we come to the ambitious attempts, like that of de Garis, to build intelligent systems with complex networks.

The work contained in this thesis falls into none of the above categories. It grows in a similar way to the networks sections 6.3.2 and 6.3.3, but the network can grow in much more complex ways. It resembles neither the tree like structures of the networks in section 6.3.4 nor de Garis's Cellular Automata.

In conclusion, the literature search has shown, that at the time of the publication of our first paper outlining this technique in April 1997 [62], the idea was completely original. The important points of differentiation from other work are:

- The use of an ordered growth structure (the strategies are prioritised) and a comprehensive list of growth strategies (the networks above use one or at most two strategies).
- The coding of the network as a string, allowing much more flexibility in how the algorithm can be run.

Chapter 7
Growth strategies for Artificial Neural Networks

7.1 Introduction to chapter

This chapter and the next (Chapter 8, Results obtained from application of Growth Strategies), contain the basic experimental results obtained during this project.

In the sections which follow, the growth strategies which may be used to grow Artificial Neural Networks in conjunction with the Embryological Algorithms illustrated in sections 5.5, 5.6 and 5.7 are outlined. Each of these strategies is described and the results obtained are presented. The results were obtained by experiment using the methods outlined below. Some results are cross referenced to literature and these are indicated in the text.

7.2 Methods used to obtain results

Several different methods were used to generate the ANNs used in both this chapter and the following. In each case, the technique most appropriate and easiest to implement was employed. To save space in the main text, these methods are given a detailed description in appendix 2. They were:

- Direct coding in the C and C++ programming languages.
- Direct coding in the BASIC programming language (mainly for prototyping).
- MATLAB™ mathematical modelling software.
- The MATLAB™ Neural Networks toolbox.

The networks were trained using Backpropagation unless topology made this difficult, in which case the training was by Cauchy Algorithm or GA learning. BP with momentum was also used in some of the more complex networks to help them overcome any local minima problems. Symmetrical feedback networks were trained using Hopfield learning complying with the Grossberg Stability Criterion or Recurrent

BP [1]. A new generalised learning algorithm was also developed using Taguchi methods [2] (see appendix 1 and chapter 9 for details). More detail on training methods is given in chapter 9. The networks were tested using test vectors from the MATLAB Neural Networks Toolbox or specifically constructed data sets. These are indicated, where appropriate, in the text and are outlined in detail in appendix 4.

It should be noted when viewing the results below and in chapter 8, that due to the pseudo-random nature of training networks (for example, the initialisation of weights and weight changes using statistical methods), slightly different results are obtained if the network is run with different initial conditions. To overcome this, all the results are obtained from four trial runs with differing initial weights. Where appropriate, the maximum standard deviation (σ) of the four final results is specified.

The example graphs show training epoch on the x axis and sum squared error on the y axis (unless this is not appropriate for a particular strategy). Training epoch refers to the number of training cycles (each member of the training set presented once) the network has gone through. For example, in the character recognition problem one epoch refers to each of the 26 letters having been presented once. Sum Squared Error (SSE) is the sum of the squared differences between Outputs and Targets for a particular input [3] as shown in equation 7.1 below.

$$S.S.E = \sum_{n=1}^{x} (out_n - \text{target}_n)^2 \qquad \text{(eqn 7.1)}$$

where x is the number of neurons in the output layer. This measure has been chosen to keep results consistent as it is also used in the *MATLAB* neural networks toolbox [4]. It can, of course, give different answers for different sizes and types of network. In particular, the number of output neurons effects the final figure and this should the borne in mind while examining the results.

The benchmark used to assess the system is also important when considering results. There are three basic benchmarks which may be used:

a. The network stops growing when it reaches the topology which trains fastest.

b. The growth stops at the simplest network which can do the job [5].

c. The growth stops at the network which displays the most noise tolerance.

Examples of results from different benchmarks are given where appropriate in the text (see particularly section 7.3.2) and in chapter 8 (section 8.6).

7.3 Growth Strategies

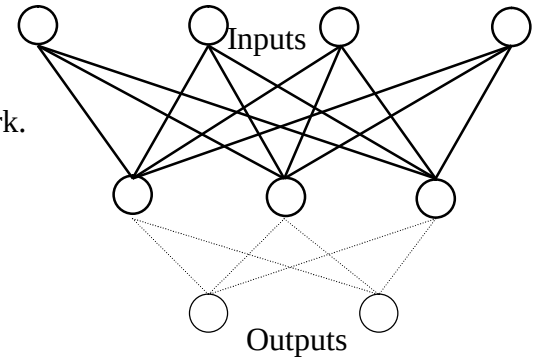
To implement an algorithm in which the neural network grows, such as those detailed in chapter 5, it must be decided how the network is to evolve. To this end, a review of previous work was initiated, the object of which was to find a comprehensive list of strategies for network growth. Details of strategies are distributed throughout the literature and these items are referenced where appropriate in the text. In the sections below are a list of the growth strategies.

A more succinct version of this list may be found in appendix 1, papers 2 and 3. For each of these growth strategies, an algorithm has been devised which allows the network to grow in this way. These are included in appendix 3. The algorithms are used to grow the networks in chapter 8. The list below is exhaustive; in practical systems only the most appropriate strategies are used. Some of the listed strategies are therefore of somewhat limited application.

Figure 7.1 shows a standard, fully connected, two layer network. This diagram is included to provide a point of reference when the growth strategies are described. Some of the strategies are applied to a three layer network; the additional layer is shown by the dotted lines.

Figure 7.1

A fully connected network.



Results obtained by direct experiment as part of the project are given the title *Example*. The type of network used is noted by *Network* followed by numbers denoting the number of neurons in each layer. For example, *Network FF 3 5 2* is a three layer feedforward network with 3 neurons in the first functional layer, 5 neurons in the second layer and 2 neurons in the third (output) layer (therefore the network shown in figure 7.1 is a *FF 3 2*). It is not always possible to denote this at each stage since some of the networks are growing, in which case, the start and end networks are given. *Training* describes by which method the network was trained. *Test data* refers to the data on which the network was trained and tested (refer to appendix 4).

The aim of the chapter is to illustrate each growth strategy in turn, not to give an absolutely exhaustive description (which would break up the continuity of the text). Further supplementary results are included in appendix 5.

7.3.1 Changing the number of layers in the network

Of all the strategies described in this chapter, changing the number of layers in a network [6] and adding feedback paths have the most effect. This is because they change, in a fundamental way, the mode of operation of the network.

Section 3.4.2 illustrated the XOR problem and the meaning of linear separability. It was shown that the XOR problem could be overcome using a multilayer network and that a three layer network was theoretically capable of separating the classes of any problem [7]. To fully appreciate the effect of increasing the number of layers within a network, it is necessary to revisit linear separability and its relationship to network size, in more depth [8]. To simplify matters, the neurons in the discussion below have

hard limiting (binary) outputs. Each neuron produces a separator which may be plotted as a straight line in input space (see section 3.4.2 and reference [8]); however, the results obtained are easily extended to other non-linear activation functions such as the sigmoid function [9].

Consider a single neuron with two inputs. As already noted, this separates 2 dimensional (referred to as 2D) data space into two regions. This may be further generalised to a three input neuron which separates 3D data space using a plane, Figure 7.2.

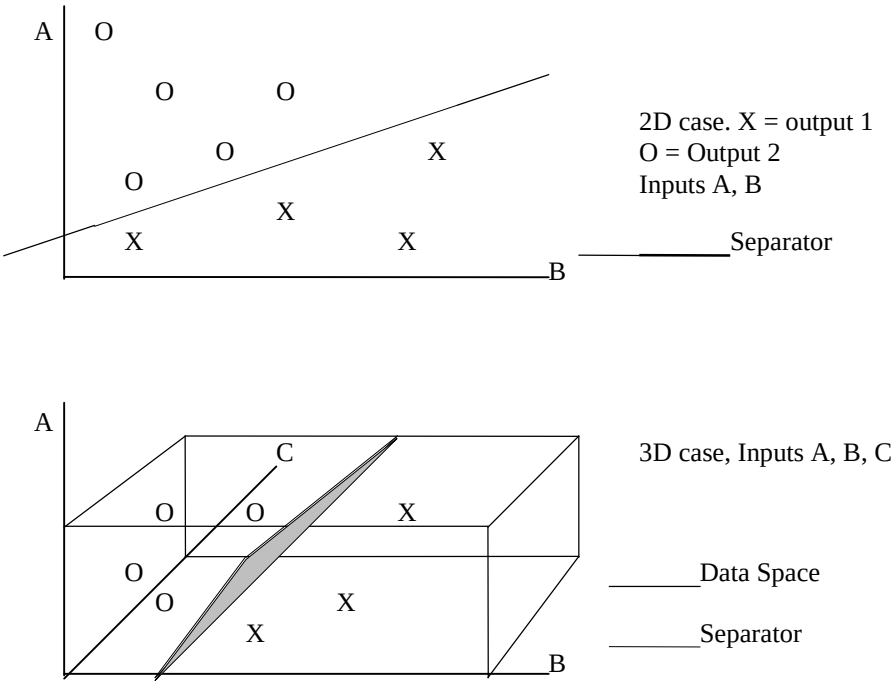


Figure 7.2
Linear separators in data space

In general, the data space of an n input neuron can be represented by a mapping contained within a n dimensional hypercube and by a separator consisting of an n - 1 dimensional hyperplane.

As discussed in section 3.4.2, a simple line or plane is not enough to separate the XOR function. To do this we need a two layer network, this allows two areas of space to be separated, figure 7.3.

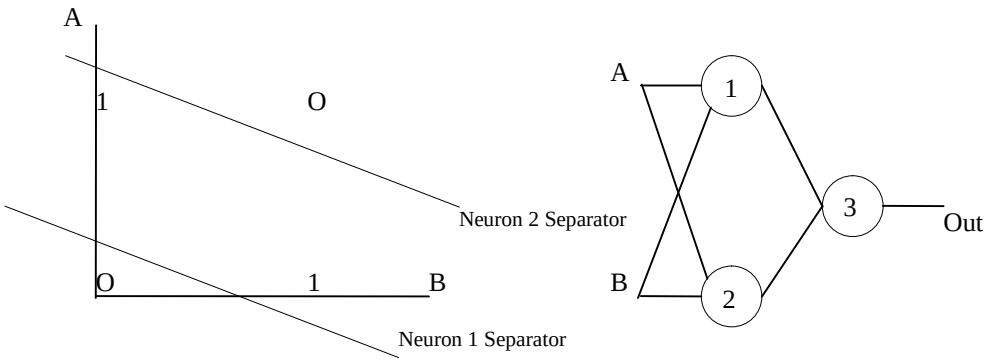


Figure 7.3 Solution to the XOR problem

The third neuron in the system (the second layer neuron), is acting like a logical *AND* gate, combining the two outputs from the previous layer. The number of neurons in the first layer controls the number of separating lines and the second layer neurons combine their outputs. The generalised system is shown in figure 7.4.

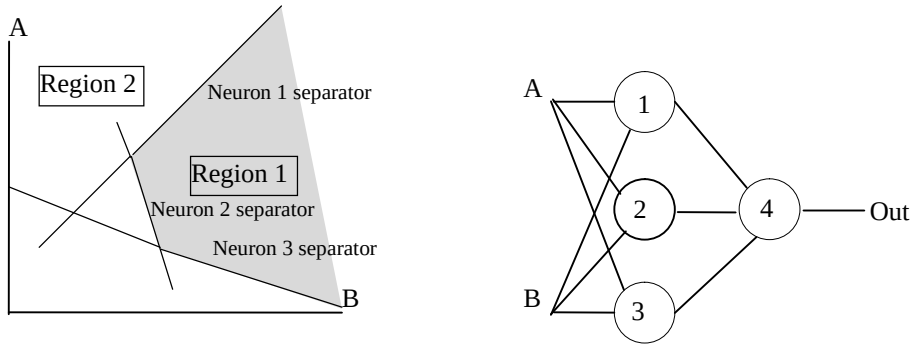


Figure 7.4 General two layer network

In the above example, a three input system would be separated by 3D regions defined using intersecting planes and so on.

In the two layer network described, notice that the separators generally do not enclose data space. They only separate regions (although the above diagrams do not show it, regions one and two extend to infinity).

The three layer network is the most general form. The neurons of the third layer act rather like logic elements, combining the unbounded areas produced by second layer, figure 7.5.

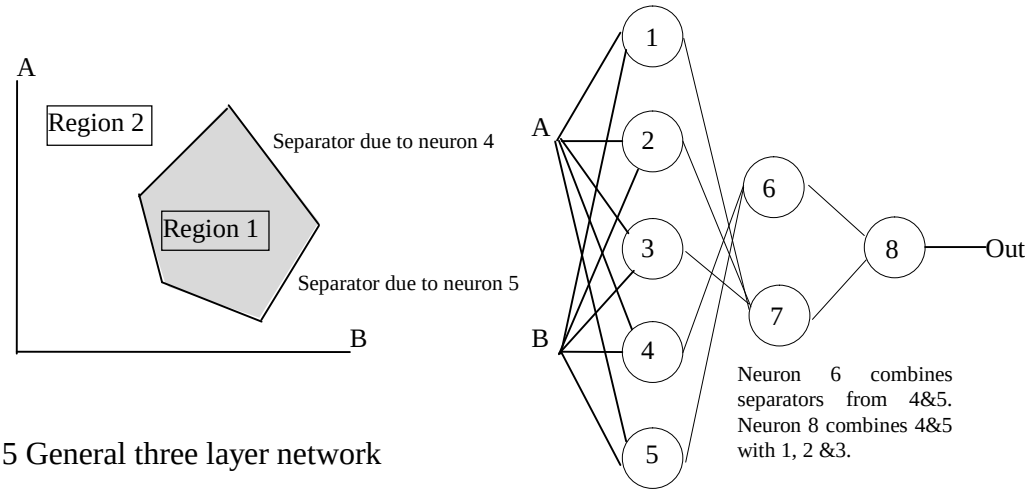


Figure 7.5 General three layer network

By adding a second neuron in layer three, we can bound a second closed region of space. Therefore, a three layer network can (theoretically) bound any number of closed areas and is the largest network (in terms of layers) required. In the worst case, where all points are unclustered in data space, the network would require as many output neurons as there are data points to be separated [8]. A four layer (and higher) network is of course possible and would produce new bounded areas given by the intersection of bounded regions produced by layer three neurons.

In cases where data points are totally unclustered in data space, the network would probably not learn to separate all points due to practical difficulties with the learning algorithm [10] (and in any case, the network would not be able to generalise [9]), in such systems, data may be mapped into another input space [11].

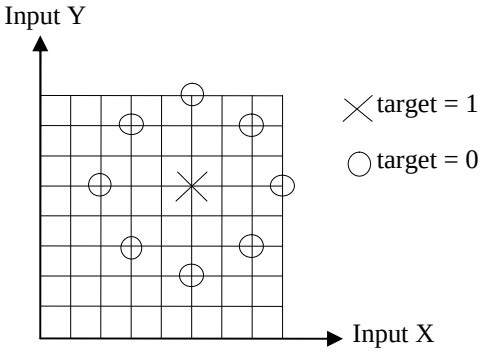
Apart from the argument outlined above for not increasing the number of layers within a network above three, there are other reasons to keep the number of layers in the network down:

a. The BNN uses very few layers in its construction (a maximum of six in the cerebral cortex and only two in motor systems) [12].

b. Adding a new layer has a major detrimental effect on network performance. This is due to the addition of more complexity to both the forward pass (because of the need to calculate the outputs from all neurons in the previous layer, before proceeding onto the next), and the reverse pass in error propagation algorithms, (see section 3.5.2).

Adding a new layer, therefore, only makes sense when a separability limit has been reached (although some authors [13] have argued that a network of more than three layers is required). This is typically apparent from the training profile of the system. It is important that the growth algorithm can distinguish between a true linear separability problem and other problems (see section 3.5.2), which can cause similar effects. As a result, this strategy is best applied after other methods of reducing error have failed.

To illustrate the importance of separability as a fundamental constraint, consider the problem shown in figure 7.6.



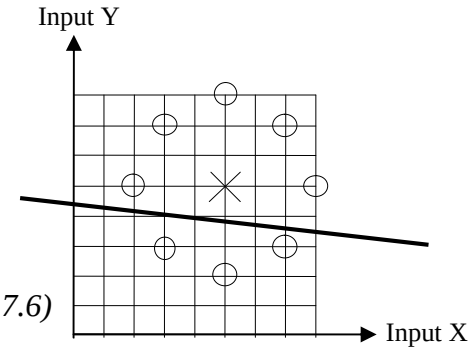
X	Y	Out
5	2	0
3	3	0
2	5	0
3	7	0
5	8	0
7	7	0
8	5	0
7	3	0
5	5	1

Figure 7.6. Problem of linear separability

Example 7.1 is the result of a one layer network trained on the data, the data separator is shown is also shown as a thick line.

Example 7.1 Single layer network trained on the data shown in figure 7.6 (outputs have been normalised to 0 or 1).

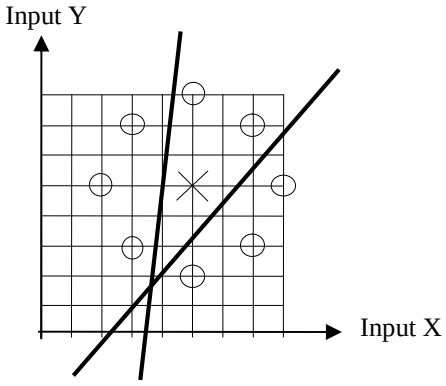
Input	Output		
5 2	0		
3 3	0		
2 5	1	*Network:*	*FF 1*
3 7	1	*Training:*	*BP (10 000 epochs)*
5 8	1		
7 7	1	*Test data:*	*Linear sep (see figure 7.6)*
8 5	1		
7 3	0		
5 5	1		



Although this is not the best theoretical result which could be achieved (probably due to local minima problems), it can clearly be seen as a line separating the lower part of the data set from the upper. Example 7.2 shows the results from a 2 layer network.

Example 7.2 Result of a two layer network trained on the data shown in figure 7.6

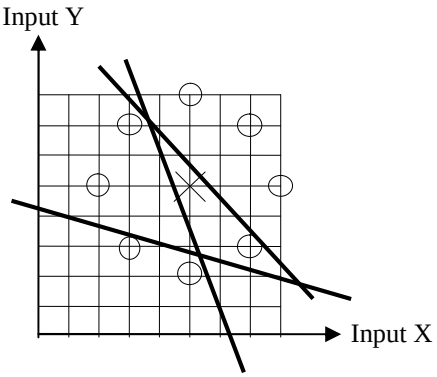
Input	Output		
5 2	0		
3 3	0		
2 5	0		
3 7	0	*Network:*	*FF 2 1*
5 8	1	*Training:*	*BP (10 000 epochs)*
7 7	1		
8 5	0	*Test data:*	*Linear sep*
7 3	0		
5 5	1		



Again, although this is not the best possible result, it illustrates data space division by two separators. Finally Example 7.3 shows a 3 layer network.

Example 7.3 Three layer network trained on data shown in figure 7.6

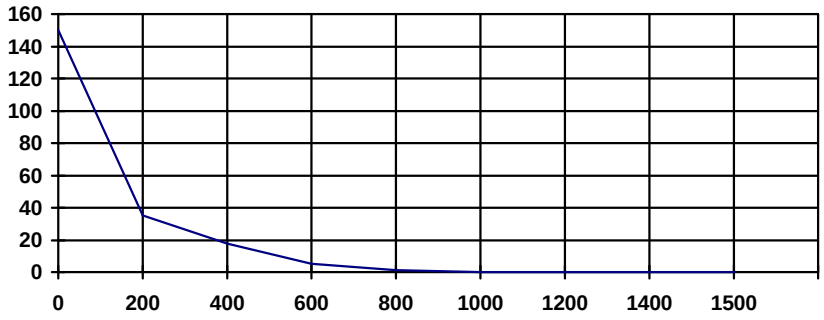
Input	Output		
5 2	0		
3 3	0		
2 5	0		
3 7	0	*Network:*	*FF 2 2 1*
5 8	0	*Training:*	*BP (10 000 epochs)*
7 7	1		
8 5	0	*Test data:*	*Linear sep*
7 3	0		
5 5	1		



It may be seen that only the three layer network is capable of separating the data (almost). If, having reached three layers, the network will not achieve a low enough error, then another solution such as input transformation may have to be used.

Adding another layer onto a network (without separability constraints) can lower the final achievable error (particularly in the presence of noise) or reduce the number of training cycles required; this is shown in Examples 7.4 and 7.5. However, almost always the penalty is a considerable increase in training time if the network is learning using Back Propagation or a related algorithm.

Example 7.4 Problem with no hidden layer.



x axis = number of training epochs.
y axis = SSE

σ = 123.95

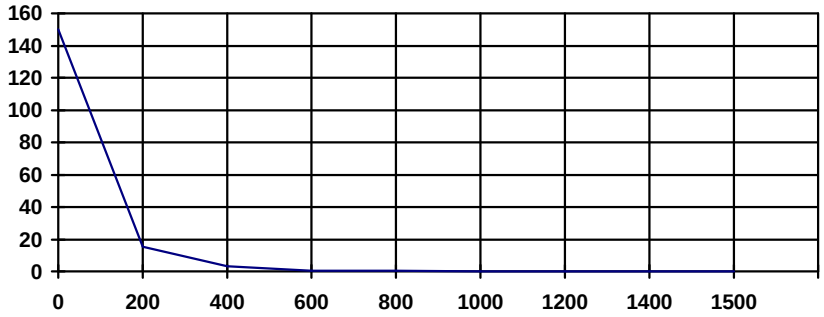
Network: *FF 30*

Training: *BP with momentum*

Test data: *Noisy letters*

Time to train to error of 0.1 = 2.34 minutes

Example 7.5 Problem with same number of neurons, but also an hidden layer.



x axis = number of training epochs.
y axis = SSE

σ = 15.13

Network: *FF 20 10*

Training: *BP with momentum*

Test data: *Noisy letters*

Time to train to error of 0.1 = 7.2 minutes

Summary for addition of layers

- ***This strategy should be applied last, after the other strategies have been tried and failed to reduce the error.***
- ***Applying this strategy causes major processing overheads.***

Adding layers to a recurrent network has quite a different effect from that detailed above for feedforward networks. A description of this is given in section 7.3.7.

7.3.2 Changing the number of neurons in a particular layer

Increasing or decreasing the number of neurons in a particular layer is one of the most obvious forms of growth [14][15][16]. It is used in a number of schemes for growing dynamic networks such a GAL [17] and ART [18]. Figure 7.7 shows a network with a reduced number of neurons in the input layer.

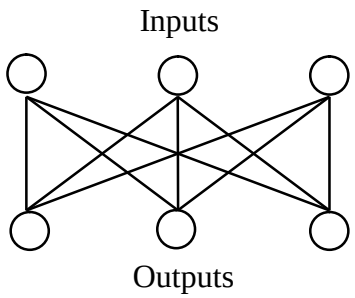
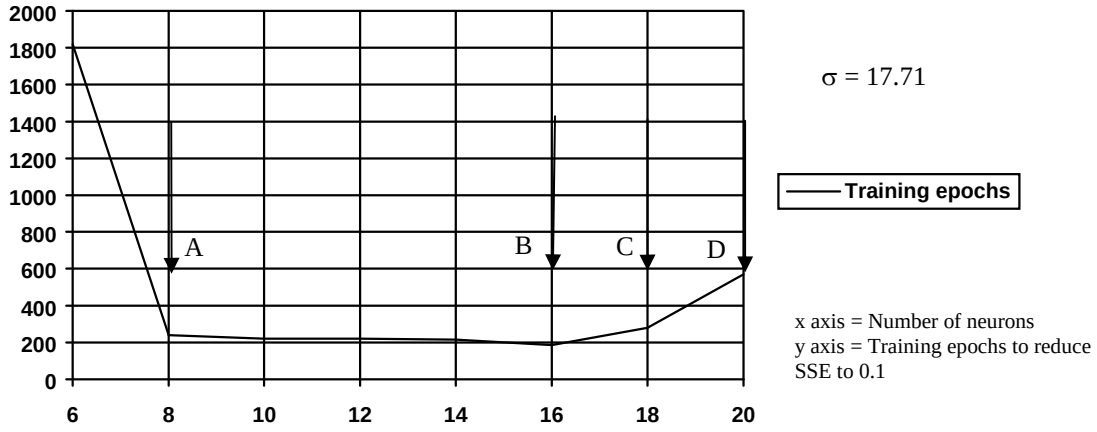


Figure 7.7
A network with a reduced number of neurons (as compared with figure 7.1).

The effect of this strategy is closely associated with the number of layers in the network. Increasing the number of neurons in the first layer of a two layer network increases the complexity of the separator. In a three layer network increasing the number of neurons in the first layer increases the complexity of the boundary surfaces. Increasing the number of neurons in the second layer has a similar effect, as it can recombine the surfaces in a more complex fashion. Increasing the number of neurons in the third layer, increases the number of enclosed boundaries possible [8]. These considerations mean that this strategy brings about a comparatively large change in error, but without a drastic increase in the processing overhead or changing the nature of the network. Therefore, in a growth algorithm, increasing the number of neurons, is one of the strategies tried first (that is, it is given a *high priority*).

Example 7.6 is a two layer network growing in a character recognition problem and illustrates the effect of growing a network by changing the number of neurons in the first layer.

Example 7.6 Result of changing number of first layer neurons.



Network: *Growing, start network FF 6 26, end network FF 20 26*

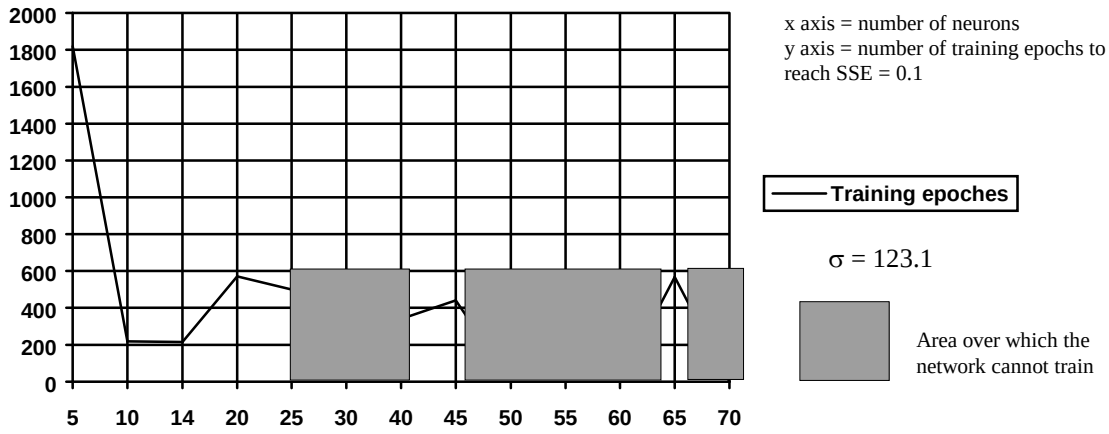
Training: *BP*

Test data: *Noisy letters*

Notice that the network cannot solve the problem at first (with less than 6 neurons), but then improves as the number of neurons increases. Finally, the network starts over-fitting [19]. Different benchmarks give different results when using this strategy. Point A represents the simplest network which can solve the problem; point B the network which trains fastest, point C is the point at which the network converged to the lowest error and point D is an overfitting network.

A peculiar facet of this strategy (and some of the others) is that although the network performance becomes worse or breaks down when large numbers of neurons are used (the overfitting problem [19]). The use of higher numbers of neurons may cause it to resume efficient operation. This effect is shown in example 7.7.

Example 7.7 Convergence and non convergence in larger networks.



Network: *Growing, start network FF 6 26, end network FF 70 26*

Training: *BP with momentum*

Test data: *Noisy letters*

The origin of this behaviour is not clear. Two possible explanations are:

a. Only part of the network is training for the problem.

b. The working areas resemble, in some respects, areas of order in chaotic systems. It is possible that overfitting is analogous to the transition from order to chaos in other systems.

A search of literature has failed to yield a satisfactory explanation or description of this phenomenon.

Summary for increasing the number of neurons in a layer

- ***This is generally the first strategy tried.***
- ***Different benchmarking systems give different results with this strategy.***

7.3.3 Changing the connectivity between layers

Removing network connections is a popular method for reducing network complexity and improving processing time. Figure 7.8 shows a network with reduced connectivity.

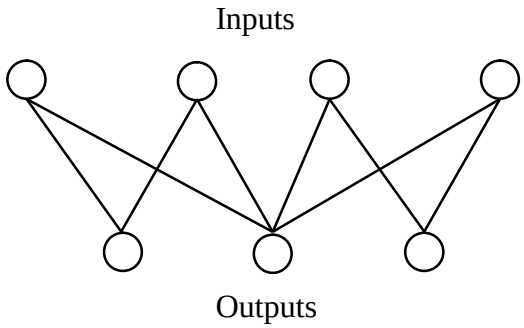


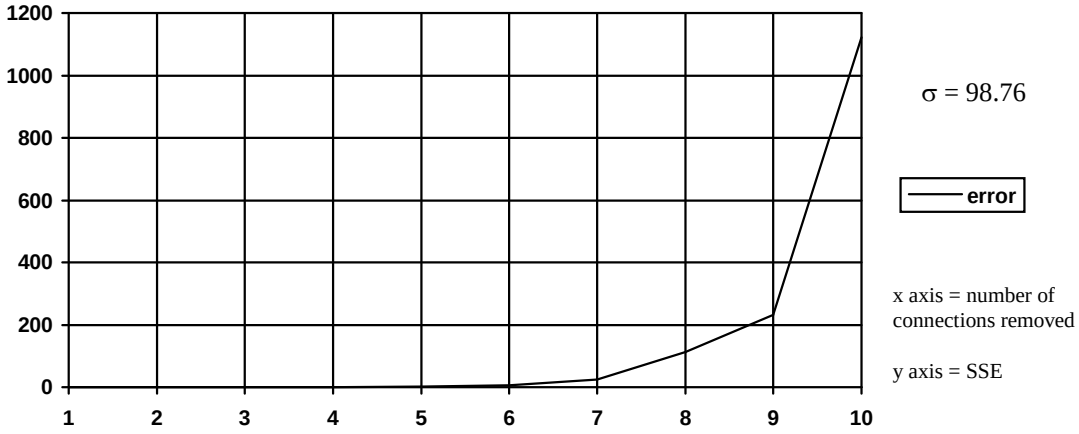
Figure 7.8.
A network with reduced connectivity.

The strategy has the advantage of not fundamentally altering network operation in the same way as adding layers or feedback. The strategy is generally referred to in literature as ‘network pruning’ [20] [21] or sometimes skeletonisation [22].

In general, we start with a fully connected network and remove connections with low activities (either with low weights or inputs). These connections have least effect on network operation. Connections with high activities can also be replaced by a unity bias connection.

The strategy should be exercised after the network has grown to its full size in terms of number of layers and neurons. Any alteration of these after pruning will alter data paths and cause problems. In multilayer networks the neurons in the next layer may also have unknown dependency on connections in the current layer. During the course of the project, methods called Taguchi methods were applied to ANNs (see chapter 9). These have the advantage that they can measure the dependencies between one layer and another and may prove useful for network pruning [23].

Example 7.8 Performance of network with pruned connections



Network: *Growing, start network FF 6 26, end network FF 6 26 (pruned)*

Training: *CM*

Test data: *Standard letters*

From Example 7.8, it may be clearly seen that a number of connections can be removed before network performance is seriously effected. The connections are being removed one by one in order of activity (starting with the one of least activity).

Summary for network pruning

- ***The strategy is usually applied last. It simplifies the network after optimum size has been established***
- ***The strategy causes fine tuning rather than major change***

7.3.4 Adding asymmetry to the weights in a particular layer

Figure 7.9 shows a network with introduced asymmetry in its connections.

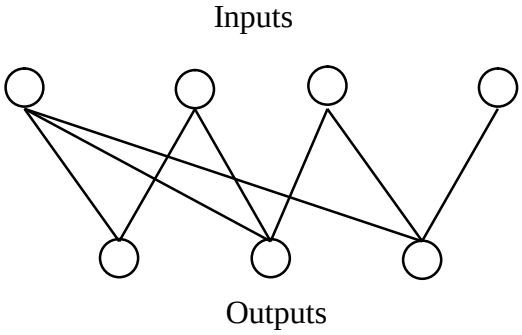
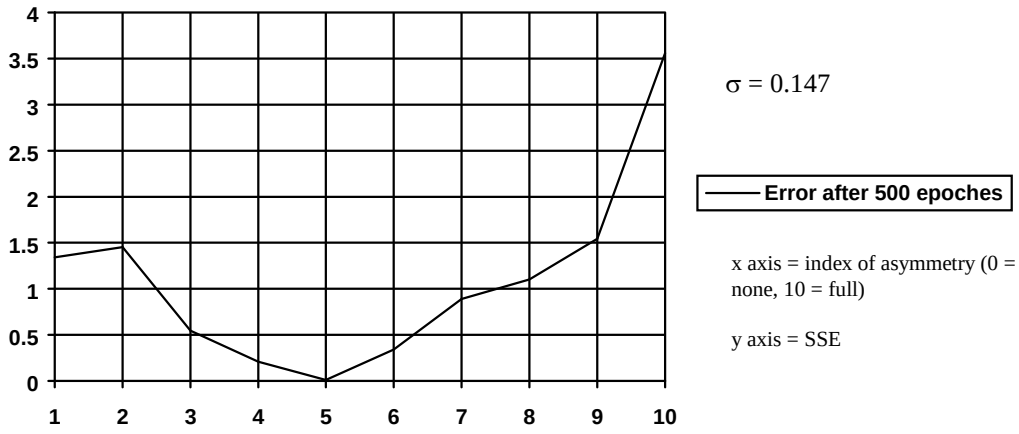


Figure 7.9
A network showing
Asymmetry.

Adding asymmetry into the network [24] connections can bias its operation. It may then display a hyper-sensitivity in that part of the data space to which the connections lead. This proves useful in non-linear problems. In these classes of problem, more processing is required by some data regions than by others. An example of this is the analogue to digital conversion of audio signals (where most of the information content is in the lower part of the dynamic range). This strategy is often given a low priority (used later in the algorithm) due to its optimising effect (what this means will become clear in the next chapter). Its actual success depends on the problem to which it is applied. Example 7.9 shows asymmetry applied to a network set up for a non-linear recognition task.

Example 7.9 An asymmetrical network performing a non linear task



Network: Growing, start network FF 2 2 1, end network FF 2 2 1 (with asymmetry)

Training: CM

Test data: Non-linear system characterisation

In this example, although the fully connected network learns to separate the data, redundancy in its connections mean that the network is not behaving efficiently. The easiest method of introducing asymmetry is to selectively prune the connections on one side of the network.

Summary for the addition of asymmetry

- **Useful in certain classes of problem.**
- **Increases efficiency of the network (due to fewer connections).**
- **Apply as one of the final (optimisation) strategies.**

7.3.5 Adding Sideways connections between neurons in a layer

If the network is operating synchronously, then we may apply the current output from a neuron to other neurons in the same layer. Such a network is shown in figure 7.10.

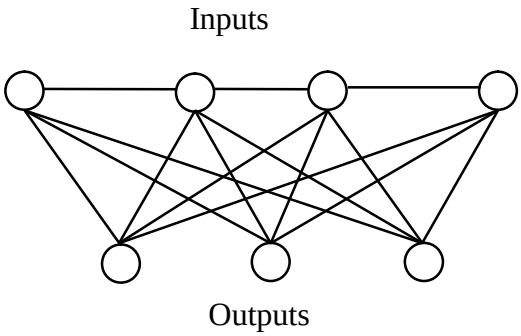
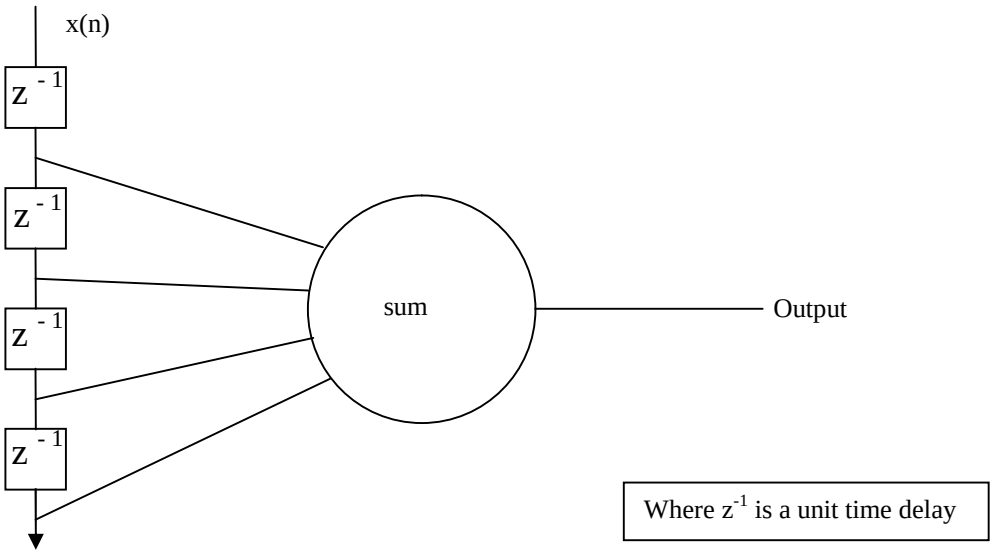


Figure 7.10
Network with sideways connections.

One way of viewing this effect is to consider it as a delay inserted into the signal path of the network, figure 7.11 (for an illustration of the operation of such a network see section 8.3.3).

Figure 7.11 Network with delay line



This tends to cause an averaging effect in the output of the network; hence this strategy sometimes finds application in networks designed for signal processing applications [26] or in networks designed for processing of temporal information, for example recognition of speech or writing [25].

Generally, the literature shows that networks which use this technique have been designed rather than grown. This is because an application of a time delayed signal can cause unforeseen circumstances, including instability (in feedback networks). The signals used in such systems are generally not spatially oriented as in the other networks considered so far (they require a somewhat different approach to training). However, a self adjusting signal processing network has great value and so the growth strategy is important and will also be useful in future work. It is included here for completeness, but is not directly concerned with the general growth systems.

Summary for sideways connections

- **Applies to a special group of networks (mainly synchronous, time series problems)**
- **Careful application is required as it may cause instability in some networks.**
- **Generally not directly applicable with the other strategies outlined.**

7.3.6 Skipping layers in a network

Figure 7.12 shows a network in which a connection has skipped a layer [27].

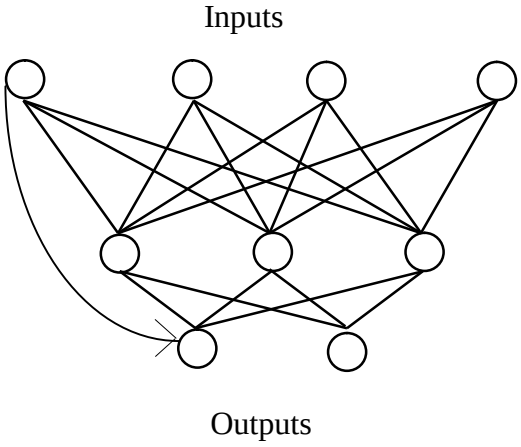


Figure 7.12
A network with one layer skipped.

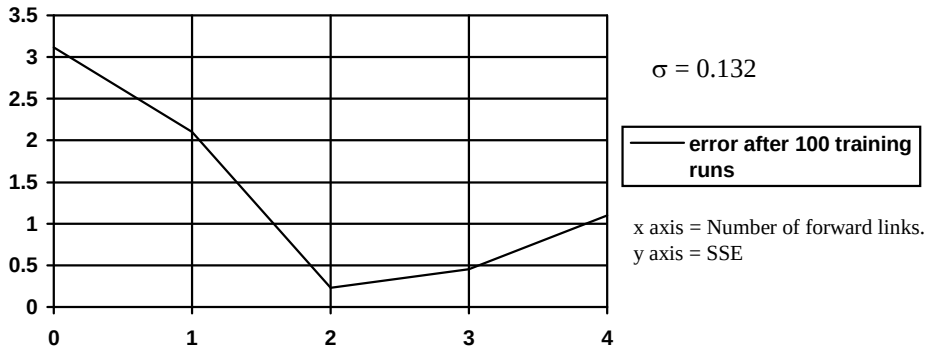
This strategy sensitises the output to activity in earlier layers. Although the action of this is somewhat complex to model and affects different networks in different ways,

there are two basic ways of describing this strategy. One is that it adds a separate input to whichever layer of the separator-constructor it is attracted to. For example, if attached to the second layer, it adds a line to a separator; this line is variable according to the input at the time. Which means that the behaviour of the network assumes a dynamic aspect according to its inputs. If attached to the third layer, it causes an input dependent enclosed region, again, this also adds a dynamic to network behaviour. The other way of thinking of the effect is that it hypersensitises the output to one or more of the network inputs. Therefore this strategy works best with networks applied to non-linear and certain time varying problems (where the boundaries, in a linearly separable sense, change with time).

The strategy may be implemented in several ways. For example, a connection could be established in turn from each neuron in the lower layer to an upper layer neuron and the effect monitored. Alternatively the opposite could also work, starting with a fixed upper neuron and trying each lower neuron in turn. The algorithm for this strategy therefore requires some careful thought. Because of its quasi-optimisation role, this strategy is best applied later (that is, given a low priority).

Skipping layers works well when there is a input or series of inputs which have much more influence on the output than others. In these circumstances, applying the strategy produces a simpler network; this is shown in example 7.10.

Example 7.10 Effect of forward skipping links on network performance



Network: *Growing start network FF 10 26 end network FF 10 26 (with links)*

Training: *GA*

Test data: *Noisy letters*

Summary for skipping layers strategy

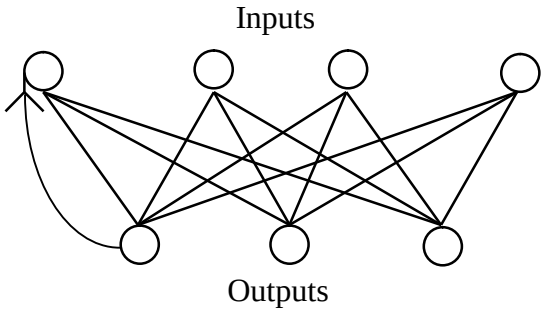
- **Generally applied to networks employed in non-linear problems**
- **A strategy best applied after others have failed**

7.3.7 Adding feedback paths to a network

The network shown in figure 7.13 demonstrates the addition of feedback to the structure [28] [29].

Figure 7.13

A network with feedback.



In section 7.3.1, the neurons were considered in terms of data space separators. A network with feedback requires a different paradigm to model its behaviour. The neurons still form separators. However, the network is now capable of oscillation and dynamic behaviour [30]. The separator model is therefore not the best way to view feedback network behaviour. An output from the network is fed back to the input and can grow or diminish depending on the weights of the signal path between input and

output. It may be seen (at least intuitively) that the relationship between feedforward networks and feedback networks is analogous to the relationship between combinational digital logic and sequential logic. This relationship is discussed in chapter 12. Feedback network function is usually expressed in terms of an *Energy Landscape* [31]. The use of the energy landscape is historical. When Hopfield started using neural models, they were considered to be useful as representations of particle spin states in a fixed lattice [32]. Therefore the term 'energy' came into use as a way of describing the neural activity, see section 3.6.1. One way of considering the learning algorithm in feedback networks is to think of it as a process in which minimas are set in the energy landscape [31]. When the inputs are placed on the network and it is allowed to relax (see section 3.6.1), it cycles round and finally comes to a halt (providing it is stable). The halting points correspond to energy minima.

Training may be done in one of two ways. Either a supervised algorithm is used (an example being recurrent back propagation) or, alternatively, a one shot training algorithm is employed (see section 3.6.1). An advantage of the one shot method is that the network maintains stability under all conditions.

Another method of introducing stability is to grow the network symmetrically [33]. If the weights form an orthogonal matrix, then the network is stable (see section 3.6.1 for further discussion).

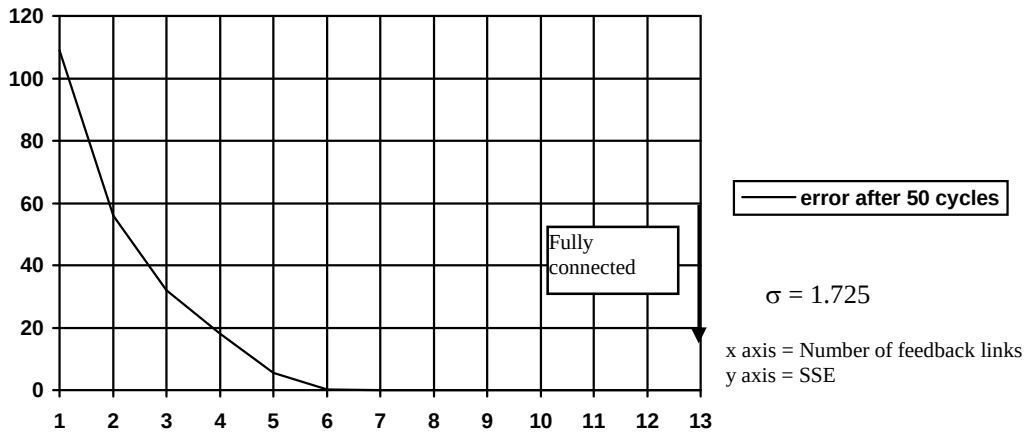
The behaviour and applications of the feedback network are different to the feedforward network. To take the case of pattern recognition as an example, the feedforward net recognises the patterns whereas a feedback net reconstructs them. This is a fundamental change in network behaviour. Adding feedback to a two layer network means that it can associate one pattern with another as described in section 3.6.2.

Adding feedback to a network is risky because of the stability problem and so is only worth considering if the network has to perform a task which cannot be performed by a basic feedforward network (for example, time series generation or reconstruction). It should be noted, however, that a feedback network can do anything a feedforward

network can (it degenerates into a FF network if the feedback path weights are zero), so there is some argument for starting with a FB network and reducing to FF if necessary. Careful consideration must be given to the training algorithm for generalised networks as methods such as BP may not work with every topology.

Shown below in example 7.11 is a network growing asymmetrically (one feedback path at a time) as a pattern reconstructor. The starting point is a fully connected FF network.

Example 7.11 Pattern reconstruction with non-symmetrical feedback



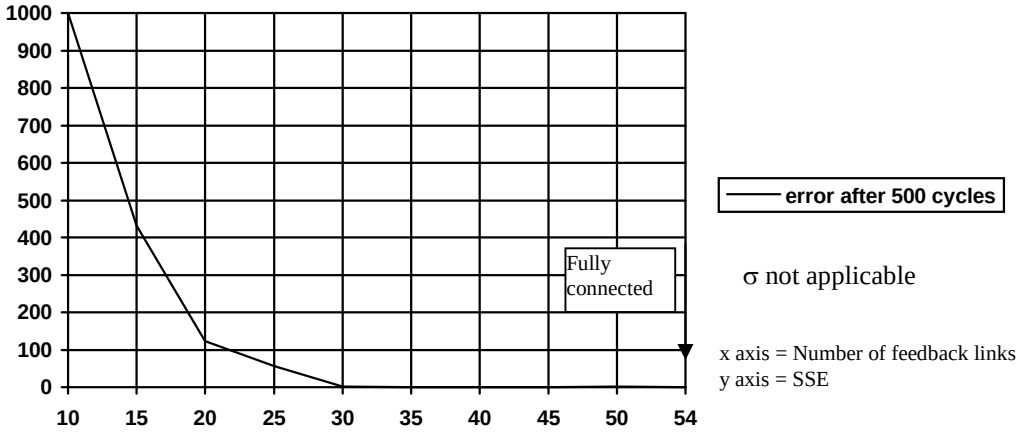
Network: *Growing, start network FF 13 13, end network FB 13 13*

Training: *GA*

Test data: *Small characters*

Shown in example 7.12 is an example of a network grown for a classical FB problem, that of time series generation. This time the network has symmetrical connections.

Example 7.12 Pattern reconstruction with symmetrical feedback



Network: *Growing, start network FF 54 54, end network FB 54 54*

Training: *Recurrent BP*

Test data: *Standard letters and numbers*

From consideration of these problems, a rule about the application of this growth strategy may be ascertained. If the growth of a feedforward network has reached a point of substantial growth but no solution has yet been reached then one FB path at a time may be added to the network until it can solve the problem. Alternatively, if the network is to be grown for a problem which can only be solved using a FB network, then the network may be grown from the outset with FB paths. These may then be pruned later if not required. This latter method appears to give the best results.

Summary for adding feedback paths

- **Strategy fundamentally changes the operation of the network**
- **Applied after other strategies have failed to reach the error target**
- **Applied to reconstruction or 'memory' type problems**
- **Can cause network instability**

7.3.8 Bias units

Adding bias may, under certain circumstances, be considered a form of network growth. Many currently used networks already use bias and adjust it as part of the training procedure for the network. As a growth strategy, it is used later, after other, more important, strategies have been used. Figure 7.14 shows a network with an added bias unit.

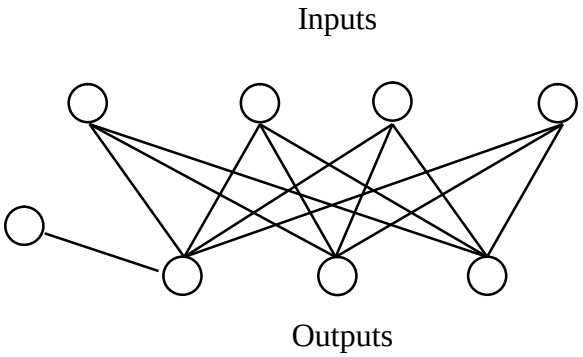
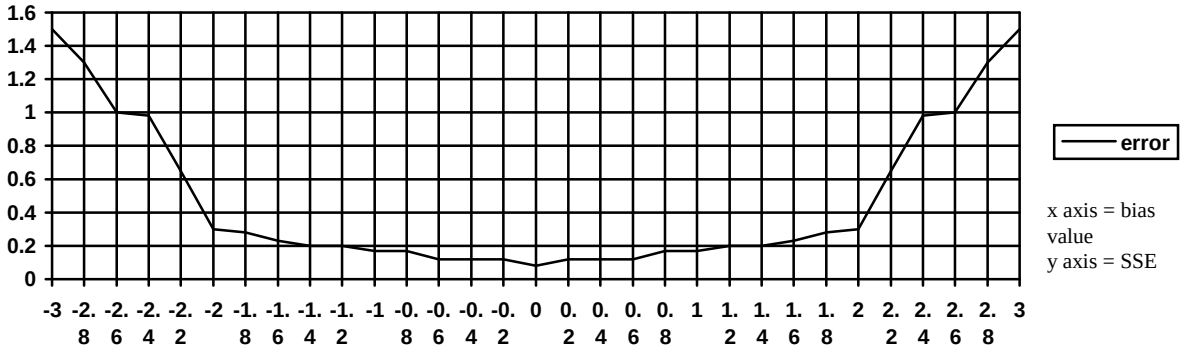


Figure 7.14
Network with added bias.

Example 7.13 shows the effect of bias on a character recognition network

Example 7.13 Effect of bias on network



Network: *FF 10 26*

Training: *BP*

Test data: *Noisy letters*

$\sigma = 0.526$

Summary addition of bias

- **Like network 'pruning' this is an optimisation technique applied in the final stages of growth**

7.3.9 Adding sideways growth to a previous layer

If, having reached a limit of network capability due to linear separability, the algorithm or network designer decides to add another layer to the network but finds that the network still does not reach the error target, then further neurons may be needed in previous layers of the network [34]. These may be separators (first layer neurons) or combining neurons (second layer neurons). A practical implementation of this strategy is problematic due to the fact that the network must be re-trained. Re-training can take one of two forms, either the neurons added can be trained using a statistical or similar algorithm, or the whole network is retrained. The latter is almost always more effective. Figure 5.15 shows a network with sideways growth.

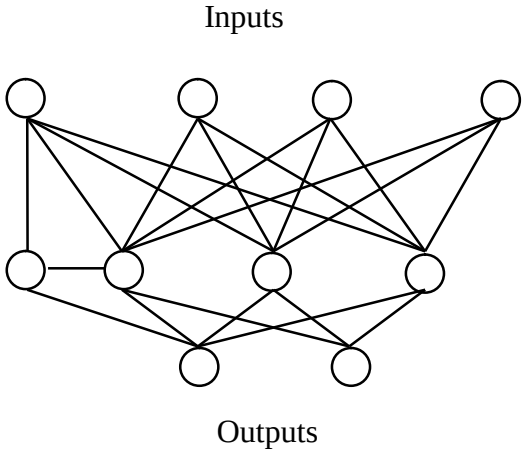
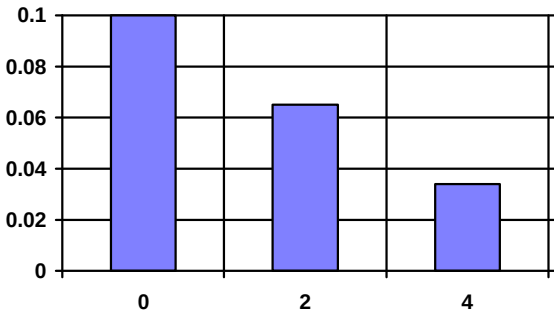


Figure 7.15
A network with sideways growth.

Example 7.14 shows a network trained using the selective application of a training algorithm to four new neurons. Example 7.15 shows the same network, this time with training applied to the whole network.

Example 7.14 Training each neuron independently



x axis: 0 = no training of new units
2 = training of two new units
4 = training of all four new units

y axis = SSE

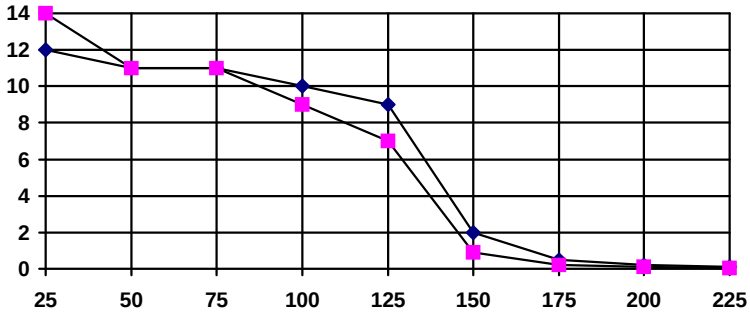
final error

Network: *FF 10 26, FF 12 26, FF 14 26*

Training: *GA (on new units)*

Test data: *Noisy letters*

Example 7.15 Training of the network as a whole



σ = 404

without sideways growth

with sideways growth

x axis = training epochs
y axis = SSE

Network: *FF 10 26, FF 14 26*

Training: *BP*

Test data: *Noisy letters*

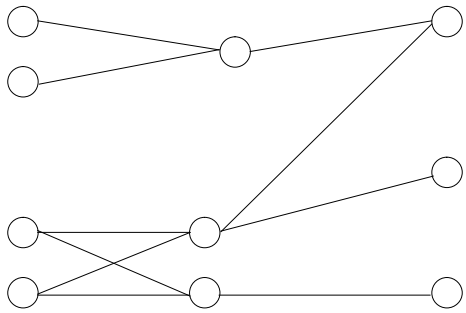
Summary for the addition of sideways growth

- **This strategy is applied if the training or error target is not low enough after reaching the three layer limit on network size.**

7.3.10 Localising Connections

In biological networks, localisation of connectivity [35] is the rule rather than the exception in network topology [36]. The fully connected network is an artificial construct. A network with localisation is shown in figure 7.16 below.

Figure 7.16 Localisation in network.

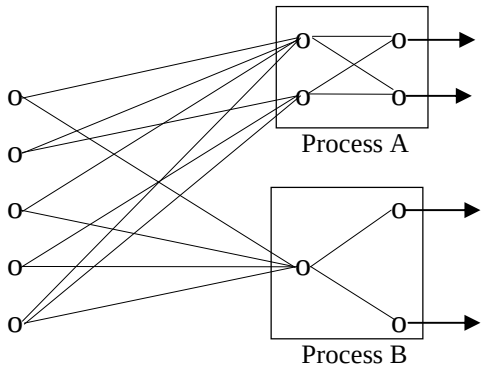


Two or more streams of neurons (and therefore signals) become separated. They may (or may not) at some point in the network meet again. This leads to the sort of ‘cascaded’ topology, seen in many BNNs.

The structure is very similar to the topology of many conventional electronic circuits and it can be seen, therefore, that not all BNNs are as fully interconnected as the literature sometimes claims [37].

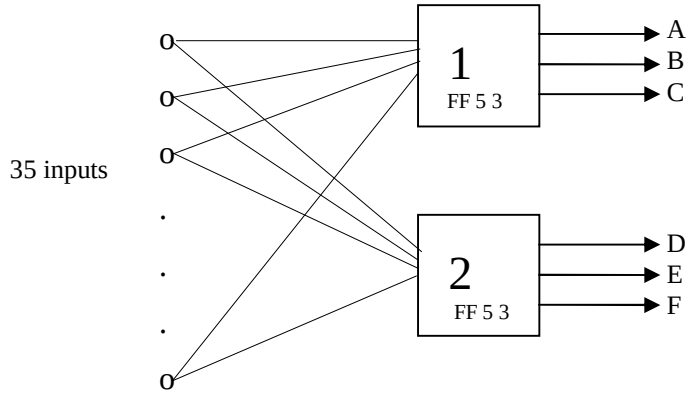
Localising connections may allow the topology of the network to become modular (see chapter 10). This leads each part of the network to process a separate part of the signal as shown in figure 7.17. However, it may be seen that this aspect is complex to implement.

Fig 7.17 Modular structure arising from localisation.

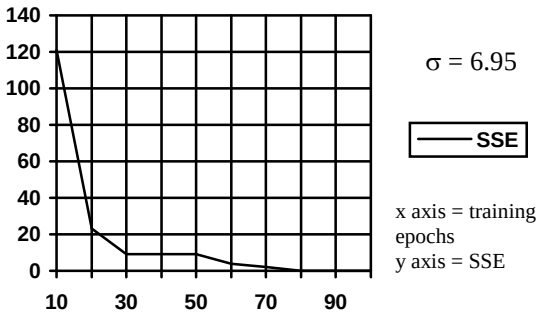


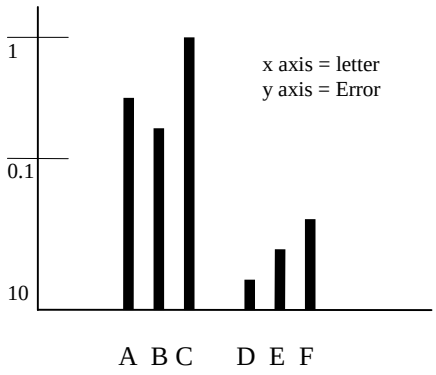
As an example, let us consider the network shown below in figure 7.18. Section 1 of the network is to be trained to recognise the letters A, B, C and section 2 the letters D, E and F. The training of the network is shown in examples 7.16a and 7.16b and the network response in examples 7.17a and 7.17b.

Figure 7.18 Network used in examples 7.15 and 7.16.

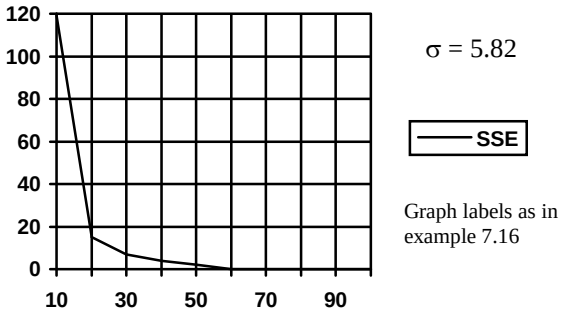


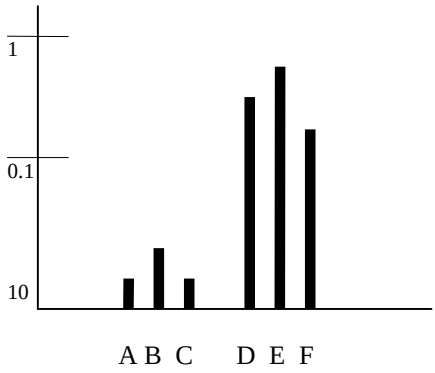
Example 7.16 a,b Training performance of modular character recognition network





Example 7.17a,b See example 7.16





Network in examples 7.16 and 7.17

Network: *Modular FF 5 3, FF 5 3*

Training: *BP (applied to each module)*

Test data: *Standard letters (restricted set)*

Several researchers have built localised and cascaded networks into robotic systems [38].

This approach is especially prevalent among workers using a biologically feasible approach to topology. The localising of connections is potentially extremely important, as it represents a ‘natural’ method of generating a modular network [39]. Some researchers consider the design of modular networks to be the next step forward in neural nets [40] since observation of the brain [41] indicates that modular construction plays a large part in its behaviour.

The use and effect of this strategy is extremely complex and still not fully resolved. Chapter 10 examines the issues involved in more detail.

Summary for localisation of connections

- **Biologically important**
- **Can partition network into modules**
- **Localisation leading to modularisation is difficult to implement (it requires a special algorithm and is not used for general problems)**

7.4 Summary

Growth strategies fall broadly into four categories:

- Those that fundamentally change the nature of the network:
  Adding a new layer, Adding feedback connections.

- Those that expand the capabilities of the network:
  Adding new Neurons, Sideways growth

- Those that optimise the network:
  Removing network connections, Adding bias

- Those that add special characteristics to the network:
  Localisation, Asymmetry, Skipping layers, Adding time delay

  Which strategies are given high priority (tried first) and which are given low, depends on the nature of the problem to which the network is being applied. This aspect will be explored more fully in the next chapter. However broadly speaking, most problems respond well to the following scheme.

- Strategies given a high priority are those which reduce the error by expanding the capabilities of the network:

a. Changing the number of neurons in a certain layer.
b. Add sideways growth to a previous layer.

- Strategies which optimise the network for a particular class of problem are only applied a) if the problem is of that type or b) if the problem type is unknown:

    a. Adding asymmetry to the network.
    b. Adding sideways connections in a synchronous network
    c. Skipping layers
    d. Applying localisation to the network

- If having tried these strategies nothing is changing, then strategies which change the performance of the network completely may be tried:
    a. Add a new layer into the network
    b. Add feedback loops to network

- Finally, after growing the network to the required size, those strategies which improve the efficiency of the network may be applied:
    a. Addition of bias
    b. Pruning the connections

To summarise the general method of growth:

The highest priority growth strategy is always the addition of neurons. This extends the capability of the network without fundamentally changing it. If this strategy is successful, then the network can be optimised by first pruning connections and then adding bias to the units. In all cases, a strategy is successful when the predefined fitness target is met within the maximum number of iterations allowed. In many cases the algorithm need not be more complex than this.

Should the network still not be able to solve the problem (meet the fitness target within a limited number of iterations), then another layer may be added and the process outlined in the first paragraph repeated (add neurons, prune and add bias). If this does not yield a working network, or the problem is known to require a recurrent network, then feedback may be added.

Finally strategies such as adding asymmetry, skipping layers or using time series can be employed if the network is being grown for a specialised task, for example, time series modelling or non-linear mapping. The order in which these strategies are applied is problem dependent (some, like time series addition need to be present from the beginning of growth). The examples in chapter 8 will help to clarify the application of these more complex forms of growth.

This scheme will become clear when illustrated in the next chapter.

Chapter 8
Results obtained from application of Growth Strategies

8.1 Introduction to chapter

In this chapter, the strategies outlined in chapter 7 are integrated into a growth algorithm and the result is applied to a representative sample of neural network problems. These examples have been chosen to illustrate all the common network types based on the McCulloch and Pitts neuronal model. Several other examples are also included to show the operation of the growth algorithm. The chapter is of critical importance since its results represent the achievement of the major objectives of the project.

8.2 Methods

The methods detailed in section 7.2 were also used to obtain the results outlined in this chapter. These techniques are discussed in appendix 2. In addition, the growth strategies were coded as the growth algorithms in appendix 3. As with the ANN coding, these algorithms were implemented in the most convenient form available at the time. The networks were tested as outlined in section 7.2 and appendix 4. The results given here are intended to be illustrative. Further examples are included in appendix 5.

In the sections below, several examples of the technique are given. These have been chosen to cover:

a) The typical applications of neural nets.
b) The modes of operation of the EA.
c) Applications which offer a direct comparison with standard models.

As typical examples the following have been chosen:

a. A feedforward pattern identifier or classifier.
b. A feedforward network operating in the temporal domain.
c. A typical symmetrical feedback network.
d. An asymmetrical feedback network.

A simple method of describing the growth process is introduced to make visualisation of the process easier. This is illustrated by the example in section 8.3.1. This also illustrates the growth process graphically and in detail. This is done so that the reader can get a clear picture of typical network changes as the algorithm proceeds.

8.3 Feedforward networks

The first major type of network which is considered is the feedforward (see section 3.5). These are networks without recursive elements and are the most widely used neural nets. They are particularly important in pattern recognition problems.

8.3.1 Network Growth

This first example is a network which is grown to identify characters on a 5 X 7 grid as shown in figure 8.1

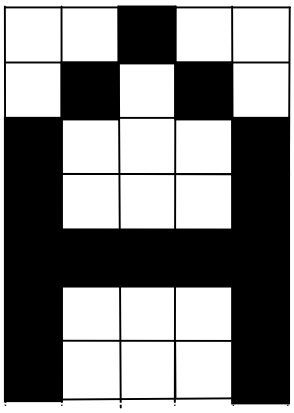
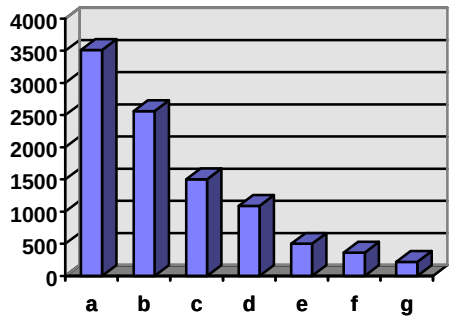


Figure 8.1
A 5 by 7 pixel grid
showing
the character **A.**

This test set (and the others used) is included in appendix 4. The network was grown using the following growth strategies (see section 8.5.1 for reasons why a restricted set is used):

- Change the number of neurons
- Change the connectivity
- Add bias units
- Sideways growth

Each of these growth strategies is denoted by a gene (or several genes) in a number string representing the network (see section 5.6). The network was grown for minimum training time and the results of this are shown in Example 8.1.



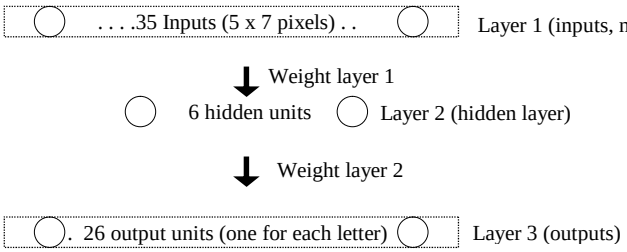
Example 8.1
The result of applying the evolutionary algorithm to a character recognition problem.

y axis = number of training epochs to train the network to a error of 0.1
x axis = evolutionary step

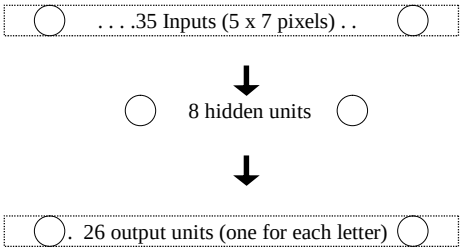
The reduction in training epochs from step a to step b is due to an increase in the number of neurons. Step b to c is also due to an increase in the number of neurons. Step c to d relates to a change in connectivity. Step d to e is caused by the addition of bias. Step e to f and f to g are due to the further addition of bias. Any further alterations to the network after this will result in a degradation of training times. The network has found its optimum size for the problem.

Figure 8.2 a - g shows the actual network growing. All training is modified BP.

Figure 8.2. Growth of network used in example 8.1

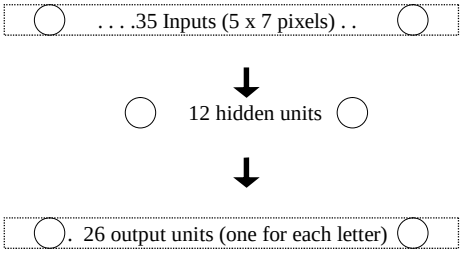


a.
Abbreviation FF 6, 26

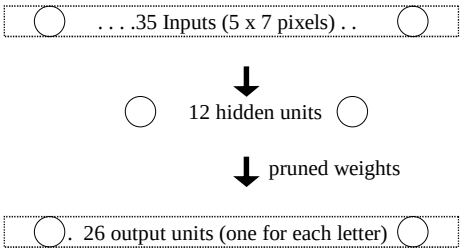


b.
FF 8, 26

Minimum working configuration

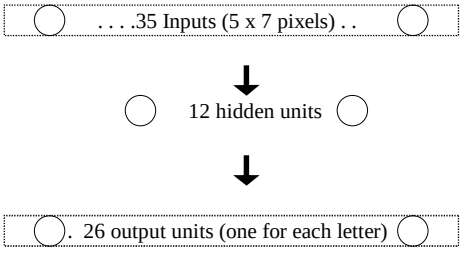


c.
FF 12, 26



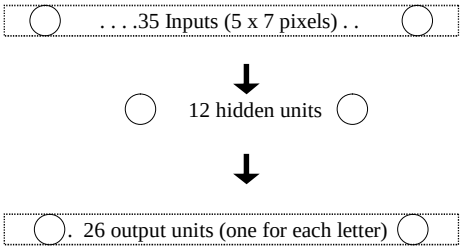
d.
FF 12, 26
(layer 2 connections pruned).

Pruned connections reduce complexity and make training faster (2% of connections removed).



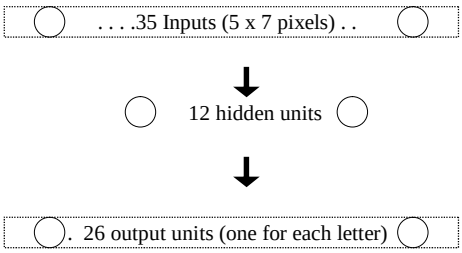
e.
FF 12, 26
(pruned).

Bias added to hidden layer



f.
FF 12, 26
(pruned).

Bias added to output layer



g.
FF 12, 26 (pruned).

Bias increased in output layer

Rather than spell out the network changes on each iteration, the following abbreviations for growth will be used in subsequent examples:

L - add layer	N - add neurons	A - add asymmetry	C - change connectivity
S - skip layer	F - add feedback	B - add bias	T - localising connections
I - add inputs	M - sideways growth		

A number is added after the letter to indicate what was added and also the layer number if appropriate. For example, N (3) 2 means 2 neurons were added to the layer 3 (layer 1 = inputs). In the case of asymmetry L or R is added to indicate left or right; + or - is used to indicate addition or reduction (pruning) of connections. After feedback S or A is added to indicate symmetrical or asymmetrical feedback. It is important to note that although L and R are included in the strings which indicate sideways growth, this was a procedural device in the growth algorithm, left and right have no meaning in terms of network functionality, we are simply adding neurons to the hidden layer.

The network in example 8.1 therefore grew in the following manner:

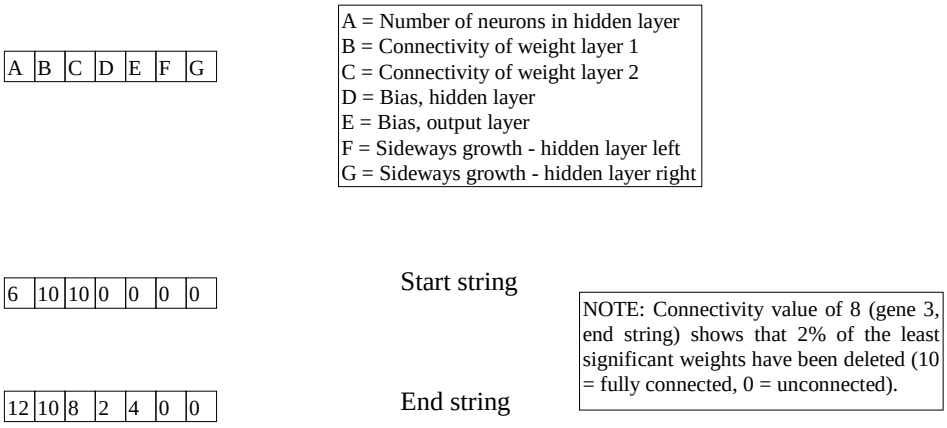
N(2)2, N(2)4, C(2)-, B(2)+, B(3)+, B(3)+

Although this is the successful combination, it is not the complete history of changes to the network. Some of those tried were unsuccessful (because they caused an increase in training time). The whole history of the network was (successful operations shown in bold, unsuccessful in italics):

N(2) -1, **N(2)2,** *N(2) -1,* **N(2)4,** *N(2) -1,* **C(2)-,** *C(2)-, C(1)-,* **B(2)+,** *B(2)+,* **B(3)+, B(3)+**, *B(3)+*

In this example the strategies were prioritised by how much they changed the network in terms of training time. Those which caused most change, (for example, increasing number of neurons) were tried first; bias, which had the least effect, was prioritised last. For more information on this aspect, see Chapter 7 and section 8.5.2 below. The starting string and finishing string for this example are shown in figure 8.3.

Figure 8.3 Growth string



This example will be used several times in the text which follows. It offers an important and convenient standard against which other approaches may be compared.

8.3.2 Measures of fitness

In example 8.1, the network was grown for training speed. There are several other fitness measures which can be used when growing the network. The most important of these are (overleaf):

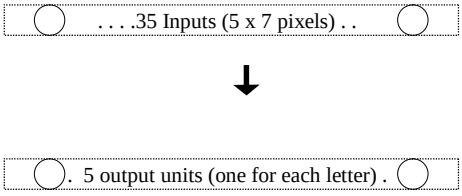
1. Network grown for simplicity
2. Network grown for lowest possible error (after fixed number of training cycles)
3. Network grown for noise tolerance

Of these, the first (Occam's Razor [1] approach) is the most important. It gives the simplest possible network (in terms of number of neurons) which can reach a set error target or fully separate a data set. Growing a network for noise tolerance is useful under some circumstances, for example when recognising hand-written characters. Growing for lowest error is perhaps the least useful and absolute errors are difficult to measure in any case.

Using the same training set and growth strategies as in Example 8.1, a network will be grown for each of these measures. Simplicity is used first, see example 8.2.

Example 8.2. Network grown for simplicity in character recognition problem.

Start network



FF 5

Single layer of 5 (binary coded) output neurons

String

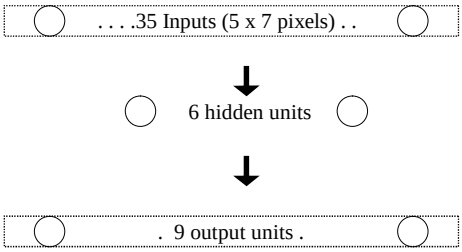
0	5	10	0	0	0	0	0

A = Number of neurons in hidden layer
B = Number of output units
C = Connectivity of weight layer 1
D = Connectivity of weight layer 2
E = Bias, hidden layer
F = Bias, output layer
G = Sideways growth - hidden layer left
H = Sideways growth - hidden layer right

Network growth (shows steps which successfully reduce final error).

L, N(2)2, N(2)2, N(2)2, N(3)2, N(3)2, C(2)-

This yields the simplest network which will train to a sum squared error of 0.1.



FF 6, 9

Final string

6	9	10	8	0	0	0	0

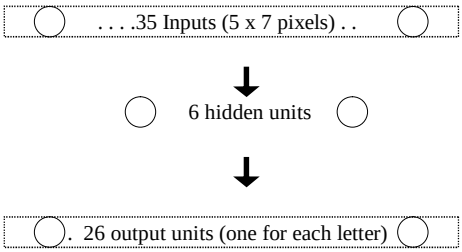
A = Number of neurons in hidden layer
B = Number of output units
C = Connectivity of weight layer 1
D = Connectivity of weight layer 2
E = Bias, hidden layer
F = Bias, output layer
G = Sideways growth - hidden layer left
H = Sideways growth - hidden layer right

This network is the first which can successfully separate all the input data. There are two measures which may be used for simplest network targets. One is an error target (such as 0.1 above) and the other is simplest structure which can linearly separate the data. In this simple case, the two solutions coincide; however, this is not always the case.

Next, consider the case of a network grown for lowest possible error, example 8.3. This usually occurs in networks just before they start to overfit. Again, for comparison, the same problem will be used.

Example 8.3. Network grown for lowest error after 20,000 training cycles.

Start Network



FF 6, 26

Start network (chosen as one of the simplest networks which can separate the dataset).

The string is different from that used in the other two models. This is because the network can grow as many layers as is necessary. The string can therefore expand to

take this into account. This property will be apparent if one compares the start and end string of the network. Such evolution is called *open ended* [2]. This network has four strategies: change layers, change neurons, change connectivity and sideways growth.

String

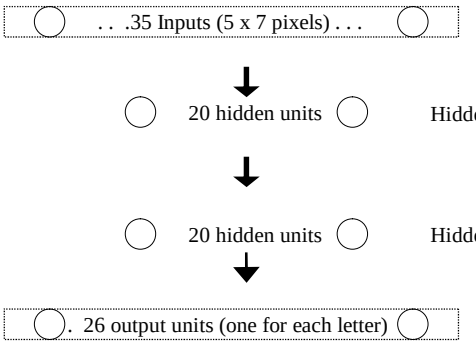
2	6	26	10	10	0	0	0	0

A = Number of layers.
B = Neurons in hidden layer
C = Neurons in output layer
D = Connectivity of weight layer 1
E = Connectivity of weight layer 2
F = Sideways growth - left hidden layer
G = Sideways growth - right hidden layer
H = Sideways growth - left hidden layer
I = Sideways growth - right hidden layer

Successful growth strategies.

N(2)4, N(2)4, N(2)2, L, N(3)4, N(3)4, N(3)4, M R(3)2, M L(3)2, N(2) 4, N(3) 4

Final network



FF 20, 20, 26

Finished network

String (notice that gene has grown with network size).

3	20	16	26	10	10	10	0	0	2	2	0	0

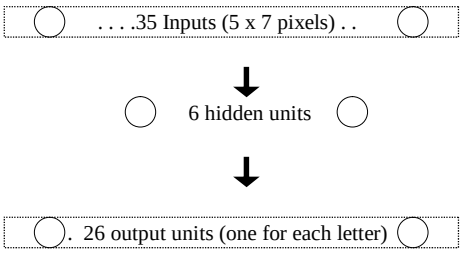
A = Number of layers.
B = Neurons in hidden layer 1
C = Neurons in hidden layer 2
D = Neurons in output layer
E = Connectivity of weight layer 1
F = Connectivity of weight layer 2
G = Connectivity of weight layer 3
H = Sideways growth - left hidden layer 1
I = Sideways growth - right hidden layer 1
J = Sideways growth - left hidden layer 2
K = Sideways growth -right hidden layer 2
L = Sideways growth - output layer
M = Sideways growth - output layer

Finally, consider the example of a network grown for noise tolerance. This network was trained with the noisy characters data set (appendix 4). The networks were

assessed by testing the probability of their producing an error at a range of added noise (for a illustration of performance see section 8.6.1). Results are shown in example 8.4.

Example 8.4. Network trained for noise tolerance.

Start Network



FF 6, 26

Start network (chosen as one of the simplest networks which can separate the dataset).

String

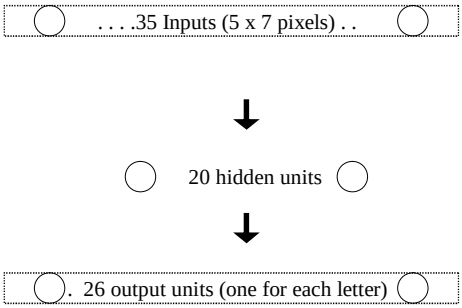
2	6	26	10	10	0	0	0	0

A = Number of layers.
B = Neurons in hidden layer
C = Neurons in output layer
D = Connectivity of weight layer 1
E = Connectivity of weight layer 2
F = Sideways growth - left hidden layer
G = Sideways growth - right hidden layer
H = Sideways growth - output layer
I = Sideways growth - output layer

Successful growth strategies

N(2) 4, N(2) 4, N(2) 4, N(2) 2

Final network



FF 20, 26

Finished network

String

2	20	26	10	10	0	0	0	0

A = Number of layers.
B = Neurons in hidden layer
C = Neurons in output layer
D = Connectivity of weight layer 1
E = Connectivity of weight layer 2
F = Sideways growth - left hidden layer
H =Sideways growth - right hidden layer
I = Sideways growth - left output layer
J = Sideways growth - right output layer

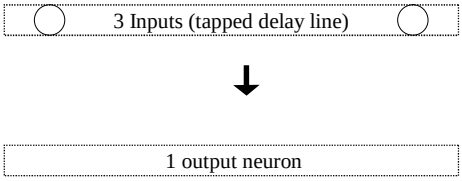
These three networks were all grown to operate on the same data, but using different success measurement criteria (fitness measure). This demonstrates the different results which may be obtained depending how one chooses to judge the network's success. It should be noted that without this method the designer would be left to guess a suitable network topology in each case.

8.3.3 Waveform prediction example

This example shows how a network grows in a typical time series problem. The problem consists of a network which has to predict a continuous time series. The output is a single neuron which tracks the behaviour of the time series at time **t+1**. The input is a delay chain of the previous values of the signal, **S(t), S(t-1) . . . S(t - N)** where **N** is the delay chain length. The error is still measured as the predicted minus the actual value. The network is tested by presenting it with a sinusoidal input which changes frequency and amplitude several times. Example 8.5 shows the development of the network which is grown for lowest error after 100 training iterations.

Example 8.5. Network grown for time series prediction problem.

Start Network



FF 1

Start network. This network will actually make an attempt at predicting the wave after only a few training cycles

String

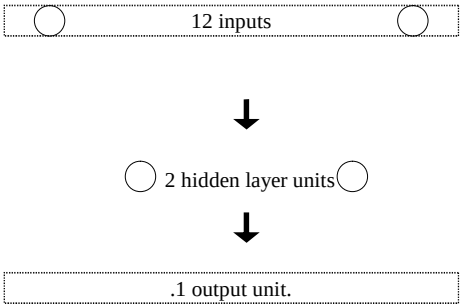
1	3	1	10

A = Number of layers.
B = Number of inputs
C = Neurons in output layer
D = Connectivity of weight layer 1

Successful growth strategies

L, I 3, I 2, N(2) 2, I 2, I 2

Final network



FF 2,1

Finished network

String

2	12	2	1	10	10	0	0

A = Number of layers.
B = Number of inputs
C = Neurons in hidden layer
D = Neurons in output layer
E = Connectivity of weight layer 1
F = Connectivity of weight layer 2
G = Sideways growth - left hidden layer
H = Sideways growth - right hidden layer

This finished network (FF 2,1 with 13 inputs) is quite different to that proposed by a reference on the subject (FF 1 with 5 inputs) [3] and demonstrates how the growth algorithm converges on the best solution.

8.4 Networks with feedback

This class of network is more general than the feedforward type encountered earlier. Their purpose is to create patterns out of related or unrelated inputs (see section 3.6).

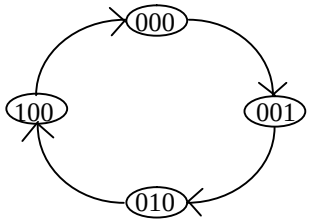
Although it is possible to grow simple Hopfield and BAM networks (see section 7.3.7), there is little point in this exercise since the neuron numbers and connectivity

is fixed by the problem specification [4]. These examples will therefore concentrate on somewhat more complex feedback network problems.

8.4.1 Network with symmetrical feedback paths

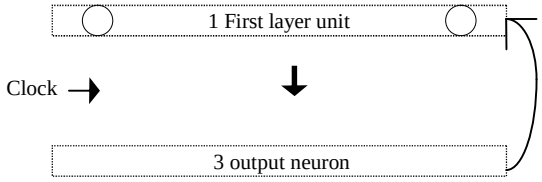
This network is being grown to produce the four state time sequence represented by the state diagram in figure 8.4. The network is operated synchronously and grown for simplicity.

Figure 8.4 problem state diagram



Example 8.6 Network growth

Start Network



FB 1, 3

Start network.

String

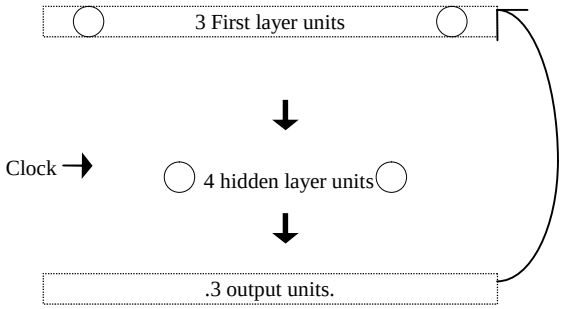
2	1	3	10

A = Number of layers.
B = Number of first layer neurons
C = Neurons in output neurons
D = Connectivity

Successful growth strategies

N(1) 2, L, N(2)2, N(2)2, C(2)-

Final network



FB 3, 4, 3

Finished network

String

3	3	4	3	10	8

A = Number of layers.
B = Number of input units
C = Neurons in hidden layer
D = Neurons in output layer
E = Connectivity of weight layer 1
F = Connectivity of weight layer 2

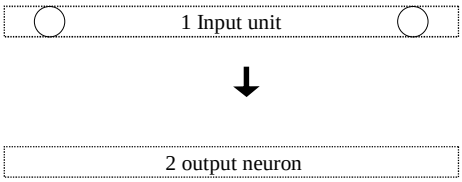
Notice that the final network operates in a hetero-associative mode.

8.4.2 Network with asymmetrical feedback paths

As an example of a network with asymmetrical feedback, a network is grown as a robot controller. The system is a simple obstacle avoidance network and is outlined in the work by Johuco [5]. Development is shown in example 8.7 below.

Example 8.7 Robot controller.

Start Network



FF 1, 2

Start network.

String

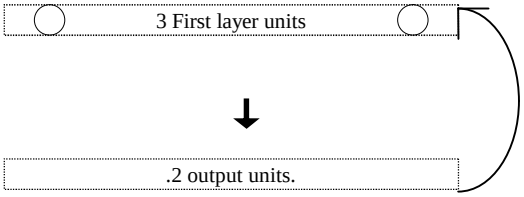
2	1	2	10

A = Number of layers.
B = Number of first layer neurons
C = Neurons in output neurons
D = Connectivity

Successful growth strategies

N(1) 2, F(1) 2 1, C(1)-

Final network



FB 3, 2

Finished network

String

2	3	2	1	2	1	8

A = Number of layers.
B = Number of first layer units
C = Neurons in output layer
D = Number of FB paths
E = FB path 1 start neuron
F = FB path 1 end neuron
G = Connectivity

8.5 Examples of different methods of running EAs

In this section, several different configurations of the EA are discussed. These include an examination of the effect of strategy priority on the end result and of running the algorithm with a few strategies as opposed to many.

8.5.1 Running with few strategies compared to running with all

First consider the character recognition problem of example 8.1. Running the algorithm with four strategies yielded the following pattern of network growth:

N(2)2, N(2)4, C(2)-, B(2)+, B(3)+, B(3)+

In this example, the algorithm can also be run with a full set of growth strategies:

- add neurons
- add a layer
- change connectivity
- sideways growth
- add asymmetry
- skip layer
- add feedback
- add bias

With these strategies available, and starting from the same seed network (FF 6, 26), growth now proceeds as in example 8.8.

Example 8.8 Network with full set of strategies available.

2	6	10	10	0	0	0	0

Start string

A = Number of layers
B = Number of neurons in hidden layer
C = Connectivity of weight layer 1
D = Connectivity of weight layer 2
E = Bias, hidden layer
F = Bias, hidden layer
G = Sideways growth - hidden layer left
H = Sideways growth - hidden layer right

N(2)2, N(2)4, N(2)2, F(1)15 6, C(2)-, C(2)-, B(2)+, B(3)+

2	14	10	6	2	2	0	0	1	15	6

End string

A = Number of layers
B = Number of neurons in hidden layer
C = Connectivity of weight layer 1
D = Connectivity of weight layer 2
E = Bias, hidden layer
F = Bias, output layer
G = Sideways growth - hidden layer left
H= Sideways growth - hidden layer right
I = Number of feedback paths
J = feedback start neuron
K= feedback end neuron

The final network is of the form FB 14, 26. From this example, it may be seen that running with all strategies is only appropriate if the nature of the problem is unknown.

For simpler problems selected strategies are usually sufficient (and in our experiments, always more efficient).

8.5.2 Effect of ordering strategies on result

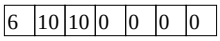
Consider the strategies used in example 8.1. These may be ordered in several different priorities. Four of these are shown in figure 8.5

Figure 8.5 Alternative ordering of strategies.

Order 1	Order 2	Order 3	Order 4
Number of neurons	**Connectivity**	**Sideways growth**	**Addition of bias**
Sideways growth	**Addition of bias**	**Connectivity**	**Number of neurons**
Addition of bias	**Sideways growth**	**Number of neurons**	**Connectivity**
Connectivity	**Number of neurons**	**Addition of bias**	**Sideways growth**

When these are applied to the same problem as example 8.1 the result is the patterns of growth shown in example 8.9.

Example 8.9 Network growth patterns using different orders of strategies.



Start string

A = Number of neurons in hidden layer
B = Connectivity of weight layer 1
C = Connectivity of weight layer 2
D = Bias, hidden layer
E = Bias, output layer
F = Sideways growth - hidden layer left
G = Sideways growth - hidden layer right

Order	Growth	Final string
1	N(2)2, N(2)4, ML, B(2)+, C(2)-	12 10 8 2 0 1 0
2	B(2)+, B(3)+, ML, N(2)4, C(2)-, C(2)-	10 10 6 2 2 1 0
3	N(2)4, ML, MR, ML, C(2)-, C(2)-	10 10 6 0 0 2 1
4	N(2)2, N(2)4, B(2)+, B(2)+, C(2)-	12 10 8 4 0 0 0

It may be seen that ordering the strategies in different priorities can result in different network structures (although these normally have similar performance). This behaviour is analogous to a BP algorithm converging to different solutions from different initial conditions [6]. It may be easily understood if one considers that the

embryology is basically, like BP, a gradient descent algorithm (see section 3.5.2) in that it always follows the path of error minimisation. Because of this, it can, like BP, find several different minima, each of which presents a low final error. These minima correspond to different network topologies.

8.5.3 Random strategy algorithm

Consider the situation if the best order of strategies is not known. The order of running could be determined by picking a random strategy and using it. As noted in chapter 5, this (and the algorithm illustrated in the next section) has similarities to an ES or EP algorithm. If the result is poor, then another random strategy is chosen. Example 8.10 shows a network generated in this way. The problem and the strategies used are the same as example 8.1; only the ordering is random.

Example 8.10 Random algorithm.

Strategies chosen at random from those in example 8.1.

6	10	10	0	0	0	0

Start string

10	10	8	2	0	1	0

End string

A = Number of neurons in hidden layer
B = Connectivity of weight layer 1
C = Connectivity of weight layer 2
D = Bias, hidden layer
E = Bias, output layer
F = Sideways growth - hidden layer left
G = Sideways growth - hidden layer right

Growth

N(2)4, B(2)+, ML, C(2)-

The resultant network is similar to that produced in example 8.1. The differences are due to the reasons outlined in section 8.5.2. The major differences, however, are due to the time taken to run the algorithm. Because the random algorithm often tries inappropriate strategies, it tends to take a less direct path to a solution.

8.5.4 Changing by random number

Another way of running the algorithm is to change the gene by a random number. In the standard version the gene is changed by one unit upwards or downwards always trying the version which makes the network simpler first. A variation of this changes the gene up and down by one unit and compares the results from both, changing the network in the direction which produces the best reduction of error. Example 8.11 shows the development of the problem in example 8.1 when tackled by a network developing under control of a random gene generator.

Example 8.11 Network generated using random algorithm.
Genes are changed by a random number between one and five.

6	10	10	0	0	0	0

Start string

15	8	9	2	4	2	0

End string

A = Number of neurons in hidden layer
B = Connectivity of weight layer 1
C = Connectivity of weight layer 2
D = Bias, hidden layer
E = Bias, output layer
F = Sideways growth - hidden layer left
G = Sideways growth - hidden layer right

Growth

N(2)4, N(2)2, N(2)3, C(2) -, C(1) -, C(2)-, B(2)+, B(1)+, ML

Again this converges into a slightly different network from the standard version. The run time can be either much greater than the standard algorithm or much smaller (particularly if the network requires major changes to converge). This approach helps to limit problems with the ‘local minima’ type effects described in section 8.5.2. The algorithm could also use a Boltzman or simulated annealing type effect to select gene changes (see section 3.5.3).

8.6 Benchmarking EAs

The purpose of this section is to compare the results obtained using this technique with some standard examples and the leading technique which is in competition - Genetic Algorithms. Since the results have already been presented in previous sections, they are quoted in the following without elaboration.

8.6.1 Comparison with recommended standard networks

In many of the previous examples there has been an emphasis on character recognition networks. This is for two reasons: firstly, they provide a typical example of a pattern recognition network and this class of ANN is the most widely used and most important. Secondly there are many published examples with which to compare the results.

The most efficient published ANN for character recognition is FF 10, 26 (fully connected), used in the MATLAB neural nets toolbox [7]. The performance of this network is shown in figure 8.6 below.

Figure 8.6

Training Cycles:	239	(example 8.1 = 167 cycles)
Network Structure:	FF 10, 26	(example 8.2 = FF 6, 26)
Maximum noise for 0% error:	30%	(example 8.4 = 45%)

Comparing first with a network grown for simplicity (example 8.2) we can see that the grown example uses a simpler topology and is therefore more efficient in terms of processing over-head. Compared to a network grown for training efficiency (example 8.1) the standard network is much slower in training, 239 cycles compared with 167 cycles. Finally in terms of noise performance (example 8.4) again the grown network performs consistently better. Of course this comparison is ‘cheating’ somewhat as it compares one network with three; however, it still illustrates the improvement one can expect by growing the network.

A standard network for waveform prediction is proposed in the MATLAB literature [8]. Comparing this with example 8.5, we can see that again, the grown network performs better (12.7% phase error as opposed to 19.1%).

Example 8.5 Phase Error over 1000 cycles	= 12.7%
Standard network Phase Error over 1000 cycles	= 19.1%

8.6.2 Comparison with GAs

Figure 8.8 shows the performance of a network, grown with the standard GA described in section 5.4.2 for the problem outlined in example 8.1.

Figure 8.8 Comparison with GA

Finished network structure:	FF 16, 26
Training cycles:	154
Maximum noise for 0% error	40%

The performance of the grown network is directly comparable with this network. Yet the GA took around 18 times longer to produce its version (the inefficiency is due to the GA having to test each member of the population, see section 5.3.1).

8.7 Summary and discussion

The examples above were chosen from the literature on neural networks, which often suggests a network suitable for application to the problem. In each case, however, the result of the growth algorithm produced a different network from the original which performed better than the suggested network. In some cases the network was also simpler than the ‘standard’ network (and always simpler if it had been grown for simplicity).

The examples were also chosen to cover all the basic types of network in common use. These are: the feedforward network, the recurrent network (with symmetrical and non-symmetrical connections) and time delay networks. The strategy for growth outlined in section 7.4 was applied to each of these in turn and the various networks show consistent results when grown using the method.

It is obvious that this technique is an extremely useful and powerful method for designing ANNs, especially when the designer is not sure of the best topology to use in the problem. The technique is therefore a relatively simple algorithm which will automatically find a good network to apply.

As demonstrated, there are several different ways of running the algorithm and different bench marks which can be used. The final decision depends very much on the problem at hand. A constrained problem such as waveform prediction yields to certain obvious strategies. A unconstrained problem such as a robot controller requires more strategies.

Chapter 9
Learning Algorithms

9.1 Introduction to chapter

Designing the structure and topology of an ANN is only half the problem. The second consideration is training the network. Although an investigation of learning is not a specific aim of this project, an understanding of its role in evolutionary networks is important as it affects the performance of the final system.

The chapter is split into two sections. Firstly, ‘standard’ algorithms for learning, such as Back Propagation and Statistical Methods are discussed in the context of evolutionary networks. Secondly, training methods related or specific to Evolutionary ANNs are considered. These include Taguchi Methods - a learning algorithm developed as a spin-off from the project. Incremental learning and learning by example are also detailed.

9.2 Traditional approaches to learning

In this context, ‘traditional approaches’, means those algorithms used to train ‘normal’ (non-evolutionary) networks. These methods include Back Propagation and its more general form, Recurrent Back Propagation, Statistical Methods such as Boltzman and Simulated Annealing and finally, Genetic Algorithms. One approach not considered is that of Competitive Learning. This is for two reasons; firstly, the subject is covered in the literature on ART networks and secondly, competitive dynamics have not been used in the networks presented in the thesis.

9.2.1 Back Propagation

For all its problems, back propagation is still the most popular learning algorithm. The reasons for this are two fold (overleaf):

- It is tried and tested, simple to implement and has many examples in literature.
- Compared with some other methods it is fast. This is due to the fact that it is not a blind search like GAs, but a gradient descent algorithm.

As noted in section 3.5.2, BP has several problems, including local minima and paralysis. Several versions of the algorithm have been devised which attempt to overcome these. For example, it has been combined with statistical methods [1] and this allows the solution to 'bounce' out of local minima.

Another improvement is the addition of momentum [2]; this has several different benefits, including an increase in speed. The fact remains, however, that in the case where BP is left on its own to train networks (as is often the case with Evolutionary ANNs) it can run into problems. The main result of this is that networks appear not to be working, when in fact it is the training algorithm which is not converging. For this reason, it is worth investigating some of the better (but more complex) BP algorithms. Combinations of BP and Simulated Annealing are useful in this respect because they will always find a solution but the algorithms are often complex.

A variant of standard BP called Recurrent Back Propagation [3] will train almost any topology of network. It is possible to confuse it with esoteric structures and for this reason other methods, such as GAs, are preferred, where research into structure is being undertaken.

The problem with BP, as with many of the other algorithms, is that, on each iteration of the network the whole structure is retrained. This is extremely time consuming. One solution which was attempted, was to train only that part of the network which has been added. This can be done successfully (see example 7.14 and 7.15 and also example 7.16) but the trouble is that, although it improves the network (at least in terms of final error), it does not take advantage of the expanded capabilities of the network as a whole. One has to be particularly careful when using BP in this way, as it is quite difficult to structure the algorithm. Training small added parts of the network is more effective using one of the statistical methods. This type of training may be refered to as *incremental*, because it one trains that part of the network, which has

been added in the last iteration.

An interesting philosophical queston is apparent from the discussion above. How does BNN train itself as it grows (in evolutionary time). Surely the individual neurons themselves must control the training so that a new neuron must have some way of optimising itself. This is much the same argument as Hebb used in his theory of learning (see section 3.2.2). Does the neuron do this by strengthening its connections or by 'rewiring' itself ? This question will be addressed in chapter 12.

To summarise the main points above:

- Back Propagation is the best learning algorithm for Evolutionary Neural Networks which are doing 'standard' tasks such as pattern recognition or classification.
- The more complex forms, which are not prone to local minima, should be used.

9.2.2 Statistical Methods and Genetic Algorithms

Statistical Methods such as Boltzmann, Cauchy and Simulated Annealing are outlined in section 3.5.3. These are useful only as research tools, due to the time taken to find a solution. The time taken also increases rapidly with the size of network. The advantage is that a well structured simulated annealing algorithm will always reach a solution, independent of network topology. As a result, statistical methods are useful for investigating some of the more unusual topologies of network.

They are also useful in training small numbers of neurons, added to the network, as part of the growth algorithm (see section on BP above). This is because the task is much smaller and it is relatively quick for the algorithm to search the weight space for the correct solution.

Genetic Algorithms are more useful. They are simple to code and implement and are generally faster than statistical methods (although this varies). Their disadvantage lies in that they must evaluate a large number of networks before the next iteration. However, they will train any network topology and when set up carefully, are not as prone to local minima as standard BP. Their usefulness therefore lies somewhere

between BP and statistical methods.

9.3 Other methods

The other methods, listed below, are either discussed because they were tried during the course of the project or because they represent a good opportunity for further research.

9.3.1 Fixed functionality - evolution as a training algorithm

Let us assume that it is not a network of neurons we are growing, but a network of logic gates. In this case the evolutionary algorithm - whether it be an embryology or a GA, will arrange and rearrange the network until the fitness increases and the functionality requirements are met. This is exactly what Adrian Thompson [4] has done. He has taken an arbitrary problem (in his case, generating a clock pulse) and used GA techniques to develop a solution. The result is that a circuit is evolved, which fulfils the function of the experiment. In other words, the GA is behaving as a learning algorithm, except that, rather than training a fixed network, it rewires the network. Given this result, why bother with a learning network at all? Why not simply take fixed gates and arrange them until the circuit performs the functionality required? The biological neural network has units (for example, electrically gapped neurons) which probably have a limited learning capability. These units tend to be contained in the PNS. This scheme has a lot to commend it. Evolving an ANN is a two stage process: place the neurons and then train the network. This adds time and complexity to the task.

The paragraph above poses a fundamental question: *Can a trained ANN do anything which a genetically created digital network cannot*? With present digital logic, the answer is yes, because an ANN has a continuous (analogue) output. Digital gates are limited to discrete (binary) outputs. However, in Appendix 8, another type of technology is presented, that of fuzzy gates and these appear to have an advantage over other technologies. It should also be possible to train these fuzzy gates and therefore optimise them, if required, after they have been placed.

9.3.2 Taguchi methods

Most neural network training methods are slow. Some of them do not work with all topologies. During the project, it was realised that certain mathematical techniques, known as *Taguchi methods* [5], could be used to train neural networks.

These techniques were developed by the mathematician Dr Genuchi Taguchi and are used in process optimisation. The method consists of a limited number of predefined experiments, the results of which can be used to calculate the optimum configuration of system variables. In the case of ANNs, a number of trials are set up using networks with different weights. The results of these trials are used to calculate the best set of weights for the network. In other words, the variables which the method optimises are the network weights.

A full outline is given in the paper in appendix 1, but the advantages of these techniques are:

- They are a one shot method of training networks (not iterative like BP) and are therefore fast.
- They can be used for rough training and then followed by BP or statistical methods which finalise the training (effectively fine tuning the network).
- They are capable of being used to analyse the network, in particular, they can be used to measure interdependency within the network.

9.3.3 Learning by example

No evolutionary strategy can succeed without a *guiding principle.* In other words, a sense of what is right and what is wrong. Consider an autonomous vehicle as an example. We would like to evolve the neural network controlling the vehicle. The object is to make the vehicle more intelligent and to stop it falling off tables or bumping into walls. However, this is impossible if we do not let the vehicle know that falling and bumping are bad. In the natural word, dying is bad and you do not want to do it. Living is good and we would like to carry on doing it. The guiding principle is usually expressed in the form of a fitness - the degree of ‘badness’ or ‘goodness’ of an action or series of actions which the network takes. Designing fitness functions is one

of the most difficult areas in EA or GA design.

The standard algorithms such as back propagation can be altered to work in this way. Returning to the vehicle again, let us suppose that it is fitted with bump sensors. If it bumps into a wall and continues to go forward, this is bad and the error fed-back to the network is unity. If it reverses backwards to escape, the error is zero and this is fed back to the network. Other, in-between responses, give errors between one and zero.

The approach outlined above has its limitations and a number of researchers have developed other training methods based on trial and error or learning from the environment. These are interesting if the user wishes to consider applying the networks to agents which operate within the real world. These training algorithms are discussed and referenced in Gurney's 1997 book [6].

Chapter 10

Application to non-specific problems

10.1 Introduction to chapter

At the start of the project, it was hoped (rather naively) that a way of growing networks to perform well on unconstrained data sets would be found (this was one of the original objectives of the project). This has not been forthcoming. The problem of unconstrained data is probably the most important in ANN research. The BNN handles this problem by using many small networks each working on a small part of the whole. Such an approach is called *modular*. Therefore this is the logical place to look when developing networks for non-specific problems. The modular nature of the biological network is discussed in detail in section 2.4.

This chapter can, at best, only make suggestions for further work in this area. The growth algorithms outlined earlier in the thesis join the many successful Genetic and Evolutionary techniques designed by other workers. However, no one has yet applied these to a satisfactory framework for the evolutionary generation of a modular network. This would be an important step towards truly intelligent systems. The chapter starts by describing the importance of the problem. It then briefly reviews previous research in the field before studying some ideas for further investigation.

10.2 The importance of non-specific problems

ANNs are normally applied to known, constrained problems. That is, problems in which the inputs have been carefully pre-processed so as to remove any extraneous data and where the nature of the classification problem is already known [1]. Networks also normally operate on a single problem domain (tackling one problem). This is obviously at odds with the organic brain, which is not only capable of operating in an unconstrained environment but is also capable of processing many problem domains (vision, audio, smell, etc). The reason for this limitation is because the network, when presented with many disparate patterns cannot decide which to classify.

A general Neural Network classifier is one capable of operating on a loosely constrained input data set with a non-specific problem domain. The advantages of this are obvious; such a network could control many tasks which are currently undertaken by humans. Intelligent robotics is just one possible application; others include vehicles which can guide themselves and intelligent household appliances. A *General Classifier* would therefore be a first step towards real intelligence in networks, as all real world problems are massively unconstrained.

10.3 Work on modular networks.

The importance of modular networks is well understood and is a popular subject of research in ANNs. Modular networks have been devised which are successful in several areas. Often, however, when one examines the design of these networks limitations become apparent. These usually arise from their synthesis, which often seems artificial. In some cases the modular structure is designed beforehand. In others standard modules are placed in a structure but the modules themselves are not evolved or designed. Strangely, although modular structure is a stepping stone towards operation in an unconstrained environment, workers seem reluctant to try their networks in such systems and many present results from problems such as character recognition, which yield to simpler network structures. Given below is a sample of typical work on modular ANNs (these are representative of the available methods). There are many more examples available in the literature.

Cho and Shimohara [2], present a network in which standard modules are placed in a genetically defined structure. The modules themselves have a designed functionality. The network has a strong biological flavour, having inhibitory and other similar biologically inspired connections. The modules are placed using Genetic Programming. Experimental results are given for handwriting recognition, but the network is aimed at controllers for behaviour based robots.

A network using modules called CALM is presented by Happel and Murre [3]. These modules have a variable size and are placed using a GA. Again, character recognition results are given for this structure.

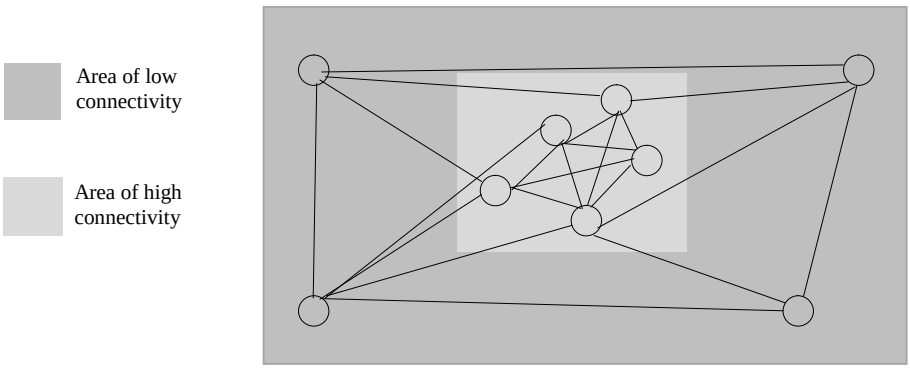
A different approach is taken by Yao and Liu [4]. They take the best individuals from an evolved population of networks and use either majority voting or linear combination of results to give an output. The paper goes on to show that this technique gives better results than an individual network (this approach is not modular as such).

Ceravelo et al [5] has produced a designed hierarchical structure for a damage assessment network. Several lower networks work under the direction of a 'manager network'. One interesting aspect of this approach is that the networks are of the Grow and Learn type outlined in section 6.3.2.

10.4 Other Approaches to modular networks

One possible method of growing modular networks is to use a mathematical model which has modularity built in. In other words, a mathematical description of the network connections which displays local areas of high density and other areas of sparse connectivity. This is best illustrated pictorially, figure 10.1.

Figure 10.1 A non homogeneous network.

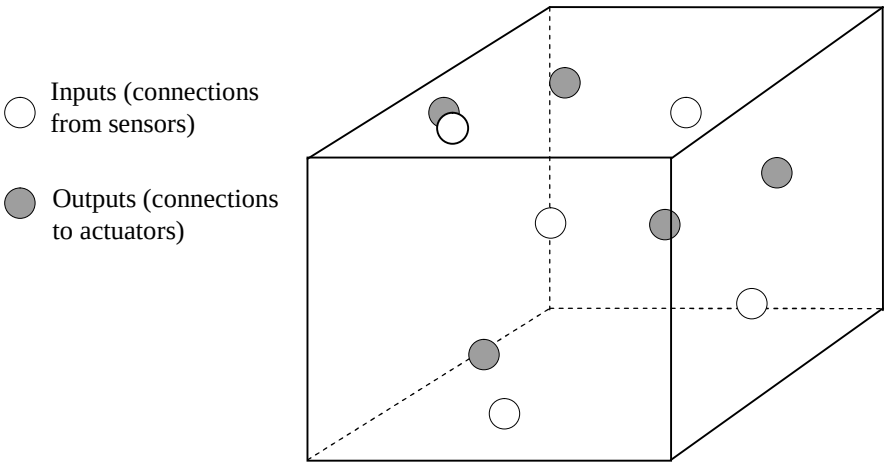


This idea may turn out to be biologically plausible (although there is no evidence at the moment for it). The gene which governs branching in biological networks has been isolated recently [6] and there seems no reason why *clustering* of this type may not also be genetically controlled.

Several mathematical models produce results similar to those described above. One well known example is that of fractals [7]. There is no satisfying description of what a fractal actually is. There are a group of related, self-similar, biologically relevant mathematical models which can show structures like that described above. Self similarity and symmetry are important in all biological systems. Another example is Dawkin's Biomorphs themselves, the original forms of which show tree like structures which demonstrate connectivity variations of the type normally seen in natural systems. Another variation (also named 'Biomorphs') is reported by Stuedell [8]. In such systems, a population is produced by running the mathematical model with different conditions. Selection may be achieved by a system similar to that proposed by de Garis [9]. That is, a group of networks selected by GA is rated for fitness and forms the basis of further development.

Another, quite different, approach is to consider evolution of the nervous system from the point of view of the environment in which the organism finds itself. The nervous system is just the 'wiring' of the organism. What makes us think that we can develop any meaningful system in a disembodied situation without environmental stimulus? After all, this is the pressure which caused the BNN to develop. To take this a step further, looking again to embryology, we see that the origin of the neural cells is in the ectoderm (see section 4.8.2). This strongly suggests that, in evolutionary terms, neurons started life as sensory skin cells (see section 4.6). Neural evolution is therefore closely tied up with sensors and actuators (because there must have been some reason for receiving a stimulus and that must surely have been to elicit a physical response to it). The basic neural network is, therefore, a series of connections between sensory cells and muscles. One can envisage a space, *the evolution space* into which the network will be placed and in which it will grow (or could be defined by a mathematical model as above). Such a system might form the basis of an evolutionary ANN, figure 10.2.

Figure 10.2, Evolution space.



Let us take these ideas one stage further. Why design modular networks using contrived mathematical models? What if the modular structure arose naturally from the network? The section below illustrates such an idea. It is based on clues from biology and has the benefit that a modular structure arises without help. The reason for this is that the network is part of the *system* which has to live and interact with the outside world. When humans design systems, they use individual pieces of electronics wired together to form a whole and this philosophy automatically gives rise to a modular structure. To design a neural network system therefore demands that we apply evolution in a systems context.

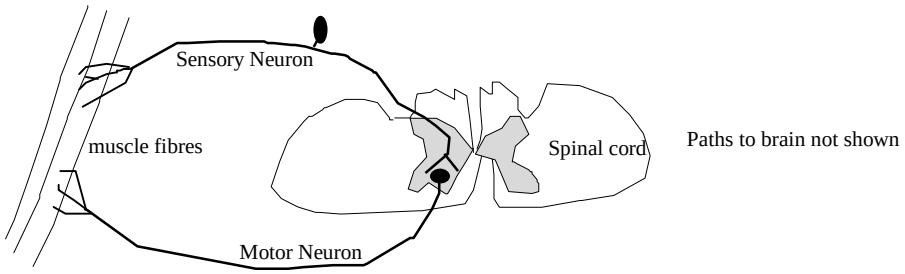
10.5 A framework for Evolution of an Animat Nervous System

The sections below outline ideas for a possible way to address the problem of creating an ANN which will operate in an unconstrained environment (or with a non-specific dataset). It focuses particularly on designing controllers for animal-like robots which are known as *animats* [10]. An animat is a (usually small) robot which is either pre-programmed to behave like an animal or learns animal-like behaviour. The idea is to develop robotic agents which are largely autonomous. However, the ideas outlined below are applicable to other similar problems.

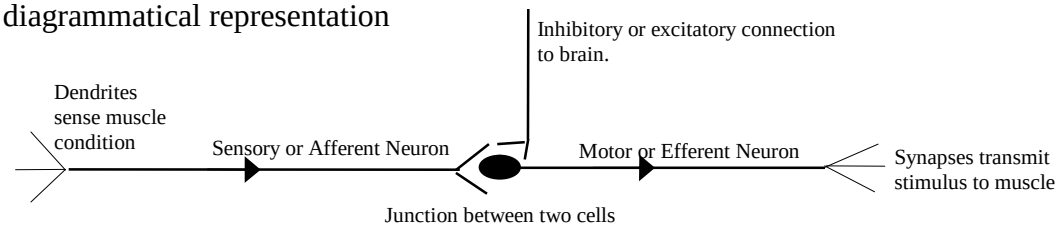
10.5.1 A Biological view of the nervous system.

In chapter 2 the nervous system was described as having two components: the Central Nervous System (CNS) and the Peripheral Nervous System (PNS). The CNS comprises the brain and spinal cord and the PNS, the nerves running to the extremities of the body. These PNS nerves consist of Sensory or Afferent Neurons which monitor the state of the system and Motor or Efferent Neurons which send control signals to the muscles [11] [12] [13]. The general arrangement of these is shown in figure 10.3 below.

Figure 10.3, Sensory and Motor Neurons and how they interact in the spinal cord.



diagrammatical representation



Each of these neuron pairs (sensory-motor) might be thought of as a reflex. The sensory neuron monitors the muscle and, depending on its input, the motor neuron sends an excitatory signal. Actually, the system 'wiring' is a little more complex then this and different reflexes have different connectivity, see [14]. As shown above, the action of these neuron pairs is controlled and mediated by excitatory and inhibitory connections from the brain [15] [16]. This structure may have evolved in the segmented worms (annelids [17]), which essentially have a reflex centre in each segment (see section 4.8.1).

In this model, the brain itself might be viewed as a processing centre for sensory signals such as vision, taste, smell, touch and sound. The job of the brain is to mediate the reflexes by means of the inhibitory and excitatory connections to the spinal cord. The brain therefore sends signals which initiate reflex actions in the presence of food, danger or other stimuli.

Between the action of reflexes (low level) and that of the brain (high level) lies an extremely important group of actions which are not just reflexes, but also not thought. These are the co-ordinated automatic actions, examples of which are walking, swimming (in marine animals), running and flying (in birds). Experiments have shown that these actions are controlled by the spinal cord and not the brain [18] [19] (it is possible to destroy the whole upper brain and still preserve walking and swimming in many animals).

Consider what these co-ordinated reactions or reflexes have evolved from. In lower animals such as *Hydra*, which have very simple nervous systems, the neurons are arranged as an homogenous network over the whole animal. This is known as a nerve net [20]. Each neuron is connected to all the others around it. The neurons serve both as sensors and motor stimulators (this structure may have arisen as a network of specialised skin cells). When a sensory input is made (for example, by touching the animal) the muscles at that point react. This is thought to be the reason for the evolution of the nervous system in the first place- localised communication, as opposed to hormones which effect the whole body- in other words the BNN was evolved to be modular! The signal is at the same time transmitted to the other neurons close by; they also react, but less strongly. The motor effect therefore diminishes with distance. One can see this type of action in a sea anemone as it captures some food [21]. The food (say a fish) touches a tentacle, which reacts strongly and pulls the fish in. As it gets drawn in, the prey touches further tentacles which envelop it (and provoke a similar, but less strong reaction in their neighbours), until finally it is in the grasp of many. The development of the reflexes themselves probably started with the development of limbs. This would have physically isolated that part of the net - or at least caused in an increase in local complexity (this process has already started in Hydra see illustration in reference [20]). Any attempt to evolve the reflexes

themselves will therefore be bound up with the evolution of the limbs (and the body plan in general).

In the action described above, a primitive type of co-ordinated response is present, due to the sideways connections between neurons in the nerve net. This may have been the primitive origin of the co-ordinated reflexes such as walking, etc. It was mentioned earlier that figure 10.3 represents a somewhat simplified picture of the wiring of the spinal cord. There are interconnections present between the simple reflexes, these may represent the evolved remnants of the nerve net and they control the relationship (and therefore the co-ordination) between reflexes.

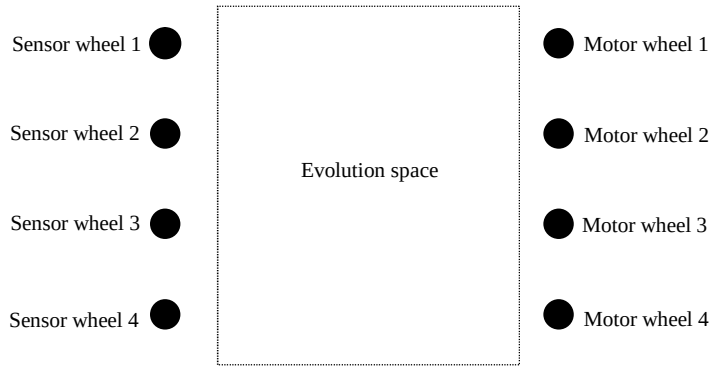
Several animals have more than one nerve net [22], each of which is unconnected. This may be the origin of the different co-ordinated reflex groups (walking, as opposed to flying in a bird for example). Each of these nets may have no contact with each other, but they are both controlled by the brain. It should be noted that although the brain does not co-ordinate these medium level actions, in many cases it can be shown to provide a ‘clock’ or regular signal which controls their speed [23].

From the above observations we can now build a picture of an artificial nervous system.

10.5.2 A model of the biology.

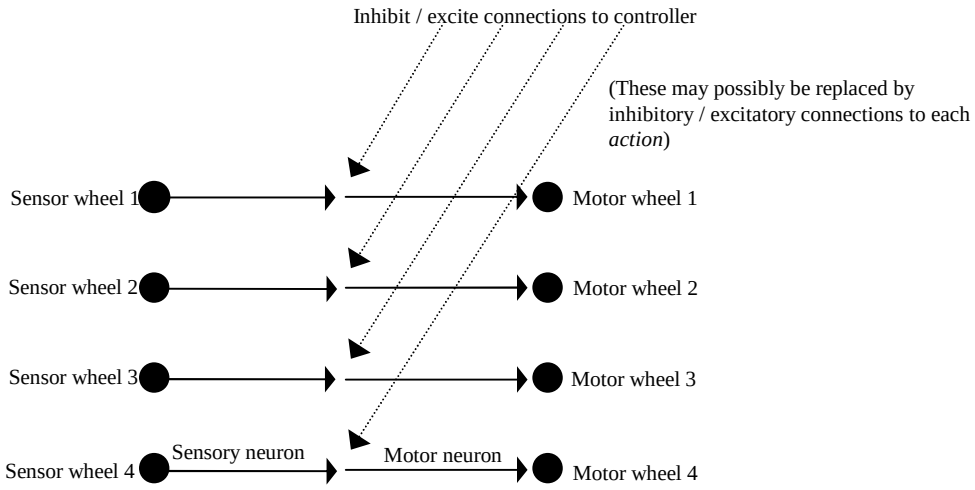
Consider how knowledge of the organisation of the biological system, outlined above, may be put to use in creating an artificial nervous system. The reflexes consist, at the simplest level, of a pair of neurons (sensory and motor). Let us suppose that we are designing the control system for a simple wheeled robot. The motor outputs, in this case, activate and control the motors which turn the wheels. The sensory inputs might be shaft encoders for each wheel. If the network is to be evolved using one of the techniques which has been outlined in the thesis, then a representation of the system such as that shown in figure 10.4 might be used.

Figure 10.4 Artificial representation of the biological model



The evolution space is the area in which the network must develop. The simplest approximation are reflexes represented by sensory-motor connections as shown in figure 10.5. Once these basic reflexes are in place (that is the wheels will turn at the correct speed when stimulated) then the co-ordinating network (equivalent to the walking or swimming response - a different network for each action) must be created in the evolution space using Genetic or Evolutionary methods. This represents the sideways connections between the reflexes (that is, the interconnection between them). Because the reflexes are such simple responses, it may be convenient to replace them with Fuzzy Logic control [24] (each reflex may be viewed as a simple control loop). Indeed the actual circuitry for control may not need to be neural in origin at all [25] [26] (see also section 9.3.1). For a biological illustration of co-ordinating wiring see reference [27].

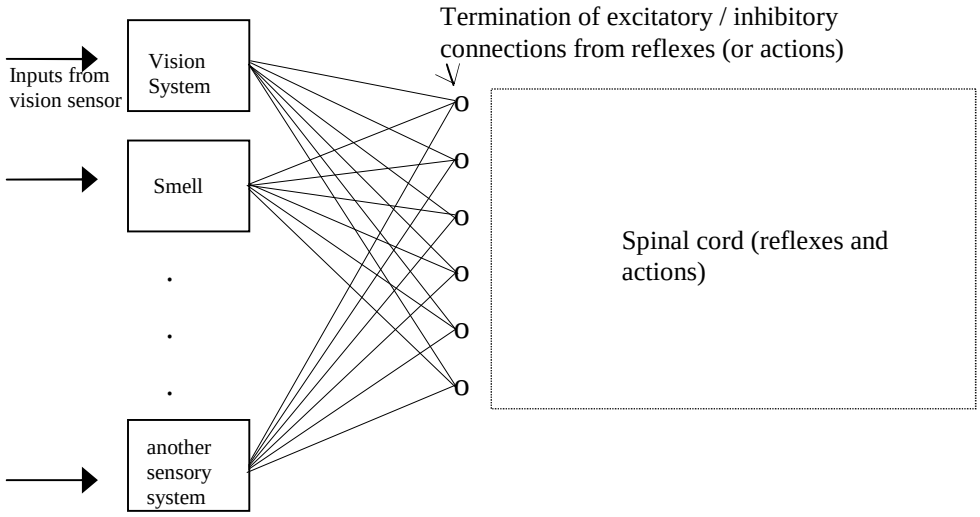
Figure 10.5, Basic reflex action.



10.5.3 Brain and higher functions.

There is still the problem of 'intelligence' to be considered and this brings us back to the inability of the network to process information in an unconstrained environment. First, consider the termination of the excitatory and inhibitory connections to the reflexes (or actions). In the biological sense these terminate in the lower brain [28] and are controlled by the sensory information from the sense organs of the animal. A model of this is shown in figure 10.6.

Figure 10.6, Model of higher functions.



One advantage of this is immediately apparent: the network has a modular structure. It does not have to be designed into the system; the modular problem does not even arise. This is not surprising; when engineers design an aeroplane, they do not consider it as a single structure and then try to devise clever and artificial ways to modularise its construction. It is a group of modules which function together to produce a working machine, and so it is with the BNN also. In simple animals the sensory information is processed in separate ganglions, each of which presumably evolved independently for its function. As noted earlier, the brain and nervous system of all animals above the level of hydra and its kin is highly modular in structure [29]. The question still remains however: how do we evolve the higher functions such as vision.

10.5.4 Input Deconstraint - a possible route to evolving high level inputs.

How did nature solve the problem of evolving the higher functions described in the previous section? The answer, of course, is by using evolution. The first eye may have been nothing more that a light sensitive spot. This is enough to tell the organism about night and day and possibly indicate the presence of a possible predator. A similar light sensitive organ exists today in some fishes [32] and is sometimes referred to as a 'third eye'.

Should this facility to react to light prove biologically useful, (as it surely must have done) then it might evolve into an organ capable of judging the direction of a light source. This might happen due to the appearance of more light sensitive cells. After aeons of becoming more sensitive and useful it will eventually resemble an eye. For an illustration of this process see reference [33].

As the sensor evolves (that is, the eye) so its neural network will also evolve. The purpose of the sensor is to cause a reaction in the presence of some stimulus: for example, escape from a predator, move towards food or location of a mate. So the neural network transforms the sensory input into commands to the spinal cord using the excitatory / inhibitory connections shown in figure 10.5.

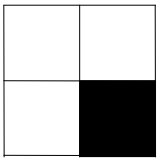
How might such a system be used to evolve the network for the higher functions. Let us take vision as an example. Assume that the primitive eye (nothing more than a sensitive spot) can only detect the difference between light and dark as shown in figure 10.7.

Figure 10.7. A view from the most primitive 'eye'.

We can train the network to respond to this basic system with a basic response: for example, if darkness falls quickly run away (as a predator is leaning over you) or if darkness falls slowly, go to sleep, it is night time. This neural reaction is a basic brain function (think of how a person jumps back when someone frightens him / her);

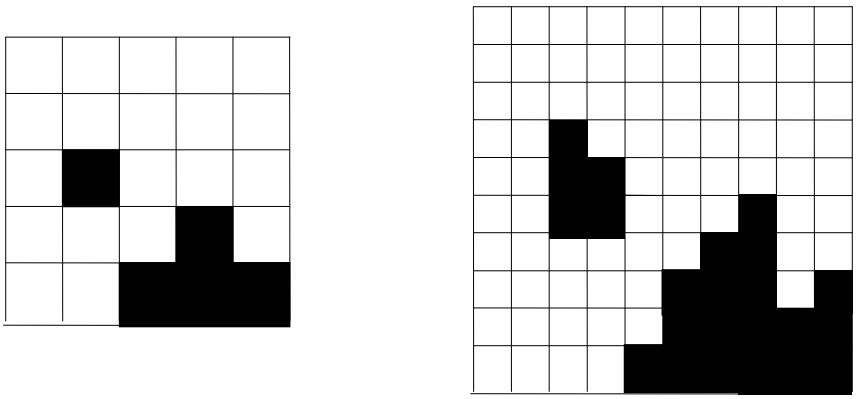
compare it also with McLean's model in section 4.8.1. As the eye evolves, it gains more resolution, figure 10.8.

Figure 10.8, First de-constraint of data.



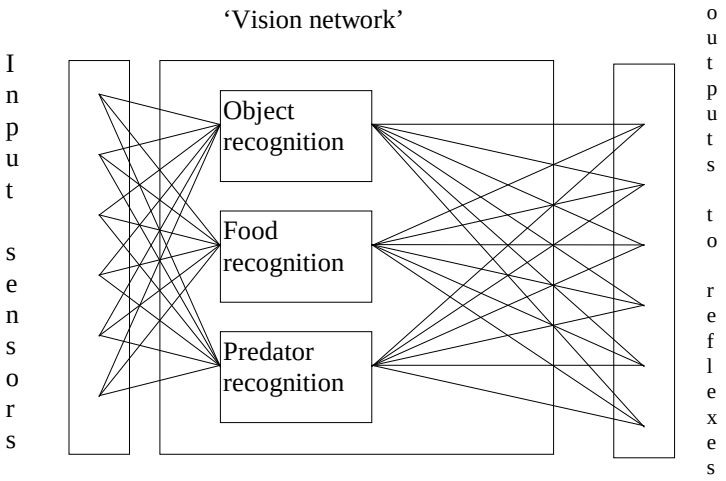
Now more information is available and the network can be trained to recognise large stationary objects, which it can avoid. Further data de-constraint is shown in figure 10.9 below.

Figure 10.9, Further relaxation of data set.



As each stage in process happens, the animat obtains another level of sophistication. We may evolve each capability by adding another neural net module to the network. See the paper in appendix 1 for further description. The general model is then as shown in figure 10.10.

Figure 10.10, Modular model of vision

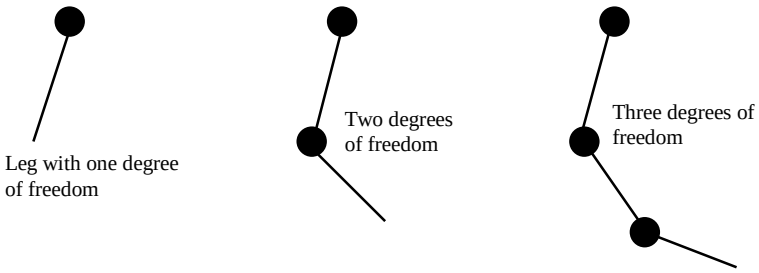


A module comes into being because it helps the system stay alive and prosper. In more complex systems, a further 'control' network will be required to decide which of these sensory networks has priority when it comes to stimulating the reflex actions (the origin of consciousness ?). It is interesting to conjecture that if the senses had been distributed throughout the body rather than in one region, the head, whether a brain and therefore possibly consciousness, would have evolved at all (because the sensory ganglions would never have grown together in the same place).

10.5.5 Output deconstraint - the motor system.

Exactly the same process may be followed to evolve a complex actuator response in the animat. A lower biological form may have only one degree of freedom in its limbs. However, by evolving the network in stages and deconstraining the actuator one stage at a time, a more complex network may be able to be formed. This is shown in figure 10.11.

Figure 10.11, Actuator relaxation.



The method outlined above is not the only way to evolve this type of network. For example, it was noted earlier that Hydra has a single homogeneous nerve net. We could start with a structure like this and allow the 'body' of the net to develop projections (limbs). This will tend to isolate parts of the network which may then be developed into reflexes. Less isolated parts of the network can then form the actions. Other schemes can also be thought up. The one presented above, however, has the benefit of simplicity.

10.6 Summary

Although this section may seem slightly out of step with the previous chapters, it is included for a particular reason. One of the original objectives of the thesis was to investigate the problem of modular evolution, and although this proved too difficult to achieve within the project, it is an important objective of further work. The purpose of this chapter is to outline the form that such further work might take and this theme is taken up in chapters 12 and 13.

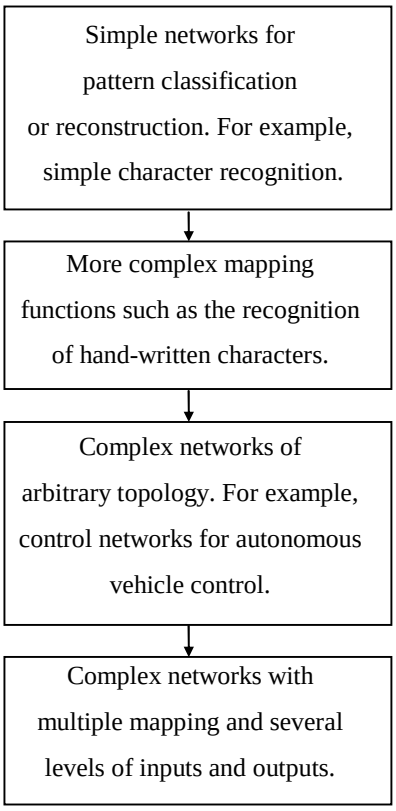
Chapter 11

Applications of Evolutionary Techniques

11.1 Introduction to chapter

The technique investigated in this project is not a universal solution to ANN optimisation problems. In the sections below, its applications are considered and the algorithm's good (and bad) points are discussed. The application areas are ordered in terms of complexity; ranging from simple classification problems, to the creation of complex neural networks for use in autonomous vehicle control systems. Figure 11.1 shows a general outline of network task complexity.

Figure 11.1, A hierarchy of task complexity.



In the following sections, each of these systems will be considered one by one.

11.2 Simple pattern recognition systems

Examples of simple pattern recognition systems include identification of computer generated characters [1], as shown in examples 7.6 and 8.1. These systems are usually fairly simple to set up. With experience, an ANN designer can make a guess at the size of network which will do the job. The less experienced designer will find it more of a problem. In any case, the network thus arrived at, will usually be less than optimal.

The systems illustrated in chapter 6, which grow networks by adding units, will solve this problem. The Genetic Algorithm is, however, not suitable, because it is much less efficient than these other methods in simple optimisation tasks such as this [2]. The technique outlined in the thesis is suitable and competitive - given a small strategy list.

So, for this type of problem, several techniques, including the one outlined here, are suitable.

11.3 More complex mapping systems

An example of this type of network is that for hand-written character recognition or simple control systems. The MATLAB Neural Nets toolbox contains excellent model examples of this and other applications of simple ANN problems [3]. This level of problem is typically shown by examples 8.5 and 8.7. These are more complex tasks than those mentioned in the previous section. The solution surface is multi-dimensional; that is, the solution may be several layers and few neurons in each layer, or many neurons with low connectivity. The networks illustrated in section 6.3.3 do not have this aspect to their dynamics. In some cases, even quite simple tasks can only be solved by using multi-dimensional growth. This is illustrated by example 7.1, where adding more units will not solve the problem; extra layers are also required.

It is in this area that incremental evolution excels. The networks are too complex for a simple growth strategy to optimise. Just adding neurons may not help; the network may

require more inputs and / or layers to solve the problem. However, the solution space is continuous (although multidimensional) and so a pseudo-random search such as the Genetic Algorithm or a directed random search such as Simulated Annealing is not a suitable technique for efficiency reasons [2].Incremental Evolution, however, works well in these situations, able to handle several growth strategies (and therefore move in several dimensions in solution space) but not randomly like the GA.

Another way of looking at the reason why the method works well in these problems is that the algorithm contains the 'big three' growth strategies. Each of these allow the network to move into a new mapping domain; they are:

- The addition of neurons. This expands the complexity of the separator surface.
- The addition of layers. Adding layers allows the separating surface to fold back on itself.
- The addition of feedback. This adds a sequential aspect to the network behaviour.

The other strategies are 'icing on the cake' - they optimise a network which is already grown for the task. It is these three which are essential. None of the other incremental strategies reviewed in section 9.2 have all three available to them. One can add, that if the network is to perform temporal functions, then delay and perhaps memory can also be added (see section 12.5.3).

11.4 Networks of arbitrary topology

Good examples of this class of network include complex control networks for robotic applications. These generally have a complex topology, which is due to the dynamics of a non-continuous solution space. These networks are rather like a digital electronics controller; their behaviour is the result of complex component-system dynamics [4]. The Embryology is not suitable in this case. A discrete solution space means that there is no incremental error gradient to follow [5]. The GA and Statistical methods are the best

solutions in this problem. Because the technique outlined is not suitable for this type of task, an example is not given in the text, but typical network tasks are shown in Thompson's papers [6].

11.5 Complex multilevel and modular networks

One of the main thrusts in ANN research is towards the design of practical modular ANNs. The reason for this, is that these systems are a possible solution to the problem of tackling unconstrained data sets. The importance of this question, and some possible answers to it, are examined in much more depth in the chapter 10. Generally speaking, this field is the domain of the GA [6]. The networks required are complex and represent a discontinuous data space. As such they are unsuitable for the EA. It may be that some of the modules, or individual networks, within a modular system might lend themselves to incremental construction, but the system in general will probably not. GAs as they currently stand, are not suitable for use in these systems either. As noted in chapter 10, the solution will possibly involve both the evolution of network and also of sensors and actuators (effectively limbs). Therefore, a more complex multidimensional GA is required.

11.6 Summary

The sections above, may be summarised as follows:

- Simple mapping systems (for example, pattern recognition) are suitable for the technique outlined here. Alternatively, the simple growing networks outlined in section 6.3.3 may be used.
- More complex mapping systems (for example, waveform prediction or simple control networks) are the main area of application for the technique. The solution space represented by these is too complex for the simple growth algorithm, but the GA is also an inefficient method (although it can solve the problem).
- The most complex networks usually have a discontinuous search space and are not suitable for solution by the method. This is the area in which Genetic Algorithms excel.

Chapter 12
Further work

12.1 Introduction to chapter

Chapters 9 and especially 10 both give suggestions for further work. The purpose of this chapter is to point out other areas, brought to light by the project, into which further research can be done. In the sections below, the suggestions from the previous chapters will be summarised and further areas will be discussed, including combinations of this method and others. Finally, there is a large section on neural functionality.

The section on neural functionality is included because, during the course of this research program, ANN models have proved unwieldy to handle (this is particularly true of digital implementations). Also, impressive 'intelligent' systems which do not use ANNs have been demonstrated; for example, the Honda Walking Robot [1] and Fuzzy logic systems [2]. This final section addresses whether ANNs should be pursued at all or if other technologies hold the key to intelligent systems (which, in the end, is what evolutionary ANNs are all about).

12.2 Ideas for research into unconstrained problems

Chapter 10 gives detailed suggestions for further research in this area. The main areas suggested were:

- Mathematical models with built-in non-homogeneous structures (section 10.5).

- The idea that biological evolution of a non-specific network is driven by sensory input and that animats should play an important role in development of unconstrained systems (due to their ability to experience the environment).

- The concept of an evolution space in which the network must grow.

- The importance of sensory - actuator connections and mappings (section 10.6).

- The central idea that modularity will arise out of a systems approach to networks.

- Input - output deconstraint to simulate the evolution of sensors and actuators in organisms.

12.3 Combinations of this method and others

While running the programs which generated the test results in chapter 8, it became apparent that the embryological method could, in the long run, be improved by combining it with other methods. Several examples of this are given below.

1. Combination with statistical methods.

It has been pointed out (section 8.5.2) that when the growth algorithms are run with different orders of strategy, they produce different results (sometimes getting stuck in a local minima). One way to avoid this is to introduce a randomising element into the algorithm. For example, it can be combined it with a statistical technique such as simulated annealing. This would allow the connections to become randomised occasionally and so shake the growth algorithm out of any local minima in which it could found itself.

2. GAs

In a similar vein, part of the network design (for example, number of neurons and layers) can be designed using EAs and the final detail of the connection pattern using GAs. This would have three beneficial effects. Firstly, it would add a random element to the network, similar to that listed above. Secondly, it would speed up the design process relative to a pure GA design. Thirdly, it would optimise the network at the connection level. The EA is best at optimising the network at the unit / layer level (because it cannot control the connection pattern as accurately as a GA).

Another interesting project would be to allow the evolution strategies themselves to evolve as the system developed. In other words, strategies with little success drop out

and those which show major success are used more and can change their parameters to become more efficient.

3. Combination with ART

The ART network (see sections 3.7.2 and 6.3.2) has several unique aspects to it. No other network has reached its level of sophistication. In particular, the vigilance function which allows a neuron to be trained or a new unit assigned to a class, is extremely useful and could be built into other networks. This aspect might benefit from:

a. being simplified
b. being built into the neurons themselves (to avoid the need for a separate subsystem).

12.4 Other related ideas for further work.

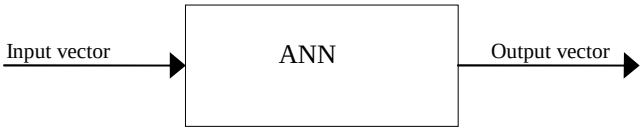
When doing a literature search for material to base this thesis on, it was evident that there were some biological areas into which further work could be done. In particular, research on the early evolution of the nervous system in simple animals would be most welcome. This may be more possible now than in the past due to the emergence of fossil deposits in which soft tissue has been well preserved [3]. Also, work on the origin of neurons in the ectoderm in evolutionary terms would provide background into the functionality of neurons and their relationship with the outside world and might lead to insights into evolutionary methods for interfacing and developing sensors.

The evolution of sensors and actuators was outlined in section 10.5.4. This topic is important because it allows the whole system to evolve naturally and as pointed out previously, the sensors and actuators cannot be viewed as separate from the rest of the nervous system. Along the same lines, some workers have written about evolving the body of the animat along with its nervous system [4].

12.5 The functionality of Neural Systems.

This section considers a point for further research: '*What is an ANN, why is it different from other systems and do we really need ANNs at all*'. The Artificial Neural Network has a set of inputs and a set of outputs as shown in figure 12.1 below.

Figure 12.1 An Artificial Neural Network as a system with inputs and outputs.



Many other electronic systems also have these features. So, what is the difference between an ANN and (say) a fuzzy logic system? After all, both have the function of mapping a continuous input onto a continuous output. To take this argument a stage further, let us compare ANNs with several similar systems - Traditional Control, Digital Electronics and Fuzzy Logic. This is shown in figure 12.2 below.

Figure 12.2 Comparison of ANNs, Fuzzy logic, Digital electronics and Traditional control systems technologies.

	Continuous outputs	Binary outputs	Can learn	Fixed Functionality	Parallel structure
ANNs	□	1	□	x	□
Digital electronics	x	□	x	□	□
Control system	□	4	2	□	x
Fuzzy logic	□	x	3	□	x

Notes:
1. Some simple ANNs do have binary outputs
2. Adaptive control systems are available
3. Adaptive Fuzzy Logic and neuro-fuzzy systems can learn
4. Some simple control systems (such as PLCs) have binary outputs

The question of ANN functionality is of critical importance. What, if anything, can an ANN do, that other systems cannot. If the answer turns out to be nothing, then why are we using ANNs at all (they are difficult to handle, especially in hardware terms). Surely man could devise some better building block for intelligent systems. In other

words, is the neuron simply nature's rather crude logic gate? In this view it is the wiring of the system which is important rather than the devices themselves.
An illustration of this point comes from the important papers by Adrian Thompson of the University of Sussex [5]. In these, Thompson illustrates how digital circuitry using standard gates can be evolved using a Genetic Algorithm in a Field Programmable Gate Array (FPGA). The circuitry can be developed to fulfil complex tasks which are akin to Neural Nets type problems (this is an example of the GA as a training methods, see section 9.3.1).

A full discussion of these points is important but somewhat periphery to the main objectives of the project and so has been included in appendix 7. However, this question of functionality is an important area for further work.

Chapter 13
Conclusions

13.1 Introduction to chapter

The primary purpose of this, the final chapter of the thesis, is to summarise the successes and failures of the project. This means a critical re-evaluation of the project objectives in terms of what has been achieved.

13.2 The project objectives revisited

The objectives, as originally stated at the start of the project, were:

1. Study of relevant literature.
2. Design and implementation of evolutionary algorithm.
3. Combination of evolutionary algorithm and neural network.
4. Comparison of generated network with standard network.
5. Study of learning algorithms related to this approach.
6. Application to non-specific problems.
7. Investigation of the applications of such networks.

These objectives were set out in the original research degree application form in 1994. As the project progressed and knowledge was gained, some of the objectives assumed a new significance and others receded somewhat from their original importance. This is to be expected, since no work had previously been done in the area and the project originated from nothing more than an idea. In fact, when the project started, the author was unaware of any other networks which grew. The idea of the embryology had originally occurred while reading Dawkins book.

Let us consider the objectives in terms of what has been achieved in the project.

1. Study of relevant literature

An extensive literature search was started at the beginning of the research. This continued intensively for the first four months of the project and thereafter at a lower level all the way through until the end. The author has a high degree of confidence that all important work has been assessed. The results of this are given, mainly in chapter 6, but also (to a lesser extent) in chapters 5 and 7.

2, 3. The design and implementation of the evolutionary algorithm and its application to neural networks

In the original specification, these were two objectives. However, in practice, they naturally became part of the same computer program. This represents the central theme of the project and its main objectives. Happily, these were achieved without any compromise and therefore the project as a whole may be regarded as a success. Chapters 5 and 7 outline the preparation work for these objectives and the results in chapter 8 represent their achievement.

4. Comparison of generated network and standard network

All the experiments in chapter 8 are derived from examples in the MATLAB toolbox and elsewhere. This means that ‘standard’ or recommended networks were given in the literature for most of the problems. The networks which were grown, are compared with these standard networks in chapter 8. In all the examples tested, the grown network performed better than the recommended network.

5. Study of learning algorithms related to the approach

This is one of the objectives which assumed less importance as the research progressed. At the start of the project, it was assumed that the growth algorithm itself might yield some learning mechanism related to its structure. However, this turned out not to the be the case. Chapter 9 outlines the relationship between the method and growth algorithms. On a related note, one original contribution to the subject of network learning was the invention of the Taguchi learning algorithm.

6. Optimisation of the network by restricted growth

This turned out to be a ‘red herring’. It was thought that a network could be optimised by growing that part of it which showed the most activity. In practice, however, it turned out to be much easier to grow the network from scratch. Growth strategies which optimise the network were then built into it. The original idea was basically the same as *sideways growth* which became one of the standard growth algorithms anyway.

7. Application of network to non-specific problems

Even at the start of the project, it was realised that this point represented a major goal in the field. However, it is one which is fraught with difficulty and subtlety. Many of the greatest minds in Neuro-computing are working on the problem. The Embryology is, in any case, unsuitable for tackling the problem, as it will only operate in a continuous solution space. This aspect must be tackled using Genetic Algorithms or some other technique. The author’s opinions and ideas on the subject are outlined in chapter 10.

There is probably several years of further research associated with solving this conundrum. It is hoped that the next project starting in the School of Electronic and Electrical Engineering can address certain aspects of this problem.

8. Investigation of the applications of such networks

Several areas within the thesis may be regarded as an investigation of the applications of the technique. The examples in chapter 8 have been chosen to give a number of disparate fields of application. Chapter 7 also gives some hints about the application areas of the system. The conclusions of these investigations are given in chapter 11.

In summary, all the objectives of the project were achieved, with the sole exception of the application of the network to non-specific problems. The technique was found to be not suitable for this particular purpose.

13.3 Original contributions to the art

When assessing any research project, an inevitable question concerns the contribution to new knowledge made by the researchers and their work. This project has several original aspects to it, all of which are a product of the work. These are:

- A unique and flexible method of controlling growth - the single string method itself.
- The consideration and classification of growth strategies for ANNs.
- The combination of the above into a comprehensive network optimisation algorithm.

Quite apart from these aspects, the combination of which is the results given in chapter 8, there are several over aspects of theory and speculation which are also original work. These include:

- A new view of routes towards modular networks based on animat evolution.
- A reconsideration of the role of unit functionality in the network.
- The use of the Taguchi method to train networks.
- A consideration of the application of these networks and their relationship to other evolutionary defined networks.

13.4 Summary of suggestions for further research

One very important aspect of the work undertaken on the project, is that, having considered all the literature in detail, the important work still to be done is now obvious. This falls into two categories. The first concerns the research relating to this technique itself. Work to be done on this aspect is in two areas:

- Work to produce a clear and practical way of implementing the algorithm, so that it may be easily used.
- A consideration of the combinations of this technique and others. For example, combination with GAs, Statistical methods or an ART type vigilance function.

The other type of further work is much more wide ranging and effects not only this technique but all evolutionary or genetic techniques and indeed the very future of AI itself.

- The definition of modular neural networks.
- The definition of unit functionality.

13.5 Concluding remarks

The project itself has been very successful. It has achieved its aims better than was thought possible.

The technique is a powerful and useful method for finding the best network in a whole range of important problems. Its configuration means that it does this very flexibly and it is possible to optimise the network for problems which require the simplest possible system or, for example, the most noise tolerant. The configuration of the gene means that the system may be driven incrementally or by a random number generator.

This work now joins a body of other research including Genetic Algorithms, Evolutionary Techniques and Simulated Annealing, all of which are capable of defining networks. These different techniques cover the whole range of problem types. One can consider that the network definition problem is essentially solved.

Appendix 1

Papers produced during research.

Papers produced during the research program. These include internal and unpublished papers and provide a ‘snap shot’ of the thinking and progress of the project at the time.

Note: Due to technical issues, the papers in this appendix are not included here. They may be seen in the paper copy of this thesis in the Robert Gordon University or British Library (or obtained from the publisher if available).

Paper 1 A single string evolutionary approach to neural network synthesis

This is an internal paper (not published) which shows the original thinking at the very beginning of the project. Many of the thoughts outlined here changed as experimental work began and new and better ideas emerged with regard to the algorithm and its formulation.

Paper 2 Training artificial neural networks using Taguchi methods

This learning method came from research into evolutionary neural nets as part of the project. This is the first time it has been used to train neural nets.

Published in AI Review (currently awaiting publication)

Paper 3 Growing neural networks using a single string evolutionary algorithm.

This paper is a brief summary of the research. It illustrates all the growth algorithms and gives a simple application example.

Published (in slightly modified form) in ICANNGA ’97 (conference) proceedings.
A more extensive version (including major results) will also be submitted to a refereed journal.

Paper 4 Using evolutionary neural networks to generate general classifiers

General (and somewhat speculative) paper on co evolving sensor output and neural network. Internal paper not published.

Paper 5 A framework for evolution of an animat nervous system

Outlines the basic ideas on modularity of networks and how this might be achieved. This paper has been published in two different forms and serves as the basis for another PhD in area.

Published in EUREL (conference) proceedings 1998
Also In Electronics World (in modified form) Nov 1998.

Paper 6 The functionality of neural networks and related technologies.

Outlines some ideas on neural unit and network functionality.
Published in Electronics World June 1997 (in a modified form).

Appendix 2

Methods used to obtain results.

MATLAB® and the MATLAB Neural Networks Toolbox

MATLAB is a mathematical modelling package. It is very powerful and has many applications including neural networks. The addition of the Neural Nets Toolbox makes it even more useful as a tool. One of the reasons why MATLAB is so powerful is that it is primarily a matrix processor and almost all neural net functions can be regarded as matrix operations. To illustrate this, consider the following example.

We can set up a matrix of weights for a network:

a = [0.3 0.5 0.2; 0.7 0.4 0.8; 0.9 0.6 0.7]

The output from MATLAB:

a =

0.3000	*0.5000*	*0.2000*
0.7000	*0.4000*	*0.8000*
0.9000	*0.6000*	*0.7000*

Next, set up an input vector:

b = [0.5; 0.7; 0.2]

MATLAB again confirms

b =

0.5000
0.7000
0.2000

Now doing a matrix multiplication of these two elements is exactly equivalent to calculating their weighted sum (see equation 3.4):

*c = a * b*

The output from MATLAB:

c =

0.5400

0.7900

1.010

which is the correct answer. A sigmoidal squashing function may then be applied:

*d = (exp (c * (-1)) + 1) ^ (-1)*

MATLAB:

d =

0.6318

0.6878

0.7330

which are the three outputs of the network. This example has only shown the forward pass on the network. Training may also be accomplished in the same simple manner. Using this, many complex problems can be dealt with easily. MATLAB contains a modelling language and the operations above can be coded in about three lines of script, executed automatically and saved until required.

At the start of the project networks, were coded in C or C++ . However, the power of MATLAB soon became apparent and most of the networks were coded in the way.

For further information see *The student edition of MATLAB, Prentice hall, 1995. (ISBN 0-13-184979-4)*

Much use was also made of the MATLAB Neural Networks Toolbox. It is possible to make any network using basic MATLAB, but in complex cases this can be tedious. The Toolbox contains many additional commands which can be used to simulate neural nets. For example, in Back Propagation, there are three high level commands to train networks available:

trainbp	-Basic BP training
trainbpx	-Fast adaptive BP with momentom
trainlm	-Train with Levenberg - Marquardt optimisation

These, in turn are composed of lower level functions (to which the user has access). Altogether the toolbox gives approximately 200 extra MATLAB high level commands. These will train everything from Perceptrons to complex Elman networks and Self Organising Maps. The facilities also allow graphs to be printed out, two examples of which are illustrated overleaf.

Using the basic MATLAB facilities together with the toolbox allows any conceivable network to be trained and therefore it was logical that this method was the easiest to use for powerful yet simple experiments.

More information can be obtained from the *MATLAB Neural Network toolbox manual, H Demuth and M Beale, The MATH WORKS inc, 1994.*

MATLAB graphs

These two graphs show the training details of a FF 8,26 network training on a noisy character set.

Note: graphs unavailable in PDF version

Coding in C and C++

The C and C++ languages were also used for coding networks, particularly at the start of the project, before discovering the full power of MATLAB. They were also used in the event of having a problem which proved easier to implement in an algorithmic language. Weights and inputs were generally coded as arrays and ‘brute force’ methods were used to implement these networks (the object of the exercise was to obtain results and not to produce elegant programs). Programs were designed which used standard functions - these could then be reused in other programs. Shown below is the example of the modules in a BP program. The diagram shows how the weights are encoded in arrays. Some prototyping was also done using BASIC, but usually only to get the algorithm working. The final version was usually implemented using one of the other methods.

Appendix 3

Growth algorithms.

In the following pages are listed the growth algorithms for the strategies explained in the text. These were initially coded in C and C++ and the resulting network structure stored in an array using an elaborate coding scheme. Although this system is neat, the disadvantage of it is complexity and speed. A simpler way to use these algorithms is as MATLAB macros. In this way, the appropriate algorithms for the problem can be cut and pasted into a file and used.

There are several growth strategies listed in chapter 7 which do not have algorithms listed. There are several reasons for this:

- MATLAB already provides an algorithm (for example, in the case of bias).
- The strategy is not used with ‘normal’ networks (in the case of delay lines - although this algorithm is trivial to code).
- The strategies were not used because its form was not fully resolved (only one strategy -that of localising connections -falls into this category).

The algorithms are given in the form of pseudo-code so that they may be coded in any convenient language. The examples given below are general. In several examples they are altered to run in slightly different ways (for example, the addition of random rather than incremental operators, see section 8.5.4). For this reason these strategies should not be viewed as absolute; they may be altered to suit the problem at hand.

To show two different approaches to the algorithms, consider the procedure below. This changes the number of neurons in the network. It is normally run as shown below, once in each cycle of the program.

Procedure: Change number of neurons

```
Done_addition_flag = 0
Result = good

If (neuron number > lower limit)
        subtract two neurons one at each end of structure
        do test
        if result = good
                Done_addition_flag = 1
        if result = bad
                add two neurons

If (neuron number < upper limit and Done_addition _flag =0)
        add two neurons
        Done_addition_flag = 1
        do test
        if result = bad
                subtract two neurons
                Done_addition_flag = 0
```

An alternative approach (which also works but gives a different result) is shown below. In this case if the addition (or subtraction) of neurons is successful, the routine keeps going with the strategy

Procedure: Change number of neurons

```
Done_addition_flag = 0
Result = good

Do while (result = good and Neuron number > lower limit)
        subtract two new neurons one at each end of structure
        do test
        if result = good
                Done_addition_flag = 1
        if result = bad
                add two neurons

if (Done_addition_flag = 0)
        Do while (result = good and Neuron number < upper limit)
                add two neurons
                Done_addition_flag = 1
                do test
                if result = bad
                        subtract two neurons
                        Done_addition_flag = 0
```

This routine prunes or adds connections to the network. This version tests the effect of removing each connection in turn. The other version used removes connection with the least

activity and allows the network error to increase by a small amount (the upper limit is set as a target).

Procedure: change connections

```
Done_pruning_flag =0

A: if connections are saturated
        subtract one connection
        Done_pruning_flag = 1
        do test
        if result = bad
                add one connection
                Done_pruning _flag = 0

B else if connections are empty and Done_pruning_flag =0
        add one connection
        do test
        Done_pruning_flag = 1
        if result = bad
                subtract one connection
                Done_pruning_flag = 0

else if not A and not B
        subtract one connection
        do test
        if result = good
                Done_pruning_flag = 1
        if result = bad
                add one connection

    If (Done_pruning_flag = 0)
        add one connection
        do test
        Done_pruning_flag = 1
        if result = bad
                subtract one connection
                Done_pruning_flag = 0
```

This routine adds sideways growth to the network.

Procedure: sideways growth.

```
Done_sideways = 0

add one neuron to left and fully connect to layers above and below.
test result.
if result = good
        Done_sideways = 1
if result = bad
        Done_sideways = 0
        delete neuron

add one neuron to right and fully connect above and below.
test network
if result = good
        Done_sideways = 1
if result = bad
        Done_sideways = 0
        delete neuron
```

Adding layers is simple. Several versions of this algorithm were tried. One starts by adding two neurons in the new layer. The version below adds a new layer with the same number neurons as the layer above. Layers are usually added as a last resort and not deleted.

Procedure: Add a layer

Add a layer on to the network with same number of neurons as the layer above (fully connected).

```
do test
if result = bad
        delete layer
```

This next procedure prunes connections asymmetrically from the network.

Procedure: asymmetry

```
first time round n = 1
left_flag = 0
right_flag = 0

if right_flag = 0
delete connection n from the left field
left_flag = 1
do test
n = n + 1
if result = bad
        restore connection
        left flag = 0
        n = n - 1

if left_flag = 0
```

(listing continued from previous page)

```
delete connection n form the right field
right_flag =1
do test
n = n + 1
        if result = bad
        restore connection
        right_flag = 0
        n = n -1
```

There are two other routines used: one adds feedback connections and the other skips layers. However, both of these take the form of the routine listed above. For example, the feedback routine starts on the left of the network and adds feedback to the first neuron of the next layer. If this is successful, the feedback is retained; otherwise, it is deleted. A counter (n) is incremented and on the next loop a connection is initiated to the second neuron and so on. When all the connections to the first neuron of the next layer have been explored then the routine increments a second counter (n2) and starts the same process from the second neuron in the first layer. The routine for skipping connections works in exactly the same way except that the connections are forward rather than feedback.

It should be obvious from the above that all the growth algorithms take a similar form. This is shown below (the lines shown in italics are only used in routines which have an incremental component):

Procedure: general

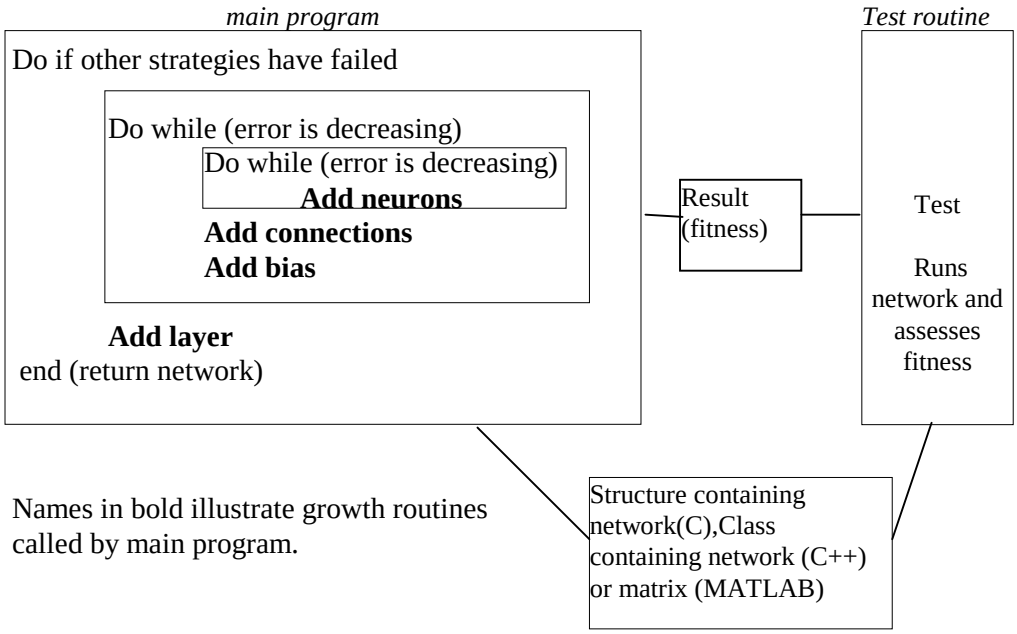
```
Flag = 0
counter = 1

Implement strategy on item (counter)
flag = 1
Do test
counter = counter +1
If result = bad
        Restore original state of network
        flag = 0
        counter = counter - 1
```

To show how these routines are integrated together, consider a growth program containing the following strategies:

- Add neurons
- Add connections
- Add bias
- Add a layer

In this scheme adding neurons adds capability to the network. Connections and bias optimise the network and adding a layer is a last resort. This is illustrated in the diagram below.



Names in bold illustrate growth routines called by main program.

Appendix 4

Test data.

<u>Standard letters</u> - Character recognition from the MATLAB Neural Networks toolbox.

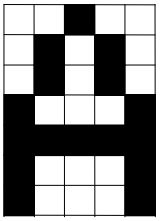
MATLAB contains a training set of the 26 standard letters of the alphabet. These can be set up in a matrix called alphabet. A corresponding set of targets is available in the matrix targets. The letters are in a 5 by 7 pixel grid format; each one represents a black pixel and each zero a white. For example, the letter A is given as (spaces have been added for clarity):

00100 01010 01010 10001 11111 10001 10001

Which plotted on a grid looks like:

0 0 1 0 0
0 1 0 1 0
0 1 0 1 0
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1

or to make it clearer



The targets are twenty five 0s and one 1. The position of the 1 corresponds with the position of the target letter in the alphabet. So the first three letters of the alphabet have targets:

A: 10000000000000000000000000
B: 01000000000000000000000000
C: 00100000000000000000000000

and so on. A set of numbers has also been coded in the same format.

The whole training set is shown overleaf.

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

Target Vectors

0 0 1 0 0 0 1 0 1 0 0 1 0 1 0 1 0 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 1 A
1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0 B
0 1 1 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 1 1 0 C
1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0 D
1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 0 0 1 0 0 0 0 1 1 1 1 1 E
1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 F
0 1 1 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0 0 1 1 1 0 0 0 0 1 0 1 1 1 0 G
1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 H
0 1 1 1 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 1 0 I
1 1 1 1 1 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 1 0 1 0 0 0 1 0 0 0 J
1 0 0 0 1 1 0 0 1 0 1 0 1 0 0 1 1 0 0 0 1 0 1 0 0 1 0 0 1 0 1 0 0 0 0 1 K
1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 1 1 1 L
1 0 0 0 1 1 1 0 1 1 1 0 1 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 M
1 0 0 0 1 1 1 0 0 1 1 1 0 0 1 1 0 1 0 1 0 1 1 0 0 1 1 1 0 0 1 1 1 0 0 0 1 N
0 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 1 1 1 0 O
1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 P
0 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 1 0 1 1 0 0 1 0 0 1 1 0 1 Q
1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0 1 0 1 0 0 1 0 0 1 0 1 0 0 0 1 R
0 1 1 1 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1 0 1 1 1 0 S
1 1 1 1 1 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 T
0 1 1 1 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1 0 1 1 1 0 U
1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0 0 1 0 0 V
1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 1 0 1 1 1 0 1 1 1 0 0 0 1 W
1 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 1 1 0 0 0 1 X
1 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 Y
1 1 1 1 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1 1 1 1 Z

Input vectors

See pages 11.40 - 11.46 in the *MATLAB Neural Network toolbox manual, H Demuth and M Beale, The MATH WORKS inc, 1994* for further details.

<u>Noisy Letters</u> - from the MATLAB Neural Networks toolbox.

MATLAB also allows noise to be added to characters illustrated in the section above. The result may be used to train networks or to test their performance in the presence of noise. This is especially useful when producing verification sets. Noise can be added in any range, for example adding a random number (n_r) in the range $-0.05 < n_r < 0.05$ adds 10 % noise to the character. Adding n_r in the range $-0.5 < n_r < 0.5$ adds 100% noise. The target vectors stay the same as outlined above. As an example, consider the vector for the letter J.

1 1 1 1 1 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 J

Adding 20 % noise onto this gives:

0.9354	1.0636	0.9978	0.9996	1.3213
0.1095	0.0536	0.9153	-0.0141	0.0296
-0.1000	-0.0673	1.083	0.0116	-0.0889
-0.0196	0.0245	1.0163	-0.0543	0.0828
-0.0956	-0.0043	1.0635	0.0032	0.0499
0.9985	0.0771	0.9504	-0.0452	-0.0890
-0.0226	0.9582	0.0124	-0.0128	0.0703

The effect of adding noise to the sequence is shown overleaf.

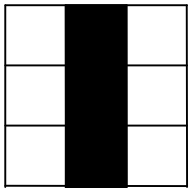
See pages 11.40 - 11.46 in the *MATLAB Neural Network toolbox manual, H Demuth and M Beale, The MATH WORKS inc, 1994* for further details.

Illustration of noisy J

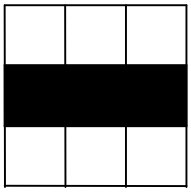
Note: Illustration not available in PDF version

<u>Small Characters</u> - For the recognition of simple shapes.

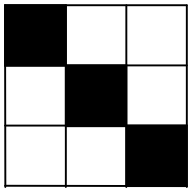
For small networks and simple tasks a set of simple shapes were defined. These are laid out on a 3 by 3 grid. Originally there were ten, but an MSc student who was working on a different project introduced further examples. The idea is illustrated below.



0 1 0
0 1 0
0 1 0



0 0 0
1 1 1
0 0 0

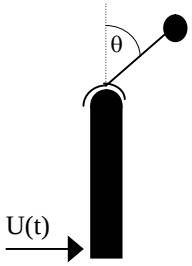


1 0 0
0 1 0
0 0 1

Reference: *MSc Thesis: Application of Taguchi Methods to Neural Network Learning, Geva Dror, 1995, School of Electronics and Electrical Engineering, RGU.*

<u>Non-linear systems characterisation</u>- from the MATLAB Neural Networks toolbox.

MATLAB contains a model of an inverted pendulum system as shown below:



The arm is controlled by a DC motor attached to the pendulum. The model can be obtained within MATLAB as *pmodel* which takes current (for the motor), angle θ, velocity and time and returns the derivatives of angle, velocity and force. The role of the network is to transform the current system state into the next state. State is defined as angle and velocity. The model is run from MATLAB with the following commands:

```
x = [angle; velocity; force];          Set up model vector
dx = pmodel (t,x);
```

See pages 11.26 - 11.31 in the *MATLAB Neural Network toolbox manual, H Demuth and M Beale, The MATH WORKS inc, 1994* for further details.

Waveform prediction - from the MATLAB Neural Networks toolbox.

There are two sets of waveform prediction models outlined in the MATLAB manual. Linear prediction and adaptive prediction. Both were used to test networks.

Linear prediction:

This simply sets up a sine wave which is the input to the network (usually in a delay chain form). The network then has to predict the next values of the wave:

```
time = 0: 0.1 : 100;          Loop 0 to 100 step 0.1
x = sin (time * pi);          Generate sine wave (x = input to network at t = n)
```

Adaptive prediction:

This is similar to the above except that the waveform can change amplitude and / or phase and frequency. Typical example:

```
t1 = 0: 0.1: 4;                       Loop 0 to 4 step 0.1
t2 = 4.1: 0.1: 8;                     Loop 4.1 to 8 step 0.1
time = [t1 t2];
x = sin (t1 *pi) sin (t2 * 2*pi);     Two component sine wave
```

This produces a wave which doubles in frequency at t = 4.1.

Other training sets

Two other training sets are described in the thesis. Both of these are explained in the main text so no further explanation is necessary here, they are:

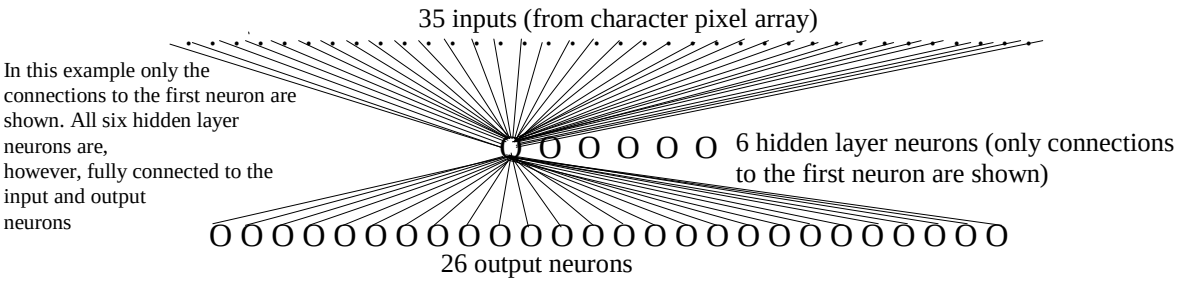
Linear Separability	Section	7.3.1
Time series generation	Section	8.4.1

Both were devised for the project.

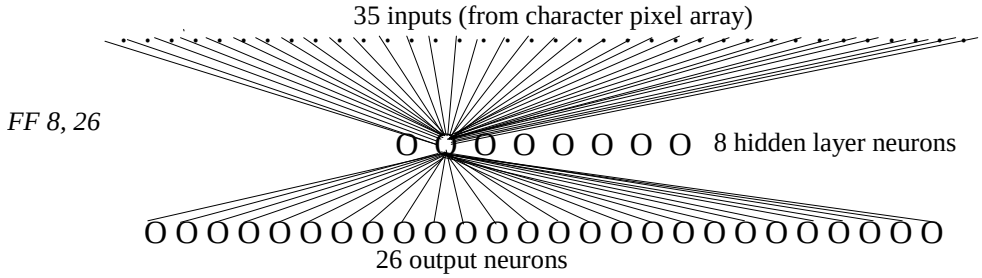
Appendix 5

Further illustrative results.

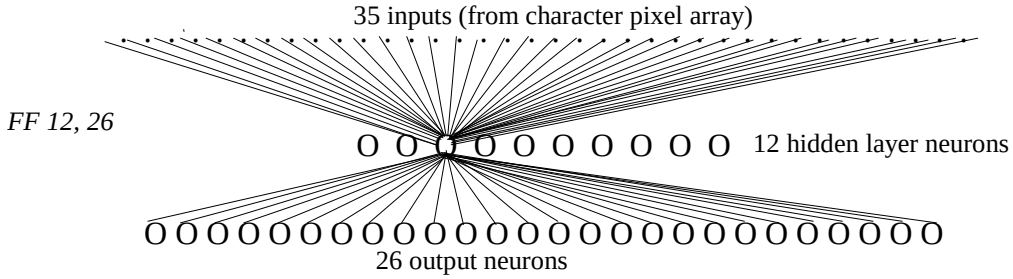
Due to lack of space in the main text of the report, abbreviations where used to show the manner in which the networks grew. It is useful however to illustrate an example explicitly. Consider example 8.1, which starts with the network *FF 6, 26.*



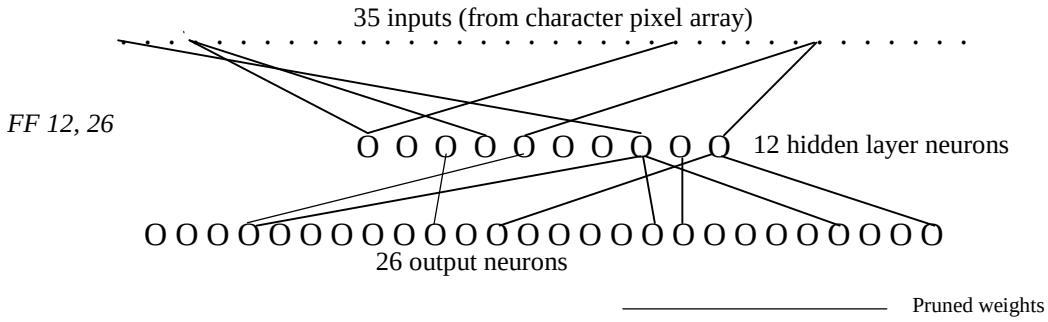
The network now grows by two neurons in the hidden layer. The network, however, remains fully connected. The abbreviation of this growth is **N(2) 2**, that is, add neurons (**N**) to the second (hidden) layer (**N(2)**) and the number of neurons added is 2 (**N(2)2**). The result of this is shown below.



This time connections to the second hidden neuron are shown (because it made the diagram easier to draw). The next stage of network growth is N 2 (4). Another 4 neurons are added to the hidden layer.



This time all the connections to the third hidden neuron are shown. In the next stage two percent of the weights which show the smallest activity have been deleted (**C(2)**-). However, since this is difficult to show, for the purposes of illustration I will only show the pruned connections.



The next stages were the application of bias, first to the hidden layer and then to the output layer (twice). This was accomplished, in this case, by using neurons with an external bias input of the form:

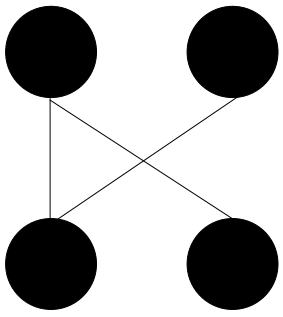
$$S = \sum xw + bias$$

(where the symbols have their conventional meaning), this is not illustrated as it makes the diagram too cluttered.

To illustrate a smaller network, here is one grown for feedback lineralisation of a non-linear system.

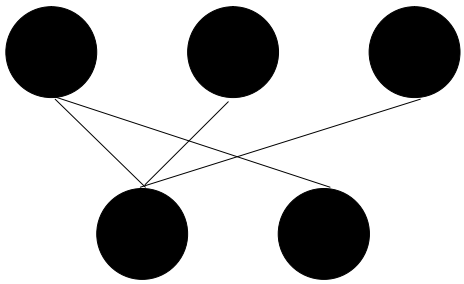
See pages 11.32 - 11.39 in the *MATLAB Neural Network toolbox manual, H Demuth and M Beale, The MATH WORKS inc, 1994*

Start Network.

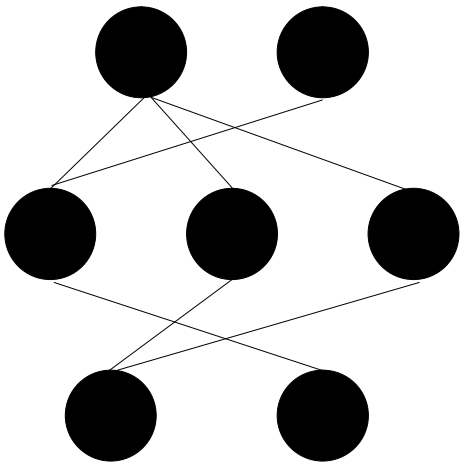


FF 2, 2 (pruned)

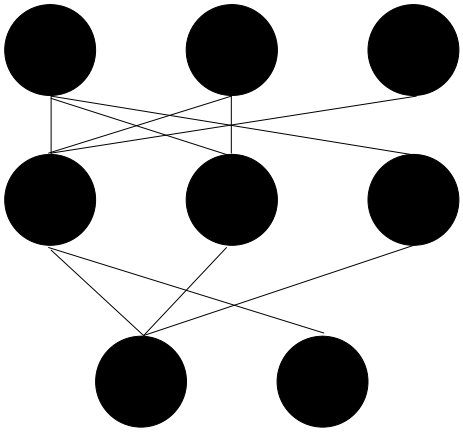
Growth **N (1) 1**



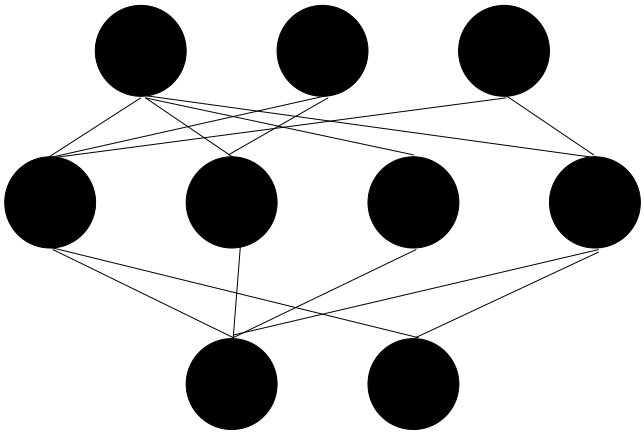
Growth L (in this case because of the nature of the problem the added layer maintains the connectivity of the previous layer)



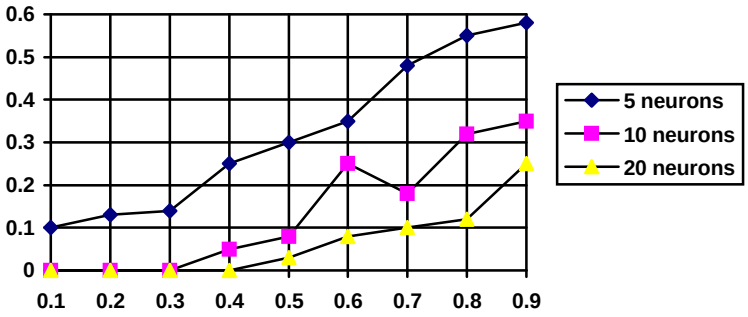
Growth N (1) 1



Growth S L (1)



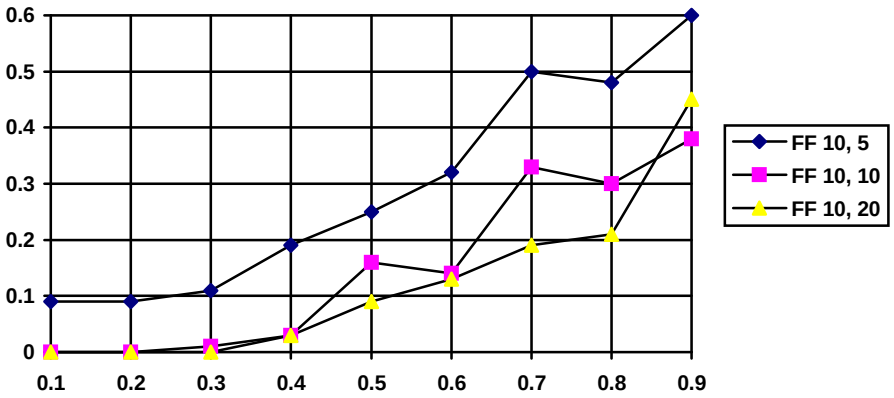
In the main text of the report only sample results are given for each strategy. To illustrate some of the more detailed work which went into the gathering and analysis of these results, some aspects of one example will be looked at more closely: that of growing a network for noise tolerance (example 8.4). First consider the noise performance of a network as it increases its number of neurons:



x axis = Noise added (1 = 100%)
y axis = error probability

Above 20 neurons the network error rises again as the network has stopped generalising

The above diagram shows three representative results, altogether 30 measurements were taken. Consider as well the effect of adding hidden layers onto the network. This was not illustrated in the results in the main text.



The labelling of this graph is the same as for the previous example. Notice how at the end the smaller network is performing better than the middle network (the FF 10, 10 network is performing better than the FF 10, 20 at a noise level of 0.9). These are three of 60 experiments designed to investigate the function of layers in networks subject to noise.

To illustrate the process further, consider the process of increasing the number of neurons in a character recognition problem. The training graphs are shown below. These graphs may not tie in exactly with the results illustrated in example 7.6 because in the final results an average of four runs with different initial weights was taken.

Finally, extra results are given for some problems not tackled (due to space and brevity) in the main text:

1. Adaptive noise cancellation (p 11.18 MATLAB toolbox manual)

Start network FF 1 (with 5 time delay taps)
Growth for lowest error: I 3, L, I 2, N(2) 2
End network FF 3 (with 10 time delay taps)

2. Amplitude detection (p 11.21 MATLAB toolbox manual)

Start Network FF 1, 5, 1 (suggested network is an elman)
Growth for simplest network: F (2) 3 1, F (2) 4 1, I 2, N(2) 2, F (2) 6 1
end network FB 3, 5, 1

3. Hopfield reconstruction of letters

Start Network FF 15, 35
Growth for simplest network F (2) 2 6, F (2) 4 3, F (2) 2 7, F (2) 14 21, F 12 31, B (2)+, C(2) -
Finishing network FB 15, 35

Appendix 6

The Biological basis of inheritance.

1. Mendel's discovery.

The best way to illustrate Mendel's discovery is with an actual example:

Let us suppose that there is a plant which can be short or tall. We breed this plant in two populations: one is always tall and all its offspring and their offspring through the generations are always tall. The same is done with the short plants. It is said that the plants breed *true*.

Now let us breed a tall plant with a small plant. It turns out that all the offspring plants are tall. This new generation of tall plants is called the F1 generation (technical name 1st *filial*). We say that the *factor* for tall is dominant (denoted *T)*, and the factor for short is regressive (denoted *t)*.

Now if we take the F1 generation (all tall) and <u>self fertilise</u> them we get the F2 generation. One quarter of the F2 generation are small, three quarters are tall. The reason for this is illustrated in the diagram below.

Original Generation (F0)

Parent *a*	Parent *b*
TT	**tt**
(tall)	(small)

First Filial generation (F1)

Child *a*	Child *b*
Tt	**Tt**
(tall)	(tall)

Second Filial generation (F2)

Grandchild *a*	Grandchild *b*	Grandchild *c*	Grandchild *d*
TT	**Tt**	**tT**	**tt**
(tall)	(tall)	(tall)	(small)

In other words the results could be explained by assuming that factors were inherited in pairs, one from each parent plant. Whenever T was present it dominated over t. This 4:1 ratio is important and can be shown to apply to other mutually exclusive contrasting traits.

2. The discovery of Chromosomes.

Late in the 19th century, long dark bodies were observed within cells, but only at the time of cell division. These bodies are known as chromosomes. The number of chromosomes in a particular species of animal is constant. It was also discovered that an egg cell and a sperm cell each contained half the number of chromosomes in the cells of the adult animal.

Through observation of the fertilisation process, it became obvious that these structures contained the hereditary factors of the animal.

In 1903 Sutton and Boveri made the connection between Mendel's 'Factors' and the chromosomes. They noted that the chromosomes allows appeared in pairs, as did factors. They postulated (correctly) that the chromosomes therefore probably contained the factors arranged in a linear sequence. The factors eventually became know as *genes*.

Further research showed that the situation outlined above is much more complex; different genes on a chromosome can be linked or related (and therefore so can genetic traits) and during the breeding process the chromosomes break and cross link to each other (this is how traits from the parents are passed to the offspring).

3. Drosophila

From 1909 onwards experimentation focused on *Drosphila Melangaster* - a fruit fly. This animal has only got four pairs of chromosomes, and was easy to experiment with.

Using these animals, in the late 1920's, the basis of mutation in genes was discovered - one of the driving forces behind evolution. Researchers discovered that most, but not

all mutations were lethal, and further that each individual gene had a characteristic rate of mutation in the absence of external factors. Mutation rate could be increased by the addition of chemicals, heat, radiation and other agents.

Using these animals, it was shown that mutation was not the only source of variation among the animals, recombination and rearrangement errors during chromosome copying could also lead to wide variation in form.

4. DNA (deoxyribonucleic acid).

In 1953, Watson and Crick showed that the chemical basis for the genetic code was a combination of 4 amino acids wound in the form of a double helix on a 'backbone' of simple organic molecules. All known living organisms use this same chemical code.

In the DNA Amino acid bases act as a 'alphabet' of four letters which spell out the genetic code of the organism. These letters spell out 3 letter 'words' known as *triplets*. Each triplet is responsible for the manufacture of one of the 20 amino acids known. Amino acids themselves are the building blocks of proteins.

The system works by coping the code contained in the DNA to another complex organic molecule called RNA (= ribonucleic acid), the RNA then used the code as a template to make the amino acids for protein production. By this method of sending proteins to build and control the body functions the genetic code controls the processes of life.

This process is now so well understood that it is possible to implant sections of DNA from one organism to another, for example to cause one plant to create a poison (ie a protein) which kills insects in another. this process would have taken millions of years to evolve by random mutation and copying errors in the genetic code alone. Such interference is known as *genetic engineering*.

Appendix 7

Neural unit functionality.

Neural network units and functionality

During the course of this project, one question kept arising – what does a neural network do that other systems cannot? Neural networks are tricky to make in hardware or to program in software. It would be of great benefit if they could be simplified or made easier to implement. An extension of this basic point involves the future of the technology itself: what model will the ANNs of the future use and are our current methods sufficient to build them?

The discussion below considers the functionality of individual neurons and of the network as a whole. It expands the discussion to consider where neural networks fit into the scheme of available technologies as a whole, both now and in the future.

1 On the functionality of real and artificial neurons

Before examining the questions of neural network functionality, which were mentioned in the section above; let us first assume that we wish to continue using artificial neurons and examine whether the usual (McCulloch-Pitts or Kohonen) type models are really a good reflection of real neurons.

The most common artificial neuron model is that of the perceptron with a non-linear activation function. Various competitive type neurons are also widely used. Although these shows good results under some circumstances, their functionality is limited and they are a poor model of the biological neuron. For example, it is known that synaptic transmission from neuron to neuron is a complex process which is controlled by many different neuro-transmitters; chemicals in the surrounding fluid also mediate the activities of large numbers of neurons in a more general way. Neurons have inhibitory connections (which some models - such as the sigma-pi network - seek to include) and display a time response often described as a *leaky integrator* [1]. Even at a simple level, actual biological neurons operate by sending a train of pulses, the frequency of which is proportional to their stimulation (quite a different type of processing to that which takes place in the artificial neuron [2]); as we will see later, this dynamic attribute may be an important factor in network functionality. There is also evidence to suggest that biological neurons use streams of pulses to encode information in more

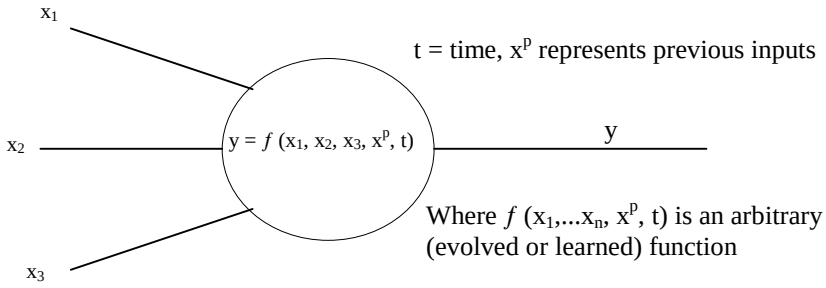
complex ways [3] – modulating information onto the pattern of pulses generated [2]. If this is indeed the case, we may have dramatically underestimated the potential computational ability of the biological neural network.

In recent years complex (and unsuspected) features of the biological neuron have also been discovered (such as local processing in dendrites). One respected author has even claimed that some neurons are affected by quantum mechanical phenomena [4].

Quite apart from these differences in operation, there are also many different types of neuron in the nervous system (see section 2.2.2) and connections from neuron to neuron are ‘plastic’ (able to rewire themselves) [5]. All this means that real biological neurons are difficult to simulate and good simulations of them (which are mainly used in biological research) are computationally expensive.

There are two approaches to overcoming this. Firstly, we can study the biological neuron closely and try and make more accurate models of it – this may be impossible if the operation of the neuron is not fully understood (for the reasons indicated above). Alternatively, we may consider using a neuron which is capable of learning any mathematical function (mapping) of its input vector to its output or use an Evolutionary Algorithm to evolve the functionality of the unit. These approaches are illustrated in figure 1.

Figure 1, A general neural model.

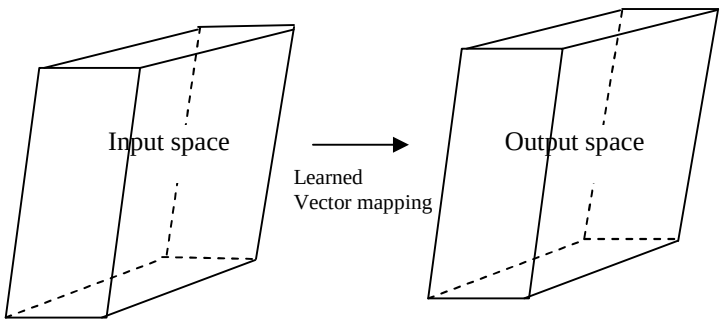


In other words, why evolve just the topology and weights of the network, why not evolve the neural unit functionality as another parameter.

An extension of this idea is to mix unit functionality evolution with concepts from Cellular Automata. Cells in the body are highly differentiated (both in the nervous system and elsewhere); different types of neurons performing different tasks. One way to achieve this in the artificial sense, is to have each cell containing its own rules or genetic code and allow local mutations (under evolutionary control) as the cells multiply. This could also be used as a mechanism to allow parts of the network to develop their own functionality (and therefore modularity).

The function generated need not even be continuous (in fact, it could be a program in the algorithmic sense). Each input plane might be mapped by a vector field onto an output plane as shown in figure 2 below. This would allow the neuron to learn functions with localized non-linearitys. Mappings not evolved directly could be interpolated from an existing vector field.

Figure 2, A vector mapping of input / output function.



There is still the question of the enormous connectivity of the biological neural network to consider. However, there are innovative ideas for overcoming even this. For example, information leaving the neuron could be ‘addressed’ to a recipient unit somewhere else in the network (similar to addressing packets in a communications system).

However, all this would prove rather a waste of time if the functionality of the unit neuron itself was of little importance. If the neuron were simply a biological ‘logic gate’ created by nature to fulfil a similar role as an artificial logic gate. In the section below an argument is given that this is indeed the case and that only certain properties of the functional unit are important and not the behavior of actual biological or artificial neurons.

2 On the functionality of networks

Let us start by comparing standard ANNs, Fuzzy Logic and Digital Electronics. Fuzzy Logic is a popular modern technique used to implement intelligent systems. At first, it appears quite different from ANNs and also from standard digital electronics. The argument given here is that these three technologies are actually aspects of the same system.

Fuzzy Logic was introduced by Lotfi Zadah in 1965 [6]. It was not well received by the engineering control community, typical comments being that it was common sense and ‘just’ statistics. Since the initial reaction, it has steadily gained acceptance and popularity, especially in Japan. Fuzzy Logic has found its way into many domestic and industrial products [7].

Fuzzy logic is most often applied to control systems. The term ‘Fuzzy’ arises from the fact that the system uses continuous as opposed to discrete inputs and outputs. The fuzzy logic system is basically an expert system, in which a continuous input is assigned a membership of several sets. These values are then operated on by an inference engine and the result, which is also continuous, is used to control the system, figure 3.

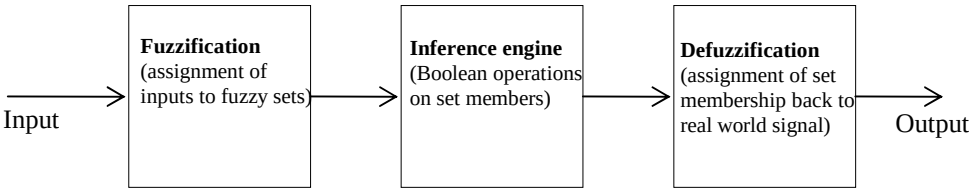


Figure 3, Fuzzy system.

Fuzzy logic systems show several advantages over conventional control systems, these include ease of design, and also better performance in many situations such as speed controllers in lifts and vehicles (where the continuous nature of Fuzzy logic causes them to speed up and slow down more smoothly).

Neural Nets, Fuzzy Logic and Digital Electronics appear quite different to one another, but are actually aspects of the same thing - a mapping system. In the case of combinational digital electronics the binary inputs are mapped onto binary outputs, figure 4.

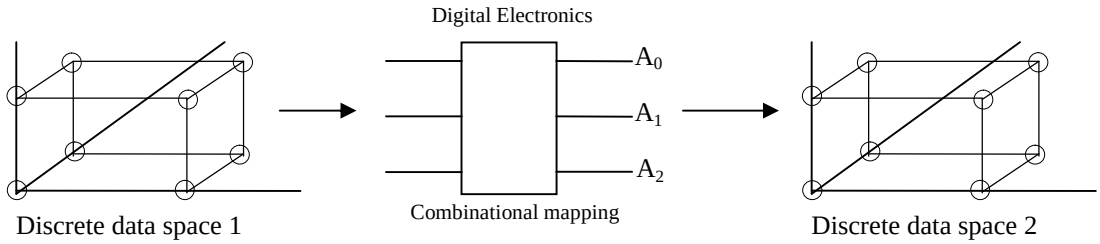


Figure 4, A discrete mapping.

The circuit is designed by the engineer, using Boolean Algebra or Karnaugh Maps to achieve this mapping. In the case of fuzzy logic or ANNs the mapping is of a continuous input to a continuous output, figure 5. The system knows how to map because (in the case of ANNs) it has learned the relationship or (in the case of fuzzy logic) the relationship has been encoded, in the inference engine, by the designer.

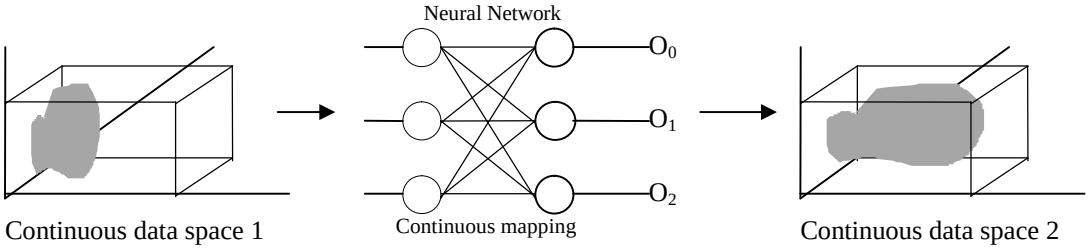
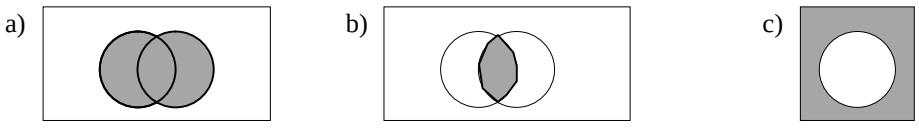


Figure 5, A continuous mapping (ANNs or fuzzy logic).

In many books, the operation of the neural net is viewed as dividing up data space with separators. This causes the network to have certain mapping limitations (the linear separability limitation). Although the networks shown above are combinational logic and feedforward networks, the similarities also apply to sequential logic which is equivalent to recurrent neural networks such as the Hopfield net. Notice how these mapping are spatial or instantaneous (they do not contain a time varying component), we will return to this aspect again shortly.

To illustrate the similarities between these various technologies let us introduce a new entity: the 'Fuzzy Gate'. This is a device which is based on normal logic functions but can fulfil many of the operations of an ANN.
The Fuzzy Gate is simply a digital gate, which accepts an analogue or continuous input rather than a discrete or binary input (as in the case of 'normal' digital electronics). First, consider the functions of normal logic represented as Venn diagrams, figure 6 a,b,c.



$A + B = A \cup B$ $A \bullet B = A \cap B$ A'

Figure 6, Venn diagram representation of logic building blocks.

In terms of normalized voltage levels these represent MIN, MAX and COMPLEMENT functions, figure 7 (these functions are actually used within the fuzzy logic inference engine).

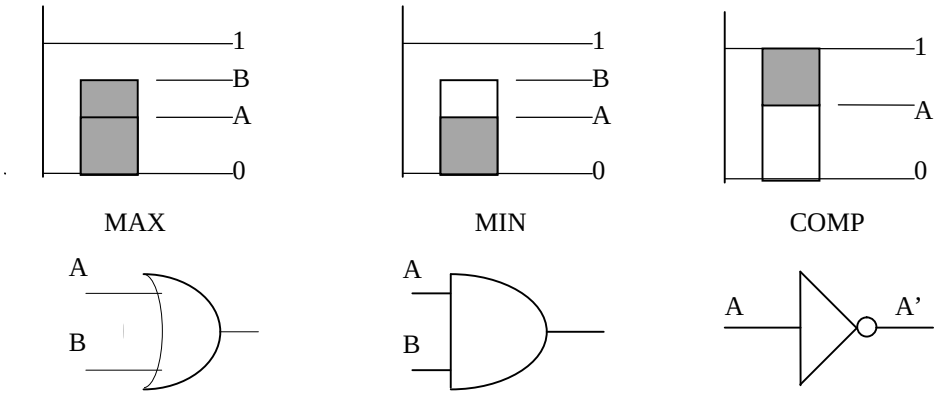


Figure 7, Boolean functions as continuous functions

Such a system, operating on continuous voltage levels has important advantages:

1. It decomposes to normal logic if the inputs are 1 and 0.

Consider the Boolean expression **A + B (A + B)** which reduces using normal methods to **A + B**, figure 8.

A discrete	B discrete	A cont	B cont	A + B = D MAX	D . B = E MIN	E + A MAX (answer)	A + B Boolean answer
0	0	small	small	small	small	small	0
0	1	small	large	large	large	large	1
1	0	large	small	large	small	large	1
1	1	large	large	large	large	large	1

Figure 8, Fuzzy decomposition to normal logic.

It may be seen that the continuous logic gives the same answer as the standard discrete logic; this can be shown in every case.

2. It may be arithmetically manipulated in the same way as normal logic.

The discussion in the example above shows that the continuous logic reduces to discrete logic if the levels are set to their maximum and minimum values, in general:

$A \cup B = A + B \qquad A \cap B = A \bullet B \qquad A' = \overline{A}$

and they may be manipulated as such.

3. It degrades gently.

Consider the case **A + B**. If a high input to a continuous logic gate is 0.7 and low 0.2, figure 9.

A	B	A + B
0.2	0.2	0.2
0.2	0.7	0.7
0.7	0.2	0.7
0.7	0.7	0.7

It may be seen that the system degrades gracefully, if logic levels do degrade.

Figure 9. Noise tolerance.

4. Mathematical logic circuits continue to fulfil their functions

Consider the half adder, figure 10.

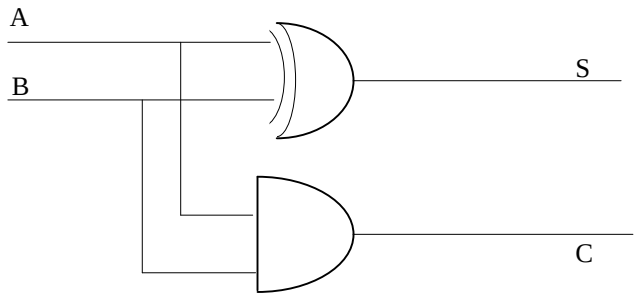


Figure 10, half adder

The XOR is implemented as $\mathbf{\overline{A} \cdot B + A \cdot \overline{B}}$. Consider the conditions in figure 11.

A = 0.4 B = 0.5

A	B	$\overline{A}$	$\overline{B}$	$\overline{A}B$	$A\overline{B}$	$\overline{A}B + A\overline{B} = S$
0.4	0.5	0.6	0.5	0.1	0	0.1

A . B = C = 0.4 output is therefore $(0.1 \times 2^0) + (0.4 \times 2^1) = 0.9$

Figure 11. fuzzy XOR function.

To clarify this, one of the operations involving the complement function is shown in figure 12.

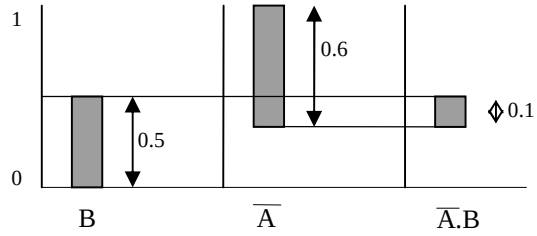


Figure 12, An operation using the complement function

As we can see from figure 12, because of the NOT or Complement function, a true fuzzy digital output requires a number pair to represent it (in the case of the function above 0.4, 0.5 ie lower limit, upper limit; for complex functions, there could be several number pairs).

Using such techniques it is possible to design, for example, a pattern recognition or control system which gives similar results to a neural network system. However, fuzzy gate systems have the advantage that they can be manipulated and proved using Boolean Algebra or Set Theory.

The modern techniques (neural networks and fuzzy logic), give important advantages over Boolean logic gates; these are:

- Neural networks can learn from example (so no detailed design is involved).
- Neural nets and fuzzy logic operate with continuous signals.

As pointed out, the continuous nature of signals gives the system certain useful properties:

- It causes the system to degrade gently in the presence of noise.
- It means that control systems such as lifts behave more naturally (it interfaces better with the natural world).
- It makes the system more fault tolerant.
- Fuzzy systems have been shown to model the way people reason better than discrete systems.

As has been shown, standard digital logic is a special case of fuzzy digital logic. This means that systems designed using fuzzy digital electronics behave in a very similar way to neural networks or fuzzy logic, and further, that ‘hard’ digital logic is a special case of ‘fuzzy’ digital logic.

The other main difference between standard logic and neural networks is that neural networks learn mappings through a training cycle; the other two systems have a designed structure. However, it is possible to design fuzzy digital logic systems which can learn. One way this can be achieved is by implementing a training algorithm which adjusts the threshold of the MAX and MIN functions of the gates so as to minimize the output error. One can therefore envisage a compound system *neuro-fuzzy digital electronics*. This would have all the advantages of Boolean logic (for example, easy mathematical manipulation), but have continuous outputs and be trainable in the manner of an ANN.

To look even further ahead: the design of the neural processing elements, is based (somewhat loosely) on the structure of biological neurons. This association has caused many workers to only consider models which are biologically feasible and this may have ham-strung researchers somewhat (in fact it has almost become an ideology). Future networks may take a much less rigorous view of neural units, and allow them to have designed or evolved processing capabilities. The end result may be a general technology (one stage beyond neuro-fuzzy digital electronics) which has all the advantages of the previous examples without any of the disadvantages.

Hopefully the illustration above will have shown that the almost religious belief which some researchers have in the biological neuron and complex models of it is misguided. Why copy nature when man can possibly do better? In the system it is not the processing elements themselves which are important but rather their ability to process continuous information and to learn from example.

However, there is more to the story than this, because the systems which we have been considering so far are for mapping one instantaneous value to another; there is also the question of how to map patterns which occur with in the time domain.

3 The relationship between ANNs and Digital Signal Processing (DSP).

Compare the artificial neuron in figure 1 to the non-recursive Finite Impulse Response (FIR) filter [8] in figure 13.

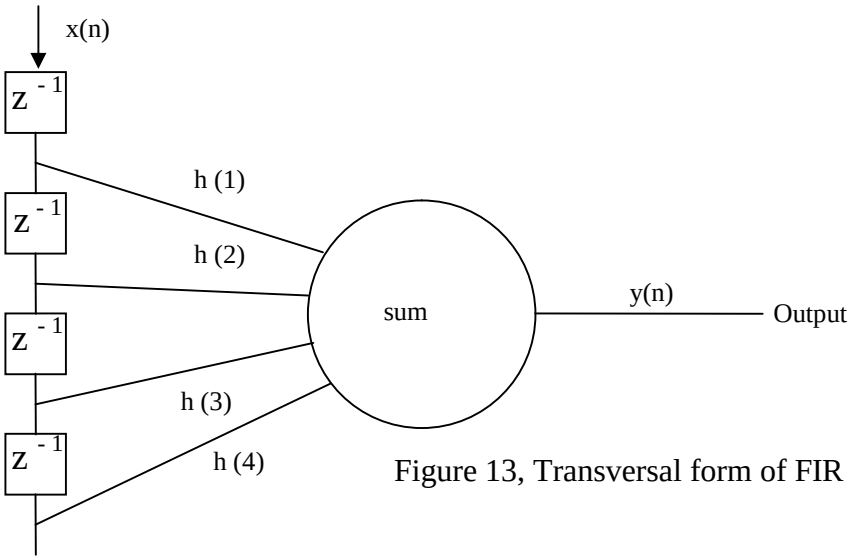


Figure 13, Transversal form of FIR filter.

$$S = \sum_{g=1}^{g=n} h(g)x(n-g) \qquad \text{(eqn 12.1)}$$

n = number of delay units

The difference between the artificial neuron and the FIR filter is therefore simply the delay line with unit delay z^{-1} and the fact that the FIR version has no threshold.

What is happening, is that the delay line is converting a time series into spatial pattern by 'holding' the values from different time periods so that the neuron can recognize a pattern in the series. This system is sometimes refereed to as a 'time delay' neuron or network.

In normal filter design, the filter coefficients h(n) are calculated laboriously [9] and are then fixed, so the filter cannot change. Several workers however, have developed algorithms for *adaptive filters* [10]. These allow the filter to 'learn' in the same way as an ANN (the algorithm is basically the same as BP). For an ANN, training is achieved by changing the weights, w_x of the network. In the DSP model this is equivalent to

changing the coefficients h(g) in the FIR filter above. In terms of a filter this means that it can change its transfer function adaptively. This can be used either to design the filter before it runs or to make the filter change its characteristics in real time as it processes a signal. For an excellent summary of this, see Ifeachor & Jervis [11].

It is also possible to make filters using a network rather than a simple neuron, figure 14.

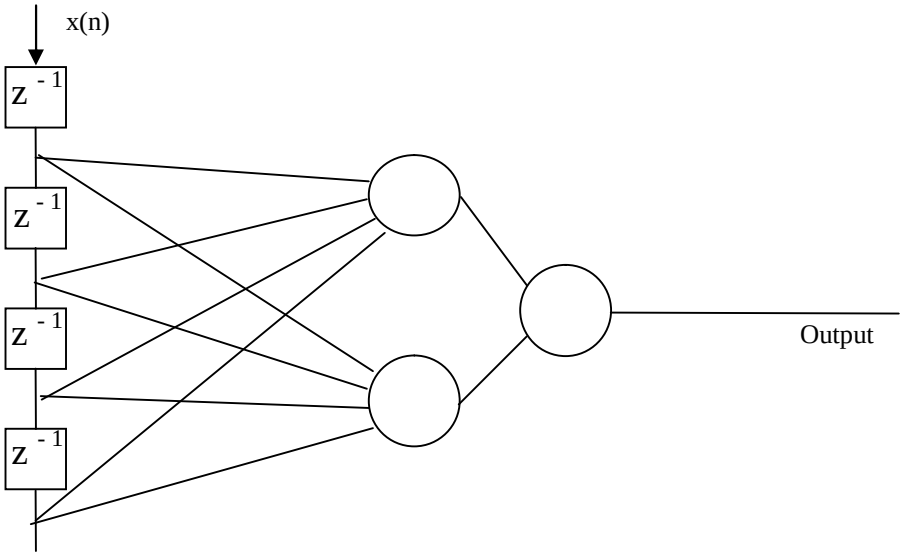


Figure 14, A FIR filter using networked neurons.

And by using several neurons connected to another (output) delay line, we can also produce a time series to time series mapping, figure 15.

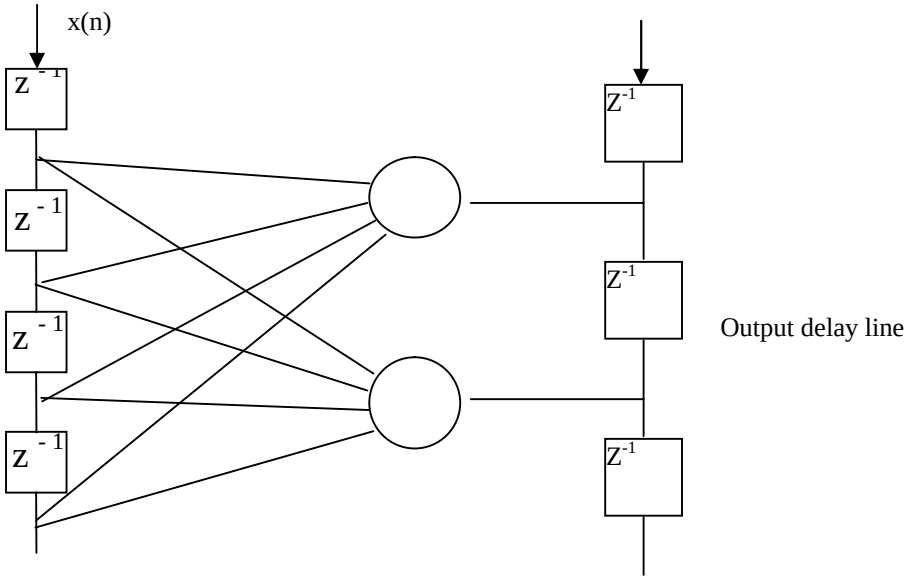
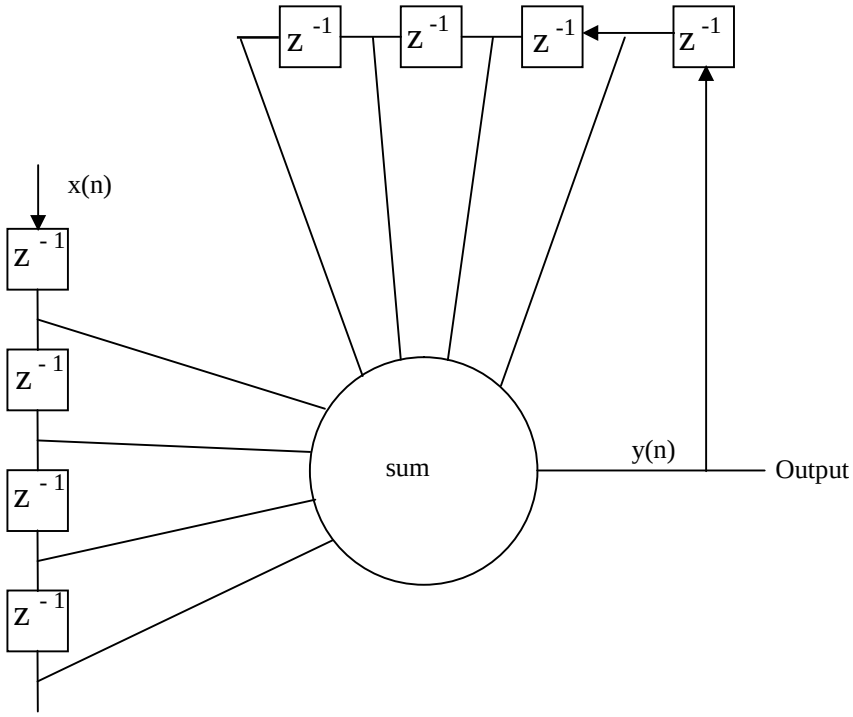


Figure 15, time series mapping.

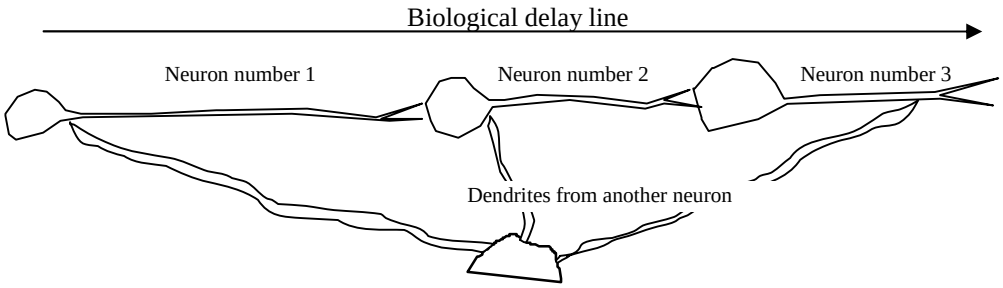
Recursive Infinite Impulse Response (IIR) filters are the DSP equivalent of the Hopfield network, figure 16.

Figure 16, an IIR filter.



In the biological network, data travels at a finite rate. Since each neuron is attached to others, there is no reason why the network should not form a filter structure as shown in figure 17.

Figure 17, a biological filter.



Some thought will show that 'filtering' is part of perception and likely to be a key element in the BNN. By this argument we need another element in our evolutionary neural network, that of a time delay. This will allow the evolving network to form temporal filtering structures (see section 7.3.5). Note that the biological representation of data (as a pulse stream) is possibly more suitable for processing in this way than the artificial representation, which brings us to the next topic.

4 The dynamics of neural networks

It has been known for many years, that the BNN contains structures which produce self sustaining, regular or *oscillatory* signals. For example, neurons which control the regular beating of the heart and other automatic functions need to produce pulsing excitatory signals [12]. Recently, however, the view that it is the pattern of electrical impulses around the nervous system which is important, as opposed to the actual wiring, has gained ground [13]. From this perspective, it is in the *dynamics* of the system that information is stored or processed rather than in the synaptic connections themselves. Taking this to its logical conclusion, one could even say that thought itself can be viewed as a changing pattern of pulses.

The argument in the above paragraph makes sense if one considers the behavior of the mind when it is deprived of sensory input. In most artificial neural networks, the lack of input will cause dynamic behavior in the network to cease (even in a Hopfield network, it will eventually stabilize – ANNs are not designed to be unstable). In

humans, sensory depravation causes vivid dreams and hallucinations. This can only be explained if the brain generates its own major dynamic patterns without input. One might take the view that signals rush hither and thither through the BNN; sensory input forces these into regular patterns (or effects their evolution) when it is applied. The system in which these signals exist must itself be able to adapt quickly; a changing dynamic signal corresponds to a 'train of thought', changing and evolving as time proceeds. Indeed, a system such as this is the only way of accounting for a *stream* of consciousness.

In the previous paragraph, there are some sweeping statements. However, evidence to support this view does exist. In particular, W J Freeman has studied the olfactory (smell) system in rabbits. In a whole series of papers starting around 1975 [14] and continuing onwards [15] he has suggested (and backed up with experimental evidence) that the olfactory neural net in rabbits is a dynamic neural network displaying chaotic behavior. Each smell the rabbit remembers, is a moving pattern of pulses - never predictable - a chaotic attractor. A new smell causes a new pattern to be set up. For a summary of Freeman's work, see Wasserman [16]. Evidence that at least part of the brain operates in this way (possibly some of the higher levels) is growing and raising much interest [13]. If this is indeed the case, complex ANN models may not be necessary to simulate such dynamical functions. Simple genetically evolved structures of gates and delays may be sufficient to provide the dynamic element. Learning and dynamical behavior might be implemented using programmable delays or by encoding information as a pulse modulated signal.

To produce simulations of such complex dynamical systems, a neural model would need to be built up which is similar to the biological neuron. This can be done (and has been done) but is costly in terms of processing power required for simulation. However, work by Adrian Thompson on digital circuits [17] has showed that it is possible to generate complex dynamical signals from simple building blocks. In his case, digital gates were used. He evolved the circuit under control of GA until it could perform a preset task. The important part of the digital network was not the logic functions themselves but rather the propagation delays between the gates in the circuit. These delays interact, one with another, and produce a dynamic or time series

waveform at the output. The output thus produced was in fact a series of spikes very like action potentials.

The gates in the above example let transients interact to produce complex behavior (and this may be the function of biological neurons). It would be possible, however, to use a series of simple time delays, gating functions (but not necessarily digital gates), integrators and oscillators, to produce similar results without having to resort to digital logic. In this case, the network could change its behavior by changing the time values of the delays used in the system.

In another development, the company *cyberlife* [18] has come up with an ingenious idea which they use to train a more standard neural network, but which could also have implications for this type of system. Their idea is that the chemicals (such as hormones) released into the blood stream in the presence of pain or pleasure (or hunger or anger) effect the synaptic strengths between the neurons. If this has a basis in fact it could 'close' the training loop for biological neurons – in effect be the 'error' which is fed back to the network to train the units (it has long been argued that there is no error feedback mechanism in the BNN and therefore supervised learning systems such as BP are not biologically feasible). It has also been suggested that chemical learning of a similar nature could take place via the glial cells.

It could even be that, in higher animals, the lower networks in the body (such as the reflexes) serve only to map these higher (dynamic) functions into movement of muscles or secretions from glands. There is certainly plenty evidence that the higher functions operate in a different way from the those at the lower level; they use different types of neurons for example. There is also no reason way nature would not have designed these networks to operate in a different manor 'on top' of the older, simpler networks, if it gave an evolutionary advantage.

5 Conclusions

All the systems in the previous sections were capable of being built from the same (simple) units. Since evolution is 'blind' – it will evolve what does the job without worrying about aesthetics, it holds that all these systems probably exist in reality,

within the biological nervous system. However, the complexity need not bother us, as we have models of the evolutionary process to evolve systems which we cannot understand. It is obvious however, that to evolve these types of function we will have to extend our ideas on neural networks beyond the current, simple models, so that they are capable of forming any of these systems.

It could also be argued, that as electronic technology develops, these systems will finally merge into an as yet non-existent technology, a *new electronics*, which will be a general mapping system with the important features presently contained within the individual systems. DSP is also included in the above, being a mapping system in the temporal domain. The author therefore suggests that it is time to stand back and view the functionality of electronics from an objective position and further considers that work on an encompassing framework for all these systems (practical and theoretical) is an important topic for further work.

As a side issue, the BNN can teach us one further thing which is of interest in practical electronics. That is, that the current trends towards smaller device sizes and faster chips are not how integration is achieved in nature but rather through use of the three dimensional space available, although this may mean the development of self organizing connection systems.

One reason for all this speculation in the first place, is that we do not have a predictive calculus of parallel systems. We have some rules which allow us to make limited predictions of behavior (such as stability predictions) but no complete system exists (a problem which, ironically, may be due to the wiring of our brains). Such a development would have a major impact on many areas of science since many physical systems (it could argued, *all*) are made up of small and interacting particles or units. Again, this should be a major area of future research.

It may prove to be the case, that as far as intelligent systems are concerned, complex learning systems are not actually required (see section 9.3.1). There are simple electrically conducting neurons in the PNS which have a limited learning capability and Thompson has shown how fixed functionality logic gates may evolve to perform

complex functions under GA control. Ijspeert et al [19] at Edinburgh University have evolved artificial controllers based on a Lamprey nervous system and found that it is relatively easy to evolve more efficient controllers than a real lamprey.

From all this work, the features which seem to be required in evolutionary neural systems are mapping networks, delays and possibly some form of memory device. These requirements are in themselves important subjects for research. It is likely that *all* these architectures and ideas are present in the BNN, evolution is blind, it will simply create the network necessary to fulfil the task and all the networks described may be implemented using biological neurons. It may also be the case that these circuits operate in different modes at different levels within the network. For example the networks in the spinal cord may be operating in a hardwired mapping mode and those in the cortex operating in a dynamic mode.

So, to have a chance of evolving higher functions, we need to introduce more flexablity and new elements (such as time delays) into our networks as well as finding a practical way to introduce modular structure into the network as discussed in chapter 10.

References:

1. *An introduction to neural networks*, K Gurney, UCL Press, 1997. p 20 - 24.
2. *dot dot dot dash dash dash*, New Scientist, vol 150, 2030. p 40 - 43.
3. *The Neuron*, I B Levitan and L K Kaczmarek, Oxford, 1997. P304.
4. *Zombies, Dolphins and Blindsight,* New Scientist, vol 150, 2028. p 20 - 27.
5. *Plastic minds*, New Scientist, vol 157, 2121. p 18.
6. *Fuzzy sets*, L A Zedah, Information and control, 8, 1965. p 338 - 53.
7. *Fuzzy controllers*, L Reznik, Newnes, 1997.
8. *Signals and systems*, M L Meade & C R Dillon, Chapman- Hall, 2nd ed 1995. p 21 - 31.
9. *Digital Signal Processing: A practical approach,* E C Ifeachor & B W Jervis, Addison - Wesley, 1995. p 251 - 363.

10. *Digital Signal Processing: A practical approach,* E C Ifeachor & B W Jervis, Addison - Wesley, 1995. p 541 - 568.

11. *Digital Signal Processing: A practical approach,* E C Ifeachor & B W Jervis, Addison - Wesley, 1995.

12. *Neuroscience,* M F Bear, Williams and Willkins, 1996. p458 - 484.

13. *Wild minds,* Anon, New scientist, 2112, 3 Dec, 1997. p 26 - 30.

14. *Mass action in the nervous system,* W J Freeman, Academic Press, 1975.

15. *Chaos in the biodynamics of pattern recognition by neural networks,* W J Freeman & Y Yao, 1990 int conf on neural networks. 1990. p 243 - 249.

16. *Advanced methods in Neural computing,* P D Wasserman, Van Nostrand Reinhold, 1993.

17. *Silicon Evolution,* A Thompson, Proc of Genetic Programming '96 (Stanford). p 444 - 452.

18. *Agents from albia,* C Davidson, New Scientist, issue 9th May 1998.

19. *Artificial Lampreys: Comparing naturally and artificially evolved swimming controllers,* A J Ijspeert et al, Proceedings of the 4th European conference on artificial life (ECAL 97), MIT Press, 1997. p 256 - 265.

Appendix 8

Copyright letters.

Note: These letters (which gave permissions from various authors for their work to be quoted) are not included in the PDF version for technical reasons. See original version for full details.

Bibliography

Neural Networks

1. *Neural Computing: theory and practice,* P D Wasserman, Van Nostrand Reinhold, 1989. *

2. *Artificial Intelligence,* I Pratt, MacMillan, 1994. *

3. *Foundations of neural networks,* T Khanna, Addison Wesley, 1990. *

4. *Advanced methods in Neural Computing,* P D Wasserman, Van Nostrand Reinhold, 1993. *

5. *Neural network architectures: an introduction,* J Doyhoff, Van Nostrand Reinhold, 1990.

6. *Artificial Neural Networks: Foundations, Paradigms, Applications and Implementations,* P K Simpson, Pergamon Press, 1990.

7. *An introduction to Neural Computing,* I Aleksander & H Morton, Chapman and Hall, 1990.

8. *Neurocomputing,* R Hecht-Nielsen, Addison Wesley, 1990.

9. *The fundamentals of neural networks,* L Fausett, Prentice Hall, 1994.

10. *Neural computing: an introduction,* R Beale & T Jackson, Adam Hilger, 1990.

Genetic algorithms and Evolutionary techniques

1. *The blind watchmaker,* R Dawkins, Penguin books, 1991. **

2. *Genetic Algorithms in search optimisation and machine learning,* D E Goldberg, Addison-Wesley, 1989. **

3. *Handbook of genetic algorithms,* L Davis (ed), Van Nostrand Reinhold, 1991.

4. *Combinations of Genetic Algorithms and Neural Networks: A survey of the state of the Art,* J D Schaffer D Whitley L J Eshelman, COGANN -92 (conf proc), IEEE comp soc press, 1992. **

5. *Evolutionary Artificial Neural Networks,* X Yao, Int Journal of Neural Systems 4, 3 (Sep 93), World Scientific pub co, 1993.

6. *Practical Genetic Algorithms,* R L Haupt & S E Haupt, Wiley, 1998.**

Neurology and Biology

1. *Human anatomy and physiology,* R Carola et al, McGraw Hill, 1992. *

2. *The physiology of nerve cells,* D H Paul, Blackwell Scientific publications, 1975.

3. *The biology of the brain from neurons to networks: readings from Scientific American,* R R Llinas (ed), 1989. *

4. *Mind and Brain,* Scientific American (special issue), Vol 267, 3, Sep 92.

5. *Basic Neuroscience: anatomy and physiology,* A C Guyton, W B Saunders, 1987.

6. *The understanding of the brain,* J C Eccles, McGraw Hill, 1979.

7. *The Brain: The last frontier,* R M Restak, Warner Books, 1979.

Evolution, Embryology and Biology

1. *The blind watchmaker,* R Dawkins, Penguin books, 1991. *

2. *The origin of species,* C Darwin, (reprinted by penguin books).

3. *Living embryos: An introduction to the study of animal development,* J Cohen, Pergamon Press, 1967.

4. *The penguin book of the natural world,* Edited E Martin et al, Penguin books ISBN 010037144.

5. *Wonderful life: The burgess shale and the nature of history,* S J Gould, Penguin books, 1989.

6. *The Theory of Evolution,* J M Smith, Pelican Books, 1975.

7. *Evolution,* A Panchen, Bristol Classical Press, 1993.

Practical implementation

1. *Artificial Intelligence,* I Pratt, MacMillan, 1994. *

2. *The student edition of Matlab,* Prentice hall, 1995. (ISBN 0-13-184979-4)

3. *MATLAB Neural Network Toolbox manual,* H Demuth & M Beale, The math work inc, 1992.

4. *Practical Neural Network recipes in C++* , T Masters, Academic Press Inc, 1993.

5. *C++ Neural networks and fuzzy logic,*V B Rao & H V Rao, MIS Press, 1993.

6. *Digital design using field programmable gate arrays,* P K Chan & S Mourad, Prentice Hall, 1996.

7. *Programmable logic: PLD's and FPGA's,* R Seals & G Whapshott, Macmillan, 1996. *

8. *An introduction to Neural computing,* I Aleksander & H Morton, Chapman and Hall, 1990.

Important theory

1. *Perceptrons,* M L Minsky & S A Papert, MIT Press, 1969.

2. *On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition,* A N Kolmogorov, Doklady Akademii SSSR, Vol 144 p 679 - 681, 1963. (American Mathematical Society translation - 28: 55 - 59)

3. *Absolute stability of global pattern formation and parallel memory storage by competitive neural networks,* M A Cohen & S G Grossberg, IEEE transactions on systems, man and cybernetics, 13: p 815 - 826. 1983.

4. *Foundations of neural networks,* T Khanna, Addison Wesley, 1990. *

Work relating to the application of Embryological Techniques to ANNs or related systems

1. *Use of Genetic Algorithms in Neural Network definition,* F J Vico & F Sandoval, IWANN 91 (conf proc) p 196-203, ISBN 3 540 54537, Springer Verlag, 1991.

4. *Neurite Networks: the genetic programming of cellular automata based neural nets which GROW,* H de Garis, IJCNN 93 (conf proc), p 2921 - 4, vol 3, IEEE, 1993. **

Work mentioning Biomorphs and ANNs

1. *The blind Neural Network Maker: Can we use constrained embryologies to design animat nervous systems,* O Holland & M Snaith, ICANN 91 (conf proc), p 1261 - 1264, North Holland, 1991.

2. *The use of recursively iterated structures to constrain neural network architectures,* O Holland & M Snaith, IJCNN 91 (conf proc) , IEEE, 1991.

Work on ANNs which GROW - Selected important papers.

1. *GAL: Networks that grow when they learn and shrink when they forget,* E Alpaydin, International journal of Pattern Recognition, Vol 8, 1. p 391 - 14. **

2. *Dynamic node creation in backpropagation networks*, T Ash, Connection science, 1, 1989. p 365 - 275.

3. *Growing non-uniform feedforeward networks for continuous mappings,* V V Vinod & S Ghoso, Neurocomputing, 10, 1996. p 55- 69.

4. *Asymptotic inferential capabilities of feed-foreward neural networks,* E Ferran & R Perazzo, Europhysics letters, 14, 2. p 175 - 180

5. *Dynamic structure adaptation in feedforeward neural networks - An example of plant monitoring,* R Kozma & M Kitamura, IEEE con on Neural Networks, 1995. p 622 - 627 vol 2. *

6. *Pattern specific Neural Network design,* M Anderle et al, Journal of statistical physics, 81, 3 / 4. 1995. p 843 - 9. *

7. *Neurite Networks: the genetic programming of cellular automata based neural nets which GROW,* H de Garis, IJCNN 93 (conf proc), p 2921 - 4, vol 3, IEEE, 1993. *

8. *Neural network design based on evolutionary programming*, J Fang & Xi, Artificial Intelligence in engineering, Vol 11. 1997 p 155 - 161. **

Fuzzy logic, DSP, Digital and related techniques.

1. *Fuzzy logic with engineering applications,* T J Ross, McGraw-Hill, 1995.

2. *Fuzzy controllers*, L Reznik, Newnes, 1997.

3. *Digital Fundamentals,* T L Floyd, Maxwell - MacMillan, 1994. *

4. *Digital Signal Processing: A practical approach,* E C Ifeachor & B W Jervis, Addison - Wesley, 1995. *

* - specially recommended books or papers.
** - work of particular interest or importance.

References

Chapter 1.

1. *How Neural Networks learn from experience,* G F Hinton, Scientific American, Vol 27, 3, Sep 92. p 105 - 109.

2. *An Introduction to Neural Computing* I Aleksander & H Morton, Chapman and Hall, 1990. p xvii.

3. *Neural Computing: Theory and Practice,* P D Wasserman, Van Nostrand Reinhold, 1989. p 199.

4. *Human Anatomy and Physiology*, R Carola et al, McGraw-hill, 1992.p 377-378.

5. *Human Anatomy and Physiology*, R Carola et al, McGraw-hill, 1992. p 373 - 375.

6. *Neural Computing: Theory and Practice,* P D Wasserman, Van Nostrand Reinhold, 1989. p 189.

7. *Neural Computing: Theory and Practice,* P D Wasserman, Van Nostrand Reinhold, 1989. p 111.

8. *Neural Computing: Theory and Practice,* P D Wasserman, Van Nostrand Reinhold, 1989. p 46, 111.

9. *Genetic Algorithms in Search, Optimisation and Machine Learning*, D E Goldberg, Addison Wesley, 1989. p 1-2.

10. *The Blind Watchmaker,* R Dawkins, Penguin Books, 1991 edition. p 51 - 74, p 327 - 334.

11. *The Blind Neural Network Maker: Can we use constrained embryologies to design animat nervous systems.* O. Holland and M. Snaith, Proceedings ICANN 91, 1991. p 1261- 1264.

12. *The use of recursively generated iterated structures to constrain neural network architectures.* O. Holland and M. Snaith, Technical report (available on request), Technology Applications Group, Alnwick England, 1991.

Chapter 2.

1. *The understanding of the brain,* J C Eccles, McGraw-hill, 1979. p 9 - 17.

2. *Mind and Brain,* G D Fischbach, Scientific American, Vol 27, 3, Sep 1992. p 29.

3. *The physiology of nerve cells,* D H Paul, Blackwell scientific publications, 1975. p 1 - 9

4. *Human Anatomy and Physiology*, R Carola et al, McGraw-hill, 1992. p 334.

5. *Human Anatomy and Physiology*, R Carola et al, McGraw-hill, 1992. p 335.

6. *Human Anatomy and Physiology*, R Carola et al, McGraw-hill, 1992. p 335.

7. *Human Anatomy and Physiology*, R Carola et al, McGraw-hill, 1992. p 348 - 350.

8. *The biology of the brain, from neurons to networks: readings from scientific American,* R R Llinas (editor),W.H. Freeman, 1989. p 7 - 9.

9. *Neuro-physiology,* R H S Carpenter, Edward Arnold, 1990. p 40 - 42.

10. *The biology of the brain, from neurons to networks: readings from scientific American,* R R Llinas (editor),W H. Freeman, 1989. p 7 - 9.

11. *Foundations of neuroscience,* M Jacobson, Plenium Press, 1993. p 218 - 225.

12. *Human Anatomy and Physiology*, R Carola et al, McGraw-hill, 1992. p 346.

13. *The biology of the brain, from neurons to networks: readings from Scientific American,* R R Llinas (editor),W H. Freeman, 1989. p 20 - 33.

14. *Human Anatomy and Physiology*, R Carola et al, McGraw-hill, 1992. p 346 - 347.

15. *Neural Computing: Theory and Practice,* P D Wasserman, Van Nostrand Reinhold, 1989. p 167 - 168.

16. *Neurology*, C N Martyn, Churchill livingstone, 1989. p 9 - 48.

17. *Brain's diseases of the nervous system,* J N Walton (ed), Oxford University Press, 1993. p 26 - 32.

18. *Human Anatomy and Physiology*, R Carola et al, McGraw-hill, 1992. p 362 - 368.

19. *Human Anatomy and Physiology*, R Carola et al, McGraw-hill, 1992. p 394.

20. *The nervous system,* C R Noback & R J Demarest, McGraw-hill, 1986. p 208 - 220.

21. *Human Anatomy and Physiology,* R Carola et al, McGraw-hill, 1992. p 414 - 415.

22. *Specialisation of the human brain: the workings of the brain, development, memory and perception: Readings from Scientific American,* R R Llinas (editor), W H. Freeman, 1990, p 105 - 120.

23. *Mind and Brain,* G D Fischbach, Scientific American, Vol 27, 3, Sep 92. p 63 - 71.

24. *The brain: The last frontier,* R M Restak, Warner Books, 1979.

Chapter 3.

1. *Artificial Neural Networks: Foundations, Paradigms, Applications and Implementations,* P K Simpson, Pergamon Press, 1990.

2. *A logical calculus of the ideas immanent in nervous activity,* W McCulloch & W Pitts, Bulletin of mathematical biophysics, Vol 5, 1943. p 115 - 137.

3. *Neurocomputing,* R Hecht-Nielsen, Addison Wesley, 1989. p 14.

4. *How we know universals,* W Pitts & W McCulloch, Bulletin of mathematical biophysics, Vol 9, 1947. p 127 - 147.

5. *Perceptrons,* M L Minsky & S A Papert, MIT Press, 1989. p viii - ix.

6. *Fundamentals of Neural Networks: Architectures, Algorithms and Applications,* L Fausett, Prentice Hall, 1994. p 22.

7. *The Organisation of Behaviour,* D Hebb, Wiley, 1949.

8. *Neurocomputing,* R Hecht-Nielsen, Addison Wesley, 1989. p 15.

9. *The Principles of Neurodynamics,* F Rosenblatt, Spartan books, 1961.

10. *Fundamentals of Neural Networks: Architectures, Algorithms and Applications,* L Fausett, Prentice Hall, 1994. p 23.

11. *Perceptrons,* M L Minsky & S A Papert, MIT Press, 1969 (new edition 1989).

12. *Neurocomputing,* R Hecht-Nielsen, Addison Wesley, 1989. p 16 - 18.

13. *Fundamentals of Neural Networks: Architectures, Algorithms and Applications,* L Fausett, Prentice Hall, 1994. p 25.

14. *Neural Computing: Theory and Practice,* P D Wasserman, Van Nostrand Reinhold, 1989. p 12 - 17.

15. *Neurocomputing,* R Hecht-Nielsen, Addison Wesley, 1989. p 4

16. *An Introduction to Neural Nets,* J McShane, Hewlett Packard Journal, Vol 43, 1, Feb 1992. p 63.

17. *Artificial intelligence,* Pratt, Macmillan, 1994. p 216 - 224.

18. *Neural Computing: Theory and Practice,* P D Wasserman, Van Nostrand Reinhold, 1989. p 30 -37. (see also Pratt above).

19. *Neural computing: an Introduction,* R Beale & T Jackson, Adam Hilger, 1990. p 83 -89.

20. *Foundations of neural networks,* T Khanna, Addison Wesley, 1990. p 58 - 74.

21. *On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition,* A N Kolmogorov, Doklady Akademii SSSR, Vol 144 p 679 - 681, 1963. (American Mathematical Society translation - 28: 55 - 59)

22. *Theory of adaptive pattern classifiers,* Amari, IEEE Transactions on electronic computers, EC 16, 1983. p 299 - 307.

23. *Beyond regression: New tools for prediction and analysis in the behavioural sciences,* P J Werbos, Masters thesis, 1974, Harvard University.

24. *Learning logic,* D B Parker, Invention report s81 64, office of technology licensing, 1982, Stanford University.

25. *Learning internal representations by error propagation,* Rumelhart et al, Parallel distributed processing, Vol 1, 1986. p 318 - 362.

26. *Artificial Neural Networks: Foundations, paradigms, Applications and Implementations,* P K Simpson, Pergamon Press, 1990. p 114.

27. *How Neural Networks Learn from Experience,* G E Hilton, Scientific American, Vol 27, 3, Sep 1992. p 108.

28. *Neural Computing: Theory and Practice,* P D Wasserman, Van Nostrand Reinhold, 1989. p 56 -59.

29. *Neural Computing: Theory and Practice,* P D Wasserman, Van Nostrand Reinhold, 1989. p 77 - 92.

30. *Neural networks and physical systems with emergent collective computational abilities,* J Hopfield, Proceedings of the national academy of science, vol 79, 1982. p 2554 - 2558.

31. *Neural Computing: Theory and Practice,* P D Wasserman, Van Nostrand Reinhold, 1989. p 93 - 112.

32. *Artificial intelligence,* Pratt, Macmillan, 1994. p 234 - 243.

33. *Absolute stability of global pattern formation and parallel memory storage by competitive neural networks,* M A Cohen & S G Grossberg, IEEE transactions on systems, man and cybernetics, Vol 13, 1983. p 815 - 826.

34. *Neural Computing: Theory and Practice,* P D Wasserman, Van Nostrand Reinhold, 1989. p 113 - 125.

35. *Neural Computing: Theory and Practice,* P D Wasserman, Van Nostrand Reinhold, 1989. p 220 -221, p 212 - 214, p 61 - 75.

36. *Neural Computing: Theory and Practice,* P D Wasserman, Van Nostrand Reinhold, 1989. p 127 - 149.

37. *Neural Computing: Theory and Practice,* P D Wasserman, Van Nostrand Reinhold, 1989. p 167 - 188.

38. *An Introduction to Neural Computing* I Aleksander & H Morton, Chapman Hall, 1990.

39. *A practical device to simulate the working of nervous discharges,* S Russell, The Journal of Animal Behaviour, Vol 3: p 15, 1913.

40. *Analogue neural VLSI: a pulse stream approach,* A Murray & L Tarassenko, Chapman Hall, 1994.

41. *Digital Neural Networks,* S Y Kung, Prentice Hall, 1993.

42. *Digital design using field programmable gate arrays,* P K Chan & S Mourad, Prentice Hall, 1996.

43. *The programmable gate array data book,* XILINX (company data book), 1996.

44. *Neural Computing: Theory and Practice,* P D Wasserman, Van Nostrand Reinhold, 1989. p 151 - 166.

45. *Neural Computing: Theory and Practice,* P D Wasserman, Van Nostrand Reinhold, 1989. p 110.

Chapter 4.

1. *Darwin a life in Science,* M White & J Gribbin, Simon and Schuster, 1988. p 27 - 31.

2. *Novum Organum,* Frances Bacon, 16th century (out of print).

3. *Zoonomia,* E Darwin, 1794 (out of print).

4. *Darwin a life in Science,* M White & J Gribbin, Simon and Schuster, 1988. p 34 - 47.

5. *Evolution,* A Panchen, Bristol Classical Press, 1993. p 14 - 21.

6. *The origin of species by means of Natural Selection, or the preservation of favoured races in the struggle for life,* C Darwin, (republished by Penguin Press), 1994

7. *The theory of evolution,* J M Smith, Pelican books, 1975. p 55 - 63.

8. *Evolution,* A Panchen, Bristol Classical Press, 1993. p 102 - 116, 147 - 158 .

9. *Evolution,* A Panchen, Bristol Classical Press, 1993. p 93 - 94 .

10. *Evolution,* J M Savage, Holt, Reinhart and Winston, 1977. p 128 - 131.

11. *Adaptation and Natural Selection,* G C Williams, Princeton University Press, 1966.

12. *The Selfish Gene,* R Dawkins, Oxford University Press, 1976.

13. *Animal dispersion in relation to social behaviour,* V C Wynne-Edwards, Coliver-Boyd, 1962.

14. S J Gould & N Eldredge, Paleobiology 3, 1977. p 115 - 51.

15 *The Blind Watchmaker,* R Dawkins, Penguin books, 1991.

16. *Living Embryos: An Introduction to the study of animal development,* J Cohen, Pergamon Press, 1967. p 1 - 3, (book as a whole gives overview of embryology).

17. *Living Embryos: An Introduction to the study of animal development,* J Cohen, Pergamon Press, 1967. p 3 - 24.

18. *Embryonic and foetal development,* Edited C R Austin & R V Short, Cambridge, 1972. p 2.

19. *Embryonic and foetal development*, Edited C R Austin & R V Short, Cambridge, 1972. p 10.

20. *Living Embryos: An Introduction to the study of animal development*, J Cohen, Pergamon Press, 1967. p 22 - 24.

21. *The Penguin book of the natural world*, editor E Martin et al, Penguin Books. p 8 - 10, (no publishing date specified).

22. *Neurophysiology*, R H S Carpenter, Edward Arnold, 1990. p 1 - 8.

23. *The brain: The last frontier*, R M Restak, Warner Books, 1979. p 50 - 52.

24. *Principles of neural science*, E R Kandel et al, Prentice Hall, 1991. p 296 - 308.

25. *Neurophysiology*, R H S Carpenter, Edward Arnold, 1990. p 6.

Chapter 5.

1. *Applied Automata Theory*, (Edited) J T Tou, Academic Press, 1968. p 217 - 218.

2. *Adaptation in natural and artificial systems*, J H Holland, Ann Arber: The university of Michigan press, 1975.

3. *Genetic algorithms in search, optimisation & machine learning*, D E Goldberg, Addison - Wesley, 1989. p 10 - 18.

4. *Evolutionary Artificial Neural Networks*, X Yao, Int Journal of Neural Systems 4, 3 (Sep 93), World Scientific pub co, 1993.

5. *Practical Genetic Algorithms*, R L Haupt & S E Haupt, Wiley, 1998. Ch2

6. *Practical Genetic Algorithms*, R L Haupt & S E Haupt, Wiley, 1998. Ch3

7. *Genetic algorithms in search, optimisation & machine learning*, D E Goldberg, Addison - Wesley, 1989. p 147 - 213.

8. *Combinations of Genetic Algorithms and Neural Networks: A Survey of the state of the Art*, J D Schaffer et al, COGANN -92 (conf proc), IEEE comp soc press, 1992, p 1 - 37.

9. *Evolutionary Artificial Neural Networks*, X Yao, Int Journal of Neural systems, 4, 3 (Sep 93), World Scientific Publishers, 1993, p 203 - 222.

10. *Hybridising the Genetic Algorithm and the K nearest neighbours classification algorithm,* J D Kelly & L Davis, INCGA - 91 (conf proc), Kaufmann, 1991, p 377 - 383.

11. *Using genetic search to exploit the emergent behaviour of neural networks,* J D Schaffer et al, Emergent computation, North Holland, 1990, p 244 - 248.

12. *Designing neural network explanation facilities using Genetic Algorithms,* R C Eberhart & R W Dobbins, IEEE joint conf on Neural networks, IEEE, 1991, p 1758 - 1763.

13. *Genetic Algorithms and Neural Networks: An Introduction,* R Reed & R J Marks, Northcon - 92 (conf Proc), Ventura, 1995, p 293 - 301.

14. *Training feedforward neural networks using Genetic Algorithms,* D J Montana & L Davis, int joint conf on AI (conf Proc), Kaufmann, 1989. p 762 - 767.

15. *A Genetic cascade correlation learning algorithm,* M A Potter, COGANN - 92 (conf proc), IEEE comp soc press, 1992.

16. *Parametric connectivity: Feasibility of learning in constrained weight space,* T P Caudell, int joint conf on Neural Networks (conf proc), Hillsdale, 1990. p 667 - 675.

17. *Genetic generation of both weights and architecture for a neural network,* J R Koza & J P Rice, int joint conf on Neural Networks (conf proc), IEEE, 1991, p 397 - 404.

18. *Genetic sparse distributed memory,* R Das & D Whitley, COGANN - 92 (conf proc), IEEE comp soc press, 1992.

19. *The Blind Watchmaker,* R Dawkins, Penguin books, 1991 edition. p 51 - 74, p 327 - 334.

20. *The Blind Watchmaker* (software package - manual), R Dawkins, 2nd edition, SPA software, PO Box 59, Tewkesbury.

21. *On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition,* A N Kolmogorov, Doklady Akademii SSSR, Vol 144 p 679 - 681, 1963. (American Mathematical Society translation - 28: 55 - 59)

Chapter 6

1. *Combinations of Genetic Algorithms and Neural Networks: A survey of the state of the art,* Schaffer et al, COGANN -92, IEEE, 1992. p 1 - 37.

2. *Genetic algorithms and neural networks an introduction,* R Reed et al, NORTHCON -92, Ventura, 1992. p 293 - 301.

3. *A review of Evolutionary neural networks,* X Yao, International journal of intelligent systems, Vol 8, Pt 4, 1993, p 539 - 567.

4. *On optimising large neural networks (multilayer Perceptrons) by learning and evolution,* H Braun, Zeitschrift fur angewandte mathematik und mechanik, Vol 76, No 1, 1996, p 211 -214.

5. *Evolving neural network controllers for unstable systems,* A P Wieland, IJCNN -91, IEEE, 1991, p 667 - 673.

6. *Neural Computing: Theory and Practice,* P D Wasserman, Van Nostrand Reinhold, 1989. p 127

7. *Neural dynamics of category learning and recognition,* G A Carpenter & S Grossberg, Brain structure, learning and memory (AAA symposium), 1986.

8. *A massively parallel structure for a self organised neural pattern recognition machine,* G A Carpenter & S Grossberg, Computer vision, Graphics and image processing, 37, 1987. p 54 - 115.

9. *ART - 2: self organisation of stable category recognition codes for analog input patterns,* G A Carpenter & S Grossberg, Applied optics, 26. 1989. p 4919 - 4930.

10. *High speed learning in a supervised, self-growing net,* D Palmer - Brown, Artificial Neural Networks, 2, Elsevier, 1992. p 1159 - 1162.

11. *FALCON: A Fuzzy adaptive learning control network,* C T lin, NAFIPS / IFIS / NASA ‘ 94., 1994. p 228 - 232.

12. *Reinforcement learning for ART - Based Fuzzy Adaptive Learning Control Networks,* C J Lin & C T lin. Proceedings of the 1995 int conf on fuzzy systems, IEEE. 1995. p 1299 - 1306.

13. *Neural Computing: Theory and Practice,* P D Wasserman, Van Nostrand Reinhold, 1989. p 127 - 149.

14. *Artificial Neural Networks: Foundations, paradigms, Applications and Implementations,* P K Simpson, Pergamon Press, 1990. p 36 - 46.

15. *GAL: Networks that grow when they learn and shrink when they forget,* E Alpaydin, International Journal of Pattern Recognition, Vol 8, 1. 1995 p 391 - 14.

16. *A proposal for more powerful learning algorithms,* E B Baum, Neural Computation, 1, 1989. p 295 - 311.

17. *Incremental supervised learning for mobile robot reactive control,* P Reignier et al, Intelligence autonomous systems int conf, IOS Press, 1995. p 287 - 294.

18. *Comparing distributed and local neural classifiers for the Recognition of Japanese Phonemes,* R Alpaydin et al, Int Joint Conf on Neural Networks (IJCNN '93), 1993. p 239 - 42.

19. *GAL: Networks that grow when they learn and shrink when they forget,* E Alpaydin, International Journal of Pattern Recognition, Vol 8, 1. 1995, p 391 - 14.

20. *Let it grow: A self organising feature map with problem dependent cell structure,* B Fritzke, ICANN '91 Proc of the int conf on Artificial Neural Networks, 1991.

21. *A dynamic neural net to compute convex hull,* A Datta & S K Parui, Neurocomputing, 10, 1996. p 375 - 384.

22. *A dynamic neural net to compute convex hull,* A Datta & S K Parui, Neurocomputing, 10, 1996. p 375 - 384.

23. *Implementing a self development neural network using doubly linked lists,* T C Lee & A M Peterson, Proc of the 13th annual int conf on software and applications, 1989. p 672 - 679.

24. *Cresceptron: A self organising neural network which grows adaptively,* J Weng et al, IJCNN '92, 1992. Vol 1 p 576 - 581.

25. *SPAN: a neural network which grows,* T C Lee & A M Peterson, IJCNN - Int Joint conf on Neural Networks, 1989. Vol 2 p 635 (abstract only).

26. *Dynamic node creation in backpropagation networks,* T Ash, Connection science, 1, 1989. p 365 - 275.

27. *Creating artificial neural networks that generalise,* J Siestma & R J F Dow, Neural networks, 4, 1991. p 67 - 79.

28. *Back propagation algorithm which changes the number of hidden units,* Y Hirose et al, Neural networks, 4, 1991. p 61 - 66.

29. *A growing network which optimises between undertraining and overtraining,* G Chakraborty, IEEE con on Neural Networks, 1995. p 1116 - 1120 vol 2.

30. *Growing non-uniform feedforeward networks for continuous mappings,* V V Vinod & S Ghoso, Neurocomputing, 10, 1996. p 55- 69.

31. *Asymptotic inferential capabilities of feed-foreward neural networks,* E Ferran & R Perazzo, Europhysics letters, 14, 2. 1993, p 175 - 180

32. *Learning in feedforward layered networks: the tiling algorithm,* M Mezard & J P Nadal, Journal of physics, A22, 1989. p 4 - 22.

33. *Single layer learning revisited: A stepwise procedure for building and training a neural network,* S Knerr et al, Neuro-computing Algorithms, Architectures and Applications, Springer, 1989.

34. *The cascade correlation architecture,* S E Fahlman & C Lebiere, Advances in Neural information processing systems, Morgan-Kaufmann, 1990. p 532 - 532. Vol 2.

35. *Evolutionary design of application tailored Neural Networks,* Z Obradovic & R Srikumar, IEEE con on Neural Nets, 1994. p 284 - 289.

36. *Dynamic structure adaptation in feedforeward neural networks - An example of plant monitoring,* R Kozma & M Kitamura, IEEE con on Neural Networks, 1995. p 622 - 627 vol 2.

37. *Skeletonization: A technique for trimming the fat from a network via relevance assessment,* M C Mozer & P Smolensky, Connection science, vol 1, 1989. p 3 - 26.

38. *A simple procedure for pruning backpropagation trained neural networks,* E D Karnin, IEEE trans. Neural Networks, 1, 1990. p 239 - 242.

39. *A simple procedure for pruning back propagation trained neural networks,* E D Karin, IEEE trans. on Neural Networks, vol 1, 1990. p 239 - 242.

40. *Pattern specific Neural Network design,* M Anderle et al, Journal of statistical physics, 81, 3 / 4. 1995. p 843 - 9.

41. *Automatic growing of a Hopfield style neural network for classification of patterns,* R F Brouwer, Image processing and its applications, IEE, 1995. p 637 - 641.

42. *A modified Hopfield Network for two-dimensional module placement,* M Sriram & S M Kang, IEEE int symposium on circuits and systems, 1990. p 1664 - 1667 Vol 3.

43. *An Introduction to Autosophy and Self-Growing Networks,* K Holtz, Northcon '90, 1991. p 160 - 165.

44. *Learning and evolution in neural networks,* S Nolfi et al, Adaptive behaviour, vol 3, 1, 1994, MIT press. p 5 -28.

45. *Pre-processing network for multiple binary objects,* S D You & G E Ford, Proc of the int joint conf on Neural Networks, 1993. vol 3, p 2177 - 2180.

46. *Incremental Evolution of Neural Network Architectures for Adaptive Behaviour,* D Cliff et al, ESANN '93 , 1993. p 39 -44.

47. *Growing adaptive neural networks with graph grammars,* S M Lucas, ESANN ' 95, 1995. p 235 - 240.

48. *A tree structured adaptive network for function approximation in high dimensional spaces,* T D Sanger, IEEE trans. on neural networks, vol 2, 2. March 1991. p 1 - 20.

49. *SplitNet: Learning of tree structured Kohonen Chains,* J Rahmel, Neural Networks int Conf, 1996. Vol 2 p 1221 - 1226.

50. *A self organising neural tree architecture,* L Y Fang et al, ACNN '92 Proc on the third Australian conf on neural networks, 1992. p 126 - 129.

51. *Let it grow: A self organising feature map with problem dependent cell structure,* B Fritzke, ICANN '91 Proc of the Int Conf on Artificial Neural Networks, 1991. p 324 - 334.

52. *Growing and pruning neural tree networks,* A Sandra, R J Mammon, IEEE transactions on computers, vol 42, 3, 1993. p 291 -299.

53. *Neural trees - using neural nets in a tree classifier structure,* J E Stromberg et al, ICASSP '91, 1991. vol 1 p 137 - 140.

54. *Growing and pruning neural tree networks,* A Sandra, R J Mammon, IEEE transactions on computers, vol 42, 3, 1993. p 291 -299.

55. *CAM_BRAIN: The evolutionary engineering of a billion neuron artificial brain by 2001 which grows / evolves at electronic speeds inside a cellular automata machine,* H de Garis, Neuroinformatics and Neurocomputers, 1995. p 62 - 69.

56. *NEURITE NETWORKS*: *The genetic programming of cellular automata based neural nets which GROW,* H de Garis, International joint conf on Neural Networks (IJCNN 93), IEEE, 1993. vol 3, p 2921 - 2925.

57. *LIZZY: the genetic programming of an artificial nervous system*, H de Garis, ICANN '91 int conf on Artificial Neural Networks, 1991. Vol 2, p 1269 - 1272.

58. *The blind neural network maker: Can we use constrained embryologies to design animat nervous systems.* O. Holland and M. Snaith, Proceedings ICANN 91, 1991. p 1261- 1264.

59. *The use of recursively generated iterated structures to constrain neural network architectures.* O. Holland and M. Snaith, Technical report (available on request), Technology Applications Group, Alnwick England, 1991.

60. *Designing Modular Network Architectures Using a Genetic Algorithm,* B L M Happel & J M L Murre, Artificial Neural Networks 2, Elseviar, 1992. vol 2, p 1215 - 1218.

61. *Hierarchical use of neural techniques in structure damage recognition,* R Ceravolo et al, smart materials and structures, 4, 1995, p 270 - 280

62. *Using Embryology as an Alternative to Genetic Algorithms for Designing Artificial Neural Networks.* C MacLeod & G M Maxwell, ICANNGA '97, 1997. p 361 - 366.

Chapter 7

1. *Neural Networks,* Haykin, MacMillan, 1994. p 577

2. *Using Taguchi Methods to train Artificial Neural Networks,* MacLeod & Maxwell, AI Review, Vol 13, number 3, 1999. p 177 - 184.

3. *Neural Network TOOLBOX*, H. Demuth and M Beale, The math works inc, 1994. p 4.5, A5

4. *Neural Network TOOLBOX*, H. Demuth and M Beale, The math works inc, 1994

5. *Effects of Occam's Razor in evolving Sigma-Pi Neural Nets,* B T Khang, 3rd international conference on evolutionary computation, 1991, p 462 - 471.

6. *Asymptotic inferential capabilities of feed-foreward neural networks,* E Ferran & R Perazzo, Europhysics letters, 14, 2. 1991, p 175 - 180

7. *On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition,* A N Kolmogorov, Doklady Akademii SSSR, Vol 144 p 679 - 681, 1963. (American Mathematical Society translation - 28: 55 - 59)

8. *Foundations of Neural Networks,* T Khanna, Addison Wesley, 1990. p 69 - 74.

9. *An Introduction to Neural Networks,* K Gurney, UCL Press, 1997. p 76.

10. *Neural Computing: Theory and Practice,* P D Wasserman, Van Nostrand Reinhold, 1989. p 56 -59.

11. *Neural Computing,* Anon, Netware, 1993. p NC246

12. *The nervous system - introduction and review,* C R Noback & R J Demarest, McGraw-Hill, 1972. p 294 - 298.

13. *3 - D object recognition from images: a neural network approach,* J J Weng et al, 2nd int conf on image processing, 1992. p 399 - 403.

14. *Dynamic node creation in backpropagation networks,* T Ash, Connection science, 1, 1989. p 365 - 275.

15. *A growing network which optimises between undertraining and overtraining,* G Chakraborty, IEEE con on Neural Networks, 1995. vol 2, p 1116 - 1120 .

16. *Evolutionary design of application tailored Neural Networks,* Z Obradovic & R Srikumar, IEEE con on Neural Nets, 1994. p 284 - 289.

17. *GAL: Networks that grow when they learn and shrink when they forget,* E Alpaydin, International journal of Pattern Recognition, Vol 8, 4. 1995, p 391 - 14.

18. *ART2: self organisation of stable category recognition codes for analog input patterns,* G A Carpenter & S Grossberg, Appl Opt, 1989, Vol 26. p 4919 - 4930.

19. *A growing network which optimises between undertraining and overtraining,* G Chakraborty, IEEE con on Neural Networks, 1995. vol 2, p 1116 - 1120.

20. *Dynamic structure adaptation in feedforeward neural networks - An example of plant monitoring,* R Kozma & M Kitamura, IEEE con on Neural Networks, 1995. vol 2, p 622 - 627.

21. *A simple procedure for pruning backpropagation trained neural networks,* E D Karnin, IEEE trans. Neural Networks, 1, 1990. p 239 - 242.

22. *Skeletonization: a technique for trimming the fat from a network via relevance assessment,* M C Mozer & P Smolensky, Connection science, 1, 1990. p 2 - 26.

23. *Growing non-uniform feedforeward networks for continuous mappings,* V V Vinod & S Ghoso, Neurocomputing, 10, 1996. p 55- 69.

24. *Growing non-uniform feedforeward networks for continuous mappings,* V V

Vinod & S Ghoso, Neurocomputing, 10, 1996. p 55- 69.

25. *A connectionist approach to authorship determination,* L Thirkell & N K Taylor, 19th Int Conf of the association for literary and linguistic computing and the 12th Int Conf on computers and the humanities. 1992, p 223 - 226.

26. *Neural Network TOOLBOX*, H. Demuth and M Beale, The math works inc, 1994. p 11.2 - 11.20

27. *The cascade - correlation architecture,* S E Fahlman & C Lebiere, Advances in neural information processing systems, 1990, Morgan-Kaufmann, vol 2. p 524 - 532.

28. *Automatic growing of a Hopfield style neural network for classification of patterns,* R F Brouwer, Image processing and its applications, IEE, 1995. p 637 - 641.

29. *Incremental Evolution of Neural Network Architectures for Adaptive Behaviour,* D Cliff et al, ESANN '93 , 1993. p 39 - 44.

30. *Artificial Neural Networks - Oscillations, Chaos and Sequence Processing,* Ed L Wang & D Alkon, IEEE, 1993.

31. *Artificial intelligence,* Pratt, Macmillan, 1994. p 237.

32. *Artificial intelligence,* Pratt, Macmillan, 1994. p 235.

33. *Pattern specific Neural Network design,* M Anderle et al, Journal of statistical physics, 81, 3 / 4. 1995. p 843 - 9.

34. *Dynamic node creation in backpropagation networks*, T Ash, Connection science, 1, 1989. p 365 - 275.

35. *Growing and pruning neural tree networks,* A Sandra, R J Mammon, IEEE transactions on computers, vol 42, 3. 1993, p 291 -299.

36. *Principles of anatomy and physiology,* G J Tortora & N P Anagostatios, Harper-Collins, 1989. p 352.

37. *Neurocomputing*, R Hecht-Nielsen, Addison Wesley, 1989. p 12.

38. *Incremental Evolution of Neural Network Architectures for Adaptive Behaviour,* D Cliff et al, ESANN '93 , 1993. p 39 - 44.

39. *Combining Neural Networks and Decision Trees,* A Sandra & R J Mammon, Proc of SPIE , Vol 1469, 1, 1991. p 374 - 382.

40. *Designing Modular Network Architectures Using a Genetic Algorithm,* B L M Happel & J M L Murre, Artificial Neural Networks 2, Elseviar, 1992. vol 2, p 1215 - 1218.

41. *Physiology of the nervous system,* D Ottoson, MacMillan, 1983. p 250 - 252.

Chapter 8

1. *Effects of Occam's Razor in evolving Sigma-Pi Neural Nets,* B T Khang, 3rd international conference on evolutionary computation, 1991, p 462 - 471.

2. *Towards the open ended evolution of neural networks,* S M Lucas, GALESIA 95, 1995, p 388 - 393.

3. *Neural Network TOOLBOX,* H. Demuth and M Beale, The math works inc, 1994, p 11.2 - 11.4

4. *Neural Computing: Theory and Practice,* P D Wasserman, Van Nostrand Reinhold, 1989. p 110 -111.

5. http://www.pcrealm.net/~johuco/n-read.html

6. *An Introduction to Neural Computing* I Aleksander & H Morton, Chapman Hall, 1990. p 112 - 113.

7. *Neural Network TOOLBOX,* H. Demuth and M Beale, The math works inc, 1994, p 11.40 - 11.46

8. *Neural Network TOOLBOX,* H. Demuth and M Beale, The math works inc, 1994, p 11.2 - 11.4

Chapter 9

1. *Neural Computing: Theory and Practice,* P D Wasserman, Van Nostrand Reinhold, 1989. p 77 - 92.

2. *An Introduction to Neural Networks,* K Gurney, UCL Press, 1997. p 71.

3. *Neural Networks,* Haykin, MacMillan, 1994. p 577

4. *Silicon evolution,* A Thompson, Genetic Programming 1996 proceedings of the first international conference, 1996. p 444 - 452.

5. *A primer on the Taguchi Method,* K R Ranjit, Van Nostrand Reinhold, 1990.

6. *An Introduction to Neural Networks,* K Gurney, UCL Press, 1997. p 185 - 188.

Chapter 10

1. *An Introduction to Neural Networks,* K Gurney, UCL Press, 1997. p 49 - 50.

2. *Modular Neural Networks evolved by Genetic Programming,* S B Cho & K Shimohara, IEEE International conference on Evolutionary computation, IEEE, 1996. p 681 - 684.

3. *Designing modular network architectures using a genetic algorithm,* B L M Happel & J M J Murre, Artificial neural networks 2, Elsevier Science, 1992. vol 2, p 1215 - 1218.

4. *Ensemble structure of Evolutionary Artificial Neural Networks,* X Yao & Y Liu, IEEE International conference on Evolutionary computation, IEEE, 1996. p 659 - 664.

5. *Hierarchical use of neural techniques in structural damage recognition,* R Ceravelo et al, Smart materials and structures, IOP publishing, 1994. p 270 - 280.

6. *Gene branches out in big way,* Anon, New Scientist, 2063. 4 Jan 1997. p 17.

7. *The Fractal Geometry of Nature,* B B Mandelbrot, W H Freeman & Co, Revised Edition, 1983.

8. *Biomorphic Mitosis,* D Stuedell, Computers and graphics, Vol 15, 3, 1991 p 455.

9. *NEURITE NETWORKS*: *The genetic programming of Cellular automata based neural nets which GROW,* H de Garis, International joint conf on Neural Networks (IJCNN 93), IEEE, 1993. vol 3, p 2921 - 2925.

10. *From Animals to Animats,* yearly international conferences

11. *The biological bases of behaviour,* Ed. N Chalmers et al, Open University Press, 1971. p 35.

12. *Principles of neural Science,* E R Kendel, 1991, Appelton and Lange, 1991. p 24, 26, 31.

13. *Neuroscience,* M F Bear, Williams and Willkins, 1996. p 366 - 373.

14. *Principles of neural Science,* E R Kendel, Appelton and Lange, 1991. p 582 - 595.

15. *Principles of neural Science,* E R Kendel, Appelton and Lange, 1991. p 24.

16. *Fundamental Neuroscience*, D E Haines, Churchill- livingstone, 1996, p 339.

17. *Annelids*, R P Dales, Huchinson, 1963. p 110 - 135.

18. *Annelids*, R P Dales, Huchinson, 1963. p 125.

19. *Animal Locomotion*, J Grey, Weidenfield and Nicolson, 1968, p 42 - 43, p 151 - 155.

20. *Animals without backbones*: 1, R Buchsbaum, Pelican 1951. p 93.

21. *The invertebrates*, R N Alexander, Cambridge university Press, 1979. p 125 - 142.

22. *The invertebrates: a new synthesis*, R S K Barnes et al, Blackwell Scientific, 1988, p 502 - 503.

23. *Animal Locomotion*, J Grey, Weidenfield and Nicolson, 1968, p 154.

24. *Fuzzy controllers*, Reznik, Newnes, 1997. p 59 - 103.

25. RGU internal paper: Comparing Fuzzy logic with ANNs and digital systems, 1997

26. *Silicon Evolution*, A Thompson, Proc of Genetic Programming '96 (Stanford). 1996, p 444 - 452.

27. *Principles of neural Science*, E R Kendel, Appelton and Lange, 1991. p 582 - 595.

28. *Fundamental Neuroscience*, D E Haines, Churchill- livingstone, 1996, p 341 - 346.

29. *Human Anatomy and Physiology*, A P Spence and E B Mason, West, 1992. p 390 - 391.

30. *Hierarchical use of neural techniques in structure damage recognition*, R Ceravolo et al, smart materials and structures, 4, 1995, p 270 - 280

31. *XneuroGene: A system for evolving artificial neural network*, C Jacob et al, MASCOTS '95, 1995, p 436 - 439.

32. *The Physiology of fishes*, Ed M E Brown, Academic press, 1957. p 14.

33. *Evolution*, M W Strickborger, Jones and Bartlett. 1996, p 37.

Chapter 11

1. *Neural Network TOOLBOX*, H. Demuth and M Beale, The math works inc, 1994. p 11.40 - 11.46.

2. *Neural network design based on evolutionary programming*, J Fang & Xi, Artificial Intelligence in engineering, Vol 11, 1997. p 155 - 161.

3. *Neural Network TOOLBOX*, H. Demuth and M Beale, The math works inc, 1994. p 11.32 - 11.39.

4. *Silicon evolution*, A Thompson, Genetic Programming 1996 proceedings of the first international conference, 1996. p 444 - 452.

5. *Practical Genetic Algorithms*, R L Haupt & S E Haupt, Wiley, 1998. p 1 - 23.

6. *Silicon evolution*, A Thompson, Genetic Programming 1996 proceedings of the first international conference, 1996. p 444 - 452.

Chapter 12

1. *http://www.honda.co.jp/home/hpr/e_news/robot/index.html*

2. *Fuzzy controllers*, L Reznik, Newnes, 1997. p 13 - 16.

3. *untitled*, Nature, vol 391, 1997, p 529 - 530, 553 - 558.

4. *A hybrid GP / GA approach for co-evolving controllers and robot bodies to achieve fitness-specified tasks*, W P Lee et al, 0-7803-2902-3, IEEE, 1996. p 384 - 389.

5. *Evolving virtual creatures*, K Sims, Computer graphics annual conf SGGRAPH ‘94, 1994. p 15 - 22.

6. *Silicon evolution*, A Thompson, Genetic Programming 1996 proceedings of the first international conference, 1996. p 444 - 452.