An Infrastructure for Neural Network Construction

A thesis submitted for the degree of Doctor of Philosophy

by

Richard Stewart

Department of Information Systems and Computing, Brunel University,
Uxbridge, Middlesex, UB8 3PH, United Kingdom.

15th September 2005

Abstract

After many years of research the area of Artificial Intelligence is still searching for ways to construct a truly intelligent system. One criticism is that current models are not 'rich' or complex enough to operate in many and varied real world situations. One way to tackle this criticism is to look at intelligent systems that already exist in nature and examine these to determine what complexities exist in these systems and not in the current AI models.

The research begins by presenting an overview of the current knowledge of Biological Neural Networks, as examples of intelligent systems existing in nature, and how they function. Artificial Neural networks are then discussed and the thesis examines their similarities and dissimilarities with their biological counterparts. The research suggests ways that Artificial Neural Networks may be improved by borrowing ideas from Biological Neural Networks.

By introducing new concepts drawn from the biological realm, the construction of the Artificial Neural Networks becomes more difficult. To solve this difficulty, the thesis introduces the area of Evolutionary Algorithms as a way of constructing Artificial Neural Networks.

An intellectual infrastructure is developed that incorporates concepts from Biological Neural Networks into current models of Artificial Neural Networks and two models are developed to explore the concept that increased complexity can indeed add value to the current models of Artificial Neural Networks. The outcome of the thesis shows that increased complexity can have benefits in terms of learning speed of an Artificial Neural Network and in terms of robustness to damage.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to dedicate this thesis to Anne Jackson, for her faith and perseverance.

I would also like to thank my supervisors and the Dept of Information Systems for their help and encouragement.

1

Thesis Introduction

What a piece of work is man! How noble in reason! how infinite in faculty! in form, in moving, how express and admirable! in action how like an angel! in apprehension how like a god! the beauty of the world! the paragon of animals! (Shakespeare, 1601)

1.1 Introduction

The ultimate long-term goal of artificial intelligence is to a produce an 'intelligent' agent. The reasons for this are many and varied but two primary reasons are:

1. to understand our own intelligence by producing cognitive models
2. and to produce efficient and flexible computer systems through the use of the traits that make such intelligent agents efficient and flexible

In any situation where computers are used, from factory floors, to controlling aircraft, to searching the Internet, a degree of flexibility, independence and robustness is useful which is lacking in many of today's computational systems. Many different approaches have been taken towards rectifying these issues, including rule-based systems, mathematical simulations, and symbolic manipulation frameworks. One approach has been to investigate and analyse intelligent systems that already exist in nature where it is possible

to easily see such systems in action. Nearly all animals have some form of intelligence, ranging from simple food searching skills and predator defence in invertebrates, such as ants, to more complex forms of intelligence as can be attributed to humans. At present the flexibility and repertoire of skills attributed to even rudimentary creatures such as ants, are beyond our skills to duplicate. In order to understand how these types of systems may work it is necessary to understand how biological systems exhibiting intelligence operate and function.

A common theme that runs through all these natural intelligent systems is that each has a nervous system of varying degrees of complexity. Each nervous system consists of a large number of interconnected processing units (neurons), working together in a form of distributed, parallel processing. This type of processing structure has been simulated in a computer environment with success and is variously termed *connectionism*, *neural networks*, or *parallel distributed processing* (Rumelhart & McClelland, 1986; Hecht-Nielsen, 1990; Beale & Jackson, 1992; Aleksander & Morton, 1995; Principe, Euliano, & Lefebvre, 2000; Haykin, 1999).

Models using neural networks offer a number of advantages over traditional algorithmic processing techniques:

- *The potential of massively parallel computational structures*

  Each processing unit can be implemented in very simple processors allowing many processing calculations to be carried out in parallel.

- *Robustness to noisy input*

  Many real world applications rely on information gained from sensors gathering information from the surrounding world. This information can often be degraded by noise and it is important to have applications that can operate consistently in varying situations.

- *Graceful degradation when processing components fail*

  In traditional serially processed programs, if a component fails or is corrupted, the

entire program is compromised and no useful work can be achieved. But if a processing unit fails or is corrupted in a neural network, the network can often produce useful output though possibly of not as high quality or accuracy.

- *The ability to change, learn and generalise from examples*

  Most networks have generally the same structure no matter what the task to be solved is. Automatic learning routines can be used to change the fine structure of the network based on a number of examples; no thought by the implementor of the system of how the task is to be solved is required. Serially processed programs generally require the programmer to explicitly devise and state what algorithms are required to solve the task. The resultant neural network is also able to cope better in exceptional circumstances where the data provided to it may not be present in its normal operational range and can often produce a 'best guess' of a correct answer. Serial programs are often unable to provide sensible results when working with data outside of their normal operating range.

- *The resemblance to biological networks*

  For many years debates have been made as to how cognitive systems in humans, and other animals, operate. Symbolic processing techniques have often produced insightful ideas on the 'functional' aspects of our cognitive abilities but often have left the 'physical' aspects in the dark. By mimicking biology more closely works, neural networks can illuminate aspects of both the functional side of the system and also its physical side.

1.2 Artificial Neural Networks

It has been argued by Gardner (1993) that neural networks have gone through at least two generations of development. First generation networks, typified by the perceptron (Rosenblatt, 1958), were loosely based on early concepts of biological neuron functioning. These networks had two groups of neurons only, an input and an output group, each consisting of a number of processing units where each unit in the input group was connected to each unit in the output group. The groups are often termed *layers*, due to the

fact that when depicted the processing units are often shown as a single column or row of units. The processing units, or *nodes*, summed and transformed their input signal to produce a binary or graded output signal. The 'strength' of a connection, also called the *synaptic weight*, between any two nodes can be adjusted in its ability to 'pass on' a signal from the transmitting node to the receiving node. By adjusting the connection strengths between the input and output layer, the input patterns, encoded as activity on the input layer nodes, can be transformed into the output pattern.

Although useful for solving simple pattern recognition problems, it was found that certain classes of problems were unsolvable by these types of networks (e.g. non linearly separable problems of the XOR nature). To remedy this, a second generation of neural networks were developed that included one or more hidden layers that 'resided between', and connected, the input and output layers. This increase in complexity lead to the development of more complex learning rules to train the networks, as typified by the Back Propagation learning rule (Rumelhart & McClelland, 1986).

Although neural networks are successful tools for building models, there are still problems with the application of neural networks to certain classes of problems. Gardner (1993) claims that this could be the result of the current generations of artificial neural network being relatively simple and unsophisticated when compared with neural networks found in nature and are thus lacking in what Gardner calls *neuromorphism*, or realism of neuron functioning and morphology. Evidence of this is that the whole area of neural networks has been investigating increasingly more complex networks as time goes on in order to enhance the networks functional capabilities. This trend for increased complexity suggests that at present either, the models being produced are too simple and lacking the depth of complexity of their biological counterparts, or the entire rationale of neural networks is flawed.

It may be that it is incorrect to suggest that all of the problems of neural networks will be solved by looking and incorporating more elements of biology into the models. But it has been shown that when a neural network is trained to solve a neurobiological problem, the hidden units of the network are seen to perform similar functions as the interneurons of

their biological counterparts (Lockery & Sejnowski, 1993), and so suggests that current models are in the right 'ball-park'.

This issue of lack of neuromorphology can be broken down in the following ways:

- *Lack of Scalability/Size*

  The techniques used to design network architectures at the moment restrict the size of an artificial network to a size that is typically many orders of magnitude smaller than those found in higher order mammals. The human brain has approximately between 10^{11} and 10^{12} neurons, with each neuron having between 10^{3} and 10^{4} connections with other neurons. Large scale artificial neural networks have typically between 10^{4} and 10^{5} neurons. This problem of scaling up the size of networks is proving to be difficult. To some extent the problem is related to that of the computational speed of the computer systems carrying out the simulations. But there are promising methods that allow groups of neurons to be processed as a whole as opposed to processing each individual neuron with the group (Mallot & Giannakopoulos, 1996).

- *Lack of Modularity*

  In nature it is often the case that networks are split into distinct areas, with communications channels between them that process different information or perform different functions, rather than a homogeneous network with indiscriminate connections between nodes (Stevens, 1993). This type of structural phenomena is thought to improve the processing speed of the network by restricting the flow of information between each module in the neural system (Zipser, 1989; Ballard, 1990). It has also been suggested that learning purely by example is not feasible for large classes of non-parametric problems, and that neural networks will be able to do no more than fine-tune solutions that are partially hard-wired (Geman, Bienenstock, & Doursat, 1993). Apart from these large scale structures, many neural systems in nature often have smaller scale structures that are repeated many times. Many networks design techniques are addressing the problem of modularity on the large scale but on the small scale repeating structures investigation is only starting.

- *Oversimplification of the processing units*

  Typically when a neural network is being designed, the designer chooses to use a particular type of network node, and each and every node of the network is of this type. When looking at nature it is found that networks are made up from a number of different neuron types. This mixing of neurons within a network has not been thoroughly investigated.

  Linked with this is the question of how each neuron type should function, and what is the algorithmic relationship between its inputs and its outputs. At the moment most neural networks are based on an abstract simplification of those neurons found in the human nervous system. Arguments have been put forward for the use of homogeneous node networks and heterogeneous node networks, and of a half way house by employing neurons with some of the complexities of natural neurons whilst having simplifications in other areas (Segev, 1992). Gardner (1993) suggests that synapses do more than simply transfer a discrete current from one neuron to another. The neuronal state depends on more than the absence or presence of a signal at a point in time, or even an average firing rate over a period of time. Factors which may affect the functioning of a neuron may occur at a number of different levels such as molecular, channel, membrane, cellular and synaptic levels. In artificial systems, this has to some extent been investigated by the use of *Product* nodes (Durbin, Miall, & Mitchison, 1989; Juedes & Balakrishnan, 1998). Juedes and Balakrishnan (1998) showed that the parity problem, even or odd, could be solved with a single node network consisting of a node with the node transfer function $sin(\Sigma_j w_j.i_j + \theta)$. In part this is reflected in that different types of network, based on the types of transfer functions used in the nodes of the network, are employed to perform different tasks more efficiently.

- *Lack of physical properties*

  Neural systems found in nature have physical limitations imposed on them. Each neuron, input connection and output connection has an actual physical size, whereas in artificial systems neurons and connections are regarded as having a point size. Each neuron has a distinct physical location, and the length of connections between

neurons have upper limits, which are again ignored in artificial systems. Although these physical characteristics are ignored in artificial systems, they may have an impact when it comes to modelling accurately neural systems that occur in nature (Mallot & Giannakopoulos, 1996; Segev, 1992). Physical aspects of a network add constraints to the design of the network. Not every neuron in the human brain is connected to every other neuron as is often the case in artificial neural networks. These constraints may help to define the architecture of the network at another level without the details having to be explicitly represented in the genome.

- *Oversimplification of developmental issues*

  In nature the nervous system grows from a small number of cells over a period of time in a physical environment; considerably different from the way that a artificial neural system may be constructed (Rosenzweig, Leiman, & Breedlove, 1999; Purves & Lichtman, 1985). The external environment that a cell finds itself in, such as chemical diffusions and/or proximity to other cells of certain types, can affect many properties of the cell, such as the type of cell it will eventually become. Each cell also has a developmental aspect over time and the processes the cell has undergone can affect the future development of that cell. This spatio-temporal interaction with itself, its peers, and its environment may affect the way a neural network may function, due to the stochastic nature of the growth process.

As well as this initial growth, there will be changes made to the network throughout its life as it learns to deal with new and novel situations. These changes are often small scale changes when compared to the initial developmental growth, and are often conceived as fine-tuning. It is usually the case with artificial neural networks that only one type of learning rule is applied to all of the nodes in the network, throughout their lifetime. It may be that a neural network may function more efficiently if a number of different rules are applied to different areas of the neural net, or at different times in its lifespan.

This thesis will consider some of these aspects with respect to natural and artificial neural networks and investigate how they may be inter-related. In particular the thesis will con-

centrate on the issues of modularity, physical properties and development. These areas will be discussed further in Chapters 2 and 3.

1.3 Designing Neural Networks

The performance and functioning of a neural network on a particular problem is critically dependent on the choice of processing units, the interconnection structure between the units, or topology, and the learning algorithm used to set the connection weights. Designing artificial neural networks that have the correct processing units with the required topology has proved to be an extremely complex task. In many cases the designer of the network simply chooses how many input units, output units and hidden units there will be, what the learning algorithm will be, and what type of network node will be used, based on past experience and experimentation. The designer constructs the general network topology using the chosen processing units and then uses the training algorithm to adjust the connection strengths between the units to achieve the required network function. As a subsequence it may be that human input in the design process may unintentionally bias the network design process producing a sub-optimal network, and in this regard, the design of networks is unsatisfactory (Balakrishnan & Honavar, 1997).

A solution to this is to take the network design out of the designers hands and use a method where the designer specifies what input information is to be used and what output information is required, and an automatic method will design the architecture of the network to fulfil these requirements. The problem can thus be considered to be a multi-objective optimisation problem. One candidate for this is the optimisation method of Genetic Algorithms (GA) which have been shown to be a robust method capable of solving complex problems (Goldberg, 1989; Holland, 1975). In the design of the networks in this thesis GAs will be used to design the networks.

GAs solve a problem by sampling the problem space over many points. This is achieved by having a group, or *population*, of representations of solutions initially scattered randomly throughout the problem space, where each individual in the population represents

one possible solution. The representation is typically a one dimensional string of characters or numbers. Each individual is evaluated with respect to their ability to solve the problem and this is termed the *fitness* of the individual. Individuals are picked to survive to the next generation of the process based on their fitness, those with a better fitness value are more likely to be chosen. This forms the basis of competition or 'survival of the fittest' as proposed in the theory of evolution (Darwin, 1859). In order for the problem space to be explored, two operators can be applied to the chosen individuals, crossover and mutation. During crossover two individuals are selected and selected portions of their representations are swapped or recombined. Mutation takes an individual and changes a small part of the representation to another value. These two operators are sufficient to produce new representations that can be used to explore the problem space. Once crossover and mutation have taken place, the individuals are reevaluated to give their new fitness values and the cycle is repeated. Once the fitness of a representation in the population reaches the user defined criteria, this iterative process ceases, with the 'fittest' representation in the population representing the best solution found.

Genetic Algorithms will be used in this thesis to optimise the structure and parameters of the artificial neural networks. Chapter 4 discusses GAs and their applications to neural networks in greater detail.

1.4 Research Objectives

Although the ultimate long-term goal of this line of enquiry is to produce an intelligent system through the means of an artificial neural network, this is beyond the scope of this thesis. This thesis will concentrate on some of the issues that may be important for the development of this long-term goal and will consider the aspects of modularity, physical nature and development and how they may be inter-related. These issues will be explored using an evolutionary method (Genetic Algorithms).

In addressing these issues, contributions will be made to a number of different research areas, including neural networks and evolutionary computing. These goals will be achieved by:

- Identify the existing structures and elements of biological neural networks and outline the differences with artificial neural networks

- Identify existing approaches, both evolutionary and non-evolutionary, to artificial neural network design and outline their advantages and disadvantages with respect to biological neural networks.

- Identify methods for incorporating more biologically inspired approaches into artificial neural network design.

- Evaluate the suitability of the proposed methods in their application to artificial neural network design by implementing and testing them in an experimental framework using software models and drawing conclusions from the results.

- Propose how the biologically inspired design methods may be utilised and enhanced in future research based on the outcome of the experimentation.

Specifically the following questions will be investigated:

- Does a modular network confer an advantage in training times over a multilayer finds difficult?

- On problems that a multilayer perceptron network finds difficult can a modular network perform more accurately?

- On problems that a multilayer perceptron network finds difficult can a modular network use resources more efficiently?

- When based on a physical metaphor, can the modularity of a network arise spontaneously?

- When based on a physical metaphor, if modularity does arise does the functional modularity (those areas of the network dedicated to processing certain subtasks within the main task) map on to the physical modularity (those areas of the network physically connected together) of the network?

1.5 Structure of thesis

This chapter has outlined the current difficulties in artificial neural network design and describes certain issues that may be of help in designing the next generation of neural networks. The current generations of neural networks at present lack many of the details of their biological counterparts and by introducing some of these details can the networks being developed become more sophisticated processing systems. Specifically by introducing a sense of the physical can this help to design neural network architectures. By using physicality does a modular design emerge and does this improve the performance of the network and does it help to help analyse the resultant structure of the network. By introducing a system whereby the network develops using a set of development rules does this help to design the network and develop the regular structures seen.

Chapter 2 describes biological neural networks, with special attention being given to the structural aspects of biological systems and to the development process. The structure of the neurons is described along with their informational processing role and the classification schemes that are used to categorise neurons. The physical structuring of the biological systems is described along with the stages in the developmental process that created the structures.

Chapter 3 describes artificial neural networks, with emphasis on their design and construction. This chapter starts by describing the different models used to simulate artificial neural networks including those that are used to specifically model biological networks and those that 'borrow' concepts to produce networks capable of solving artificial problems. A simple artificial neural network model is described in order to demonstrate the informational processing capabilities of artificial neural networks. The chapter goes on to outline a variety of methods for constructing neural networks. The chapter describes how artificial neural networks are used to model the physical aspects of biological networks. Finally, observations are made as to the neural networks described as to their relevance in incorporating biological aspects and how it may be that more biological complexity can improve network performance and analysis.

Chapter 4 gives an overview of evolutionary algorithms with emphasis on Genetic Algo-

rithms. The suitability of evolutionary techniques in application to neural network design is discussed and outlines how genetic algorithms can be adapted to the design of artificial neural networks. The concept of fitness is discussed and its relationship to the selection process and survivability. The genetic operators of crossover and mutation are described and how they operate on the population of solutions. Finally the coding schemes for representing artificial neural networks is discussed.

Chapter 5 presents an intellectual infrastructure which will be subsequently used as a basis for building network models.

Chapter 6 describes a series of models that are used to explore the concepts developed from the ideas outlined in Chapters 2, 3 and 4.

Chapter 7 concludes the thesis by summarising what has been achieved and suggests further avenues of research and finishes by highlighting the contributions that have been made in this thesis.

2

Biological Neural Networks

2.1 Introduction

The concept of Artificial Neural Networks (ANNs) has grown out of the research surrounding the investigation of Biological Neural Networks (BNNs) as found in many organisms. In order to understand the background to ANNs and how they differ from BNNs, a knowledge of BNNs is required. This chapter will first describe the physical structures that are evident in BNNs and how they are constructed. A brief discussion on the information processing roles of neurons in BNNs will take place. In all cases the primary reference will be to the cells and structures found in the mammalian nervous system.

2.2 Biological Networks

Two cell types predominate in nervous systems, glial cells and neurons, with glial cells comprising the majority. Santiago Ramon y Cajal (1852-1934) was one of the first to discover that the nervous system was built up from these discrete units. It is currently thought that of the two types of cell, neurons are the more important of the two, due to the generally held view that they are the main providers of information processing within the nervous system (Rosenzweig et al., 1999; Levitan & Kaczmarek, 1997).

Glial cells have not been investigated to the same thoroughness as neurons and consequently less is known of them. A brief description of their function will serve at the

present to put their role into context. Glial cells are thought to serve multi-functional support roles (Rosenzweig et al., 1999), including:

- acting as scaffolding for neuronal migration and axon outgrowth
- regulating neurotransmitters and buffer ions
- segregating groups of neurons and acting as electrical insulators
- providing structural support for neurons
- (possibly) playing a role in information processing and memory storage

A typical individual neuron (see Figure 2.1) consists of a *dendritic tree*, an *axon*, *synapses* and a cell body or *soma*. Dendrites and axons are sometimes grouped together with the term *processes*.

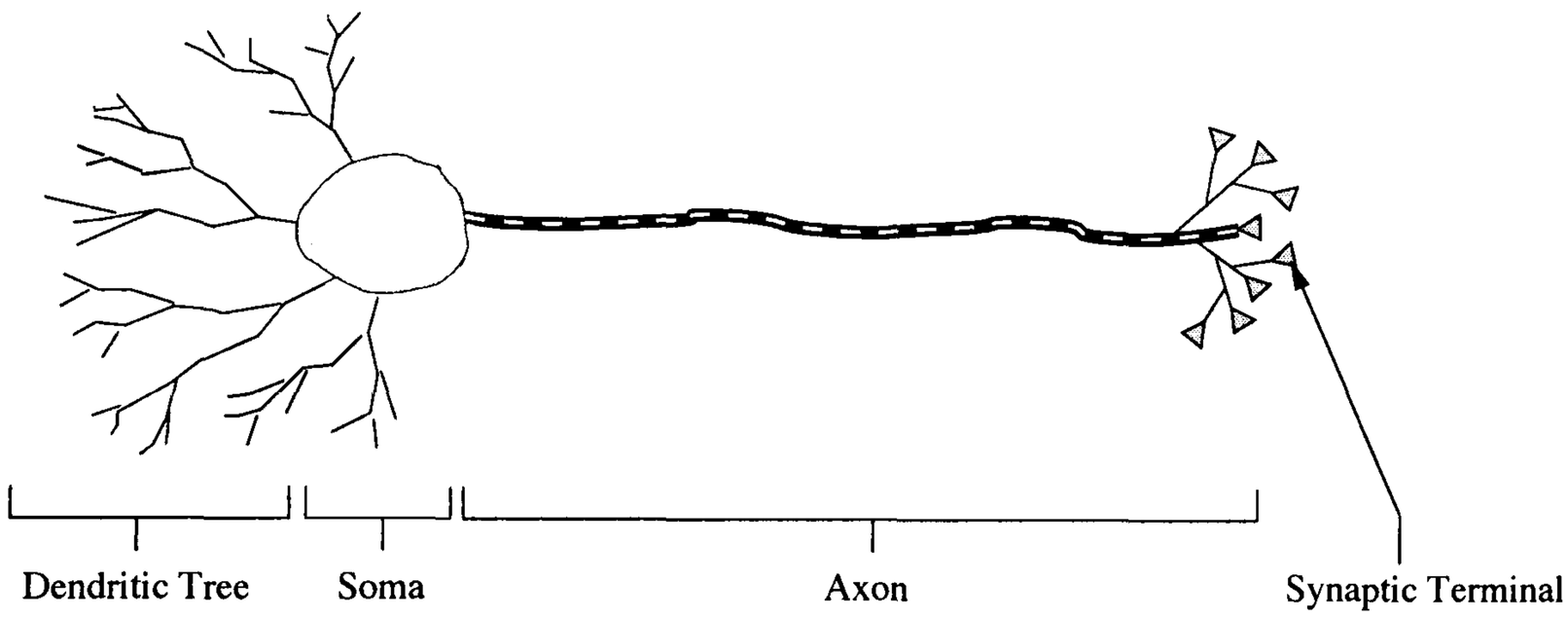


Figure 2.1: Biological neuron

The axon is a thin tube-like extension of the neuron that is attached to the soma and connects the neuron to other neurons. Signals travel from the soma along the axon towards the synaptic terminals. The axon originates at the *axon hillock*, a cone shaped thickening of the soma. The axon can vary in length from a few micrometers to several meters, dependent upon the organism, and is often, though not always, unbranched until just before the synaptic terminals, at which point it may branch many times.

The dendrites tend to be shorter and thicker than the axon. Dendrites tend to branch closer to the soma than the axon, and form a dense network termed the *dendritic tree*. *Dendritic*

spines, projections occurring along the length of the dendrite, form the input sites at which the synaptic terminal from other neurons are attached.

A synapse is the point at which a synaptic terminal from one neuron (the *presynaptic neuron*) is attached to a dendritic spine of another (the *postsynaptic neuron*), and is used to transfer a signal from one to the other. Almost the entire soma and dendritic surfaces of a neuron are covered with dendritic spines. This leads to enormous convergence of information from hundreds or thousands of presynaptic neurons onto a single neuron. From this point of view, neurons can be seen as a mechanism for integrating multiple input signals. The axon can also distribute a signal to many other neurons and this can be viewed as a mechanism for distributing information. A neuron therefore acts as a collector and distributor of information.

Synapses can operate in two major ways, a chemical effect and an electrical effect. Information transfers in chemical synapses are mediated by neurotransmitters. The neurotransmitter is released from the presynaptic neuron axon terminal, diffuses across an extra-cellular space between the two neurons, to the dendritic spine of postsynaptic neuron. Chemical synapses are unidirectional, but it has been found that electrical synapses can form two-way communications channels, allowing both neurons to transmit signals as well as receive them and in this situation each neuron may be either presynaptic or postsynaptic at different times.

In chemical synapses there is a distinct morphological asymmetry of the synapse between the presynaptic and postsynaptic elements of the two connected neurons. But in electrical synapses there is a structural symmetry in the synapse between the presynaptic and postsynaptic elements of the two connected neurons which reflects the bi-directional nature of communication. Although the synapses are symmetrical in structure it is not necessary for there to be symmetry in function. There are examples where the electrical efficacy of the synaptic transmission is higher in one direction than in the other (*rectifying synapses*).

Neurons are usually classed according to one or more characteristics such as shape, size and function. Neurons can be classed into one of three main categories depending upon the structural layout of their component parts; multipolar neurons, bipolar neurons and

monopolar neurons (see Figure 2.2). Multipolar neurons are neurons that have many dendrites and a single axon and this type cell predominates in mammalian nervous systems. Bipolar neurons have a single dendrite at one extreme of the cell and a single axon at the other and this cell type can be usually found in some sensory systems such as the olfactory system and the retina. Monopolar neurons have a single process that extends in two directions perpendicular to the cell body. This type of cell is typically found in the spinal column and is used to transmit sensory information.

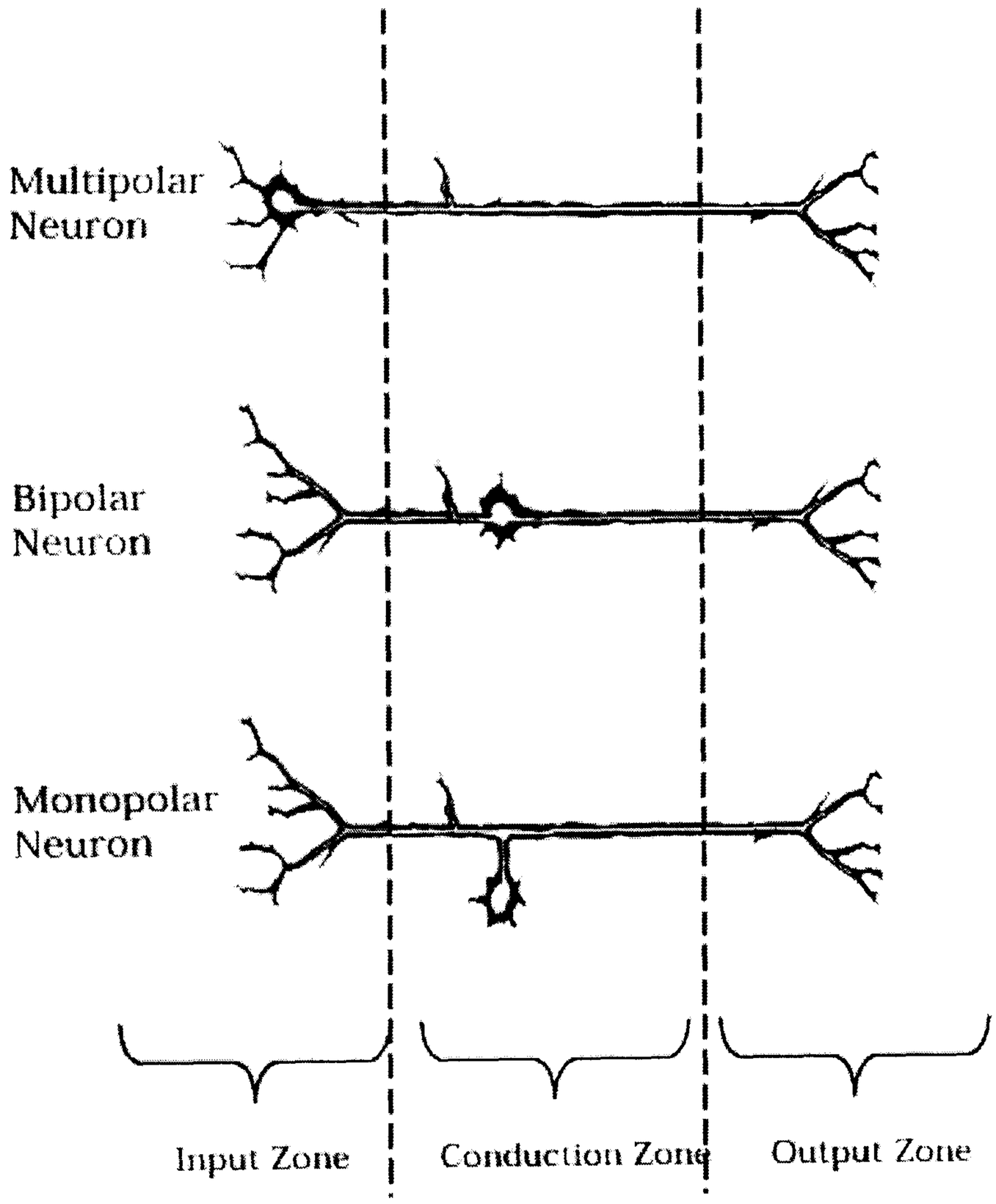


Figure 2.2: Classification of neurons based on structure

Other categories of shape exist that describe the outline shape of the cell, such as *granule* (grain), *spindle*, and *stellate* (star-shaped).

Classification by the function of the neuron also occurs. For instance *motoneurons* are used to make muscles contract or change the activity of a gland. *Sensory neurons* are

used to gather information form the environment. *Interneurons*, which predominate in most nervous systems, are used to process information from other neurons and pass the result on to other neurons.

2.3 Structural Organisation of Biological Neural Systems

The neurons, and glial cells, described so far are the basic building blocks of an organism's nervous system. During development these are "built up" in a predetermined manner, under genetic control, to the final structure of the nervous system. The structure of the nervous system, like the neurons, can be analysed in a number of ways, but are primarily analysed by physical shape and by functional ability.

The physical structures form a range from the whole brain down to the individual neuron with a large number of regular and intricate structures between these two extremes. The nervous system in mammals can be broken down into two main sections the central nervous system, consisting of the brain and spine, and the peripheral nervous system, all other parts other than the brain and spine. Looking specifically at the brain, it is divided into two main sections, the left and right *cerebral hemispheres*, connected together by a number of other structures. Figure 2.3 shows how the different areas of the brain are associated together. Although each structure contains a number of functional abilities, certain functions are associated with some areas more than others. The *basal ganglia* is involved with motor control, the *limbic system* with emotion and learning, the *thalamus* with processing sensory information, the *hypothalamus* with bodily functions such as digestion, temperature regulation and reproductive behaviours, the *cerebellum* with motor coordination, the *pons* with integrating sensory and motor information, and the *medulla* with regulation of bodily functions such as breathing and heart rate. Each area is distinctly different looking anatomically, and this is one of the reasons such divisions have been chosen. It should be noted that this is not the only way to categorise the physical structures of the brain.

The *cerebral cortex* serves many functions, including those considered to be "higher cortical functions", which include reasoning, planning, and language. The neurons of the

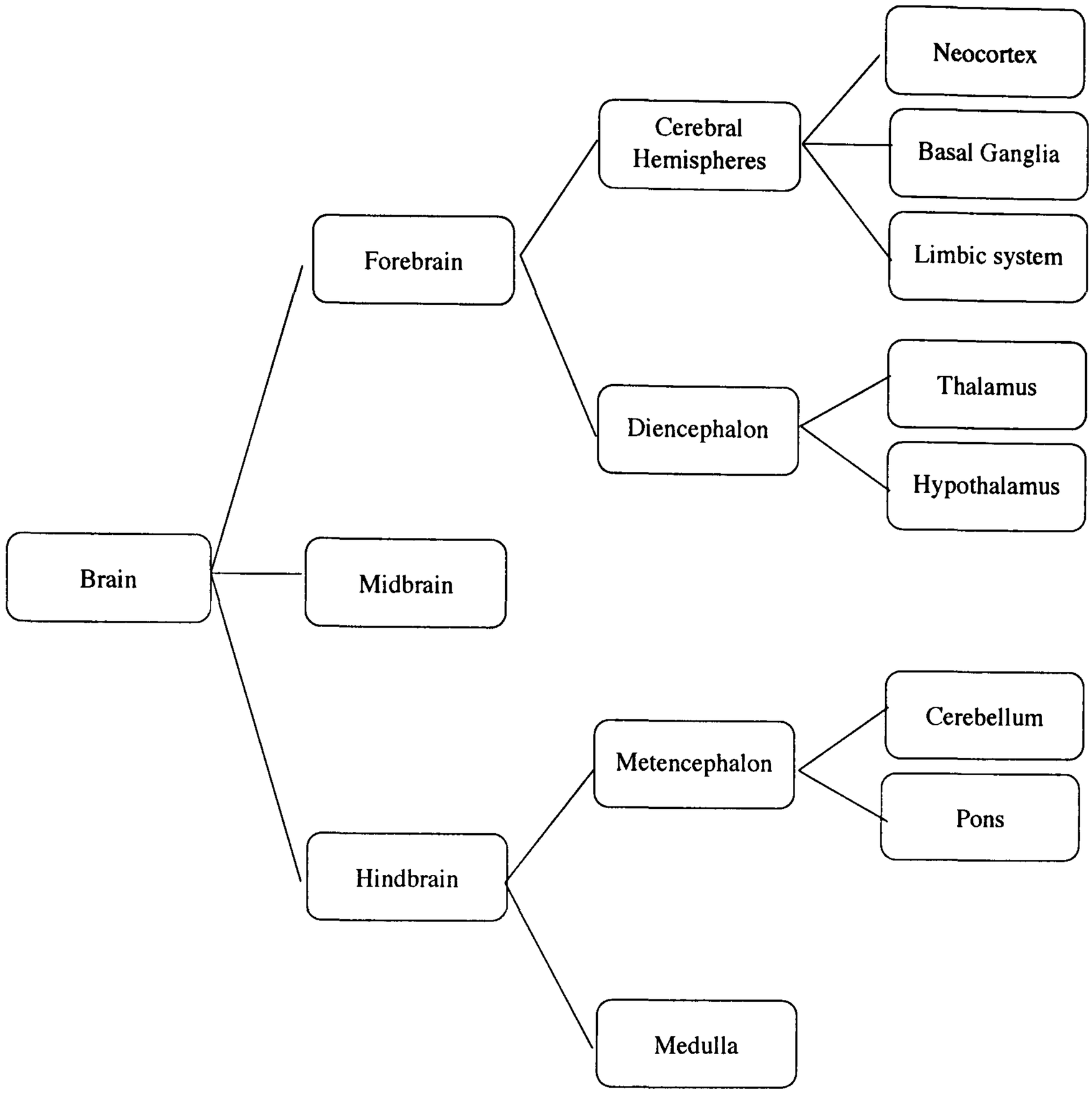


Figure 2.3: Anatomical structure of the brain

cortex are arranged into six distinct layers in a laminar type of organisation. Each cortical layer can easily be distinguished under a microscope due to the different layers consisting of groups of cells of particular sizes, and due to the different patterning of the dendrites and axons between layers. The relative thickness of the layers varies across the cerebral cortex suggesting a division of the cortex into sub-regions defined in terms of differences in the aggregation of neurons. Maps of the brain have been formed based on cell density, the size and shape of cortical neurons, and inter-cortical connection patterns. These maps show the regional divisions can also correspond to functional specialisation, such as regions processing sensory information, producing motor coordination, involved with memory storage and retrieval, and processing and producing language. The *Pyramidal* cell predominates in the cerebral cortex and the cell bodies of these cells usually are lo-

cated in layers 3-5. One dendrite of each Pyramidal cell usually extends to the outermost surface of the cortex. Other dendrites branches of the Pyramidal cell spread out horizontally from the cell body. Sensory fibres from the thalamus often terminate in layer 4 and in the area of the cortex dealing with visual stimulus layer 4 is so prominent that it appears to the naked eye as a stripe in cross sections. Fibers that leave the cerebral cortex often arise from layer 3, which is particularly prominent in the main motor regions of the cortex.

Certain regions of the brain display a distinct repeating geometrical organisation of cells, visible in the cortex as columns of neurons, approximately 3mm deep and approximately 400μm to 1000μm in diameter in humans(Mountcastle, 1979). These *cortical columns* are thought to function as information processing units. Within each column, the functional connections are mainly in the vertical direction but horizontal connections also exist. It is estimated that the human cerebral cortex contains about one million cortical columns.

Part of the organisational structure of the nervous system is not only the processing units (neurons and cortical columns) but the connections between them. Some of these connections are short pathways that loop beneath the cortex to nearby cortical regions. Others travel longer distances from one hemisphere to the other connecting regions in both performing similar processing tasks. Longer links between cortical regions involve multi-synaptic chains of neurons that loop through sub-cortical regions such as the thalamus and the basal ganglia.

The nervous system does not form fully developed but instead grows from a small number of cells to become the large and intricate structure of neurons that allows us to function as we do. In order to achieve the desired structure developmental processes must produce the large scale structures that are evident in the nervous systems of the organisms and these will be discussed in the next section.

2.4 Biological Neural Network Construction

Recent estimates suggest that the brain contains between 10^{11} and 10^{12} neurons and at least as many glial cells that needs to be organised into a highly intricate structure con-

sisting of many types of neurons with specific connections. The fact that there are between 10^{11} and 10^{12} neurons each with between 10^{3} and 10^{4} neurons suggests that the development process in exceeding complex (Purves & Lichtman, 1985; Rosenzweig et al., 1999; Levitan & Kaczmarek, 1997).

The development process relies on a number of mechanisms:

1. Neurogenesis — neuronal pre-cursor cells divide to provide the required number of neurons for the development process

2. Cell migration — the pre-cursor cells move to establish distinctive nerve cell populations at specific locations

3. Differentiation — cells change into distinctive types of neuron dependent upon their intended function

4. Synaptogenesis — the establishment of synaptic connections and the growth of axons and dendrites

5. Neuronal cell death — the selective death of neurons

6. Synaptic rearrangement — the loss of some synapses and development of others, to refine synaptic connections

These stages proceed at different rates and times in different parts of the nervous system and some of the stages may sometimes overlap within a region. The following sections will look at each of these processes in turn.

2.5 Neurogenesis

Neurons do not divide themselves but instead are derived from neuronal pre-cursor cells. The pre-cursor cells during the developmental process specialise into specific types of neurons by the process of differentiation. The pre-cursor cells can either become glial cells or neurons. Traditional doctrine until recently has stated that no new neurons are

developed after birth but recent evidence has suggested that a small number of neurons can be developed after birth and well into adulthood (Nottebohm, 1991; Gould, Tanapat, McEwen, Flugge, & Fuche, 1998).

2.6 Cell migration

At some point in the development of the neural system the neuronal pre-cursors move away from their original location of development. Even after long-range migrations cells will often undergo local rearrangement with neighboUring cells. A number of mechanisms have been suggested for the guidance of cells to their final locations:

- Inherent directional preference — cells may have a directional predilection. Cell division duplicates, and polarizes cell organelles and so the motility apparatus in the daughter cells is often at opposite poles, which can lead to mirror-symmetry (Lewis & Albrecht-Buehler, 1987)

- Chemotaxis — a variety of cell types can orient their locomotion along the concentration of a chemical gradient

- Differential adhesion — it has been demonstrated that the substrate on which the neurons grow can have different adhesive affinities and cell movement can rely on a these different affinities of substrate (Reichardt & Tomaselli, 1991)

- Contact guidance — has been demonstrated that cells can detect discontinuities in the substrate in which it grows and will align and migrate along features such as fibres. Neurons may use glial scaffolds to organise their growth and movement (Rakic, 1985)

It has been found that sometimes the migrating cells form "trains" of cells in a single-file appearance, which at the end of the journey aggregate to form the initial nuclei of the adult brain.

The process of cell movement relies on three cues, initiation cues, when the cell should start to migrate, directional cues, how the cell should determine the direction of movement from its start location to its end location, and cessation cues, how the cell should determine that it has reached it's final destination.

2.7 Differentiation

After migration the pre-cursor cells specialise into one of the types of neuron. It is thought that there are two main influences which act to decide what type of neuron the pre-cursor will become. The first method is an intrinsic self-organisation factor, that is the cell is *cell-autonomous*, where the characteristics of the cell are not influenced by its environment and it is thought that the differentiation is controlled the genes within the cell.

The second method is the influence of the environment surrounding the cell. Young neural cells seem to have the ability to become any type of neuron and the particular type of neuron it does become depends on it's location and what it's neighbouring cells happen to be. The influence of one set of cells on the fate of neighbouring cells is termed *induction* and thought to act through chemical messenger proteins (Getting, 1989; Harris-Warric, 1988; Kaczmarek & Levitan, 1987).

2.8 Synaptogenesis

Once differentiation has taken place the neurons start the process of forming connections. *Growth cones* form at the ends of the tips of the initial axon and dendrites and are used to extend and guide the them to their goal locations. One of the primary methods for axon guidance is through the use of chemical gradients released by the target neurons or other tissues and is termed *Chemotaxic/Chemotrophic guidance.* The axon growth cone responds to the concentration of the chemical gradient and so provides directional guidance. The chemical gradient can act as an attractant or a repellent and can act on specific cell types (Goodman, 1996; Tessier-Lavigne & Placzek, 1991; Keynes & Cook,

1992; Song et al., 1998). There are several families of neuronal guidance molecules. One example is that of molecule named "*Slit*" which can act as a repellant for migrating neurons and growth cones, herding them towards the necessary direction and has been show to be necessary for the correct formation of neural structures in rodents.

Dendrite growth is also affected by both intrinsic factors and cell-cell interactions, particularly the approach of axons from other cells (Brown, Zador, Mainen, & Claiborne, 1992).

In the early development of neurons, the axon contains different proteins than the dendrites, suggesting that the neuron has a basic polarity along the axon-dendrite axis. Cells may have a directional predilection and cell division duplicates, and polarizes cell organelles and so the motility apparatus in the daughter cells is often at opposite poles, which can lead to mirror-symmetry (Albrecht-Buehler, 1978).

During this phase of development synapses form at an increasingly rapid rate. Often synapses will form at specific locations on a target neuron and it is proposed that a type of chemical recognition facilitates the bonding of a presynaptic ending to a particular postsynaptic site.

2.9 Neuronal Cell Death

Cell death forms a crucial stage in the structuring of the nervous system and is seen as a type "sculpturing" process, paring down the number of neurons selectively to the correct "shape". In the development of the nervous system, it is estimated that between 20% and 80% of neurons die depending upon region. Once this period of early development has passed by, the majority of neurons that die are not replaced.

Programmed cell death occurs when cells that follow from a certain lineage are programmed to die (by genetic means). All cells in a lineage may not be needed in a given part of the animal and so programmed cell death may be an expeditious way of achieving regional variation in an adult. Cell death could be used in a number of ways. It seems

that the overproduction of neurons can be used to produce a number of different neuronal circuits at a particular region, the circuit that best suits the organism is then kept and the others die off – this suggests then that there is some sort of "testing" during development that allows a circuit to be picked above the others. In grasshoppers, programmed cell death allows differences to emerge between segmental ganglia. Cell degeneration is more pronounced in the abdominal segments than in the thoracic segments, leading to a disparity in ganglionic size (in the thoracic segment there approximately 3000 neurons, and in the abdominal segment approximately 500). This seems to be used to develop different segments in a neural network that follow essentially the same floor plan but with minor differences allowing the size representation needed to specify the network to be smaller.

Target-dependent neuronal death is where the target of the growing neurons is instrumental in determining which cells degenerate. The timing of cell death is usually quite sharp and often occurs at approximately the same time that axons reach their targets. Evidence suggests that neuron survival depends on a target-derived agent which is limited in supply. It is proposed that neurons compete for connections to targets. Neurons that connect to the target receive a chemical (*neurotrophic factors*) that is crucial for their survival; those cells not able to connect to the correct targets don't receive the chemical and so die.

2.10 Synaptic rearrangement

Once cell death has taken place, a period of synapses restructuring takes place, where some original synapses are lost and others formed. Probably the most important factor for determining which synapses are lost and formed is due to neural processing of information. The process of synaptic rearrangement continues after the organism is born an may be one of the methods for learning and fine-tuning the neural structures formed.

2.11 Communications and Information processing in neurons

The main form of communication within an organism's nervous system is that of electrical activity communicated directly from one neuron to another via synapses. This leads to relatively local spread of information in a large group of neurons and the receivers of the information are specific. Chemical communication also exists in the form of neurotransmitters, with over 30 neurotransmitters having been discovered so far. Neurotransmitters affect neurons in a much more general form, affecting many neurons at one time, though there are instances of cell-cell communication through neurotransmitters with cells that have direct physical contact.

Information in the neuron is represented as electrical signals, variously termed *nerve impulses*, *spikes* or *action potentials*, that are generated at the axon hillock and propagate along the length of the axon, out to the axon terminals (Koch & Bernander, 1995; Levitan & Kaczmarek, 1997). This is achieved through the manipulation of a voltage difference known as the *membrane potential* that the neuron exhibits, a feature common with most cells. This small electrical difference between the inner and outer surfaces of the cell membrane is achieved by controlling the relative distribution of ions within and without the neuron (Hille, 1992). The concentration of positively charged ions (cations) and negatively charged ions (anions) is asymmetric within respect the inside/outside of the cell with a higher concentration of anions inside the cell and a higher concentration of cations outside. The cell membrane contains many specialised pores or channels which allow specific ion types to flow in and out of the cell and these channels can be opened or closed to control and manipulate the potential difference at different points of the cell membrane. The opening and closing of the channels can be influenced by a number of factors such as the transmembrane voltage, the binding of neurotransmitters, hormones and the action of certain enzymes. Generally the membrane potential can range from +40mV to −90mV depending upon the relative concentrations of ions inside and outside the cell.

The axon has certain electrical characteristics which mould the shape of the action potential waveform which have the same behaviour as a resistor and a capacitor connected

together in parallel (Levitan & Kaczmarek, 1997). If a current is injected into an axon (as if it came from the soma), the rate of change of voltage is a function of both the injected current and the membrane capacitance (see Figure 2.4 and is shown graphically in Figure 2.5). The capacitance is due to a double lipid layer, which must be charged or discharged and which takes some time constant to do, $\tau = R_m C_m$, where R_m is the cell membrane resistance and C_m is the cell membrane capacitance.

$$dV/dt = I/C_m$$

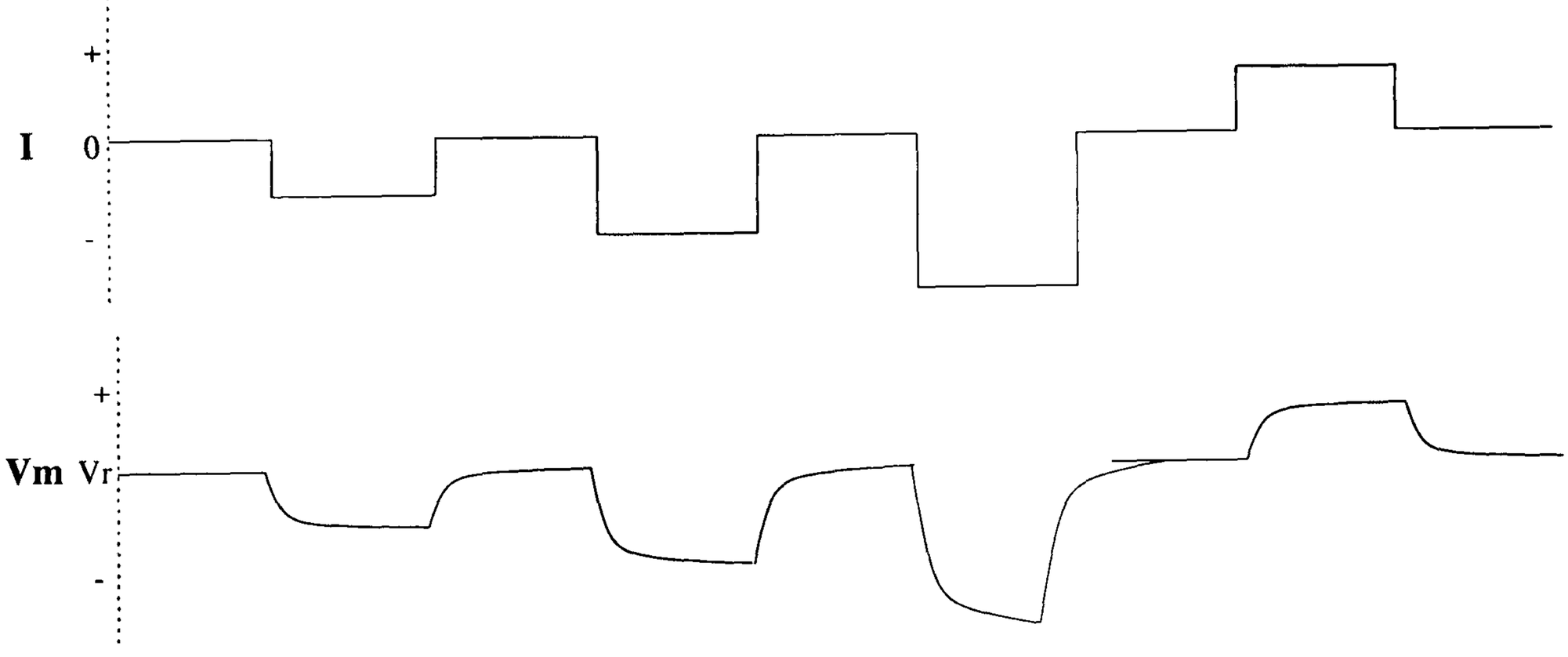
where:

V is the membrane potential voltage

I is the depolarising current

C_m is the cell membrane capacitance

Figure 2.4: Passive voltage change of membrane potential



where:

Vr is the resting potential of the neuron

I is the depolarising current

Figure 2.5: Passive voltage change of membrane potential

When the axon is at rest (not active), the potential difference (*resting potential*) at any par-

ticular point across the membrane is about −60 to −90mV. When the membrane potential is less negative than the resting potential the cell is said to be *depolarised* and when it is more negative it is *hyperpolarised.* The characteristics of the passive membrane response apply to hyperpolarising stimuli of any size and to small depolarising stimuli. But when the depolarising stimulus goes beyond a certain threshold,the action potential threshold (approximately −50mV), a sharp increase in the membrane voltage is observed. It is this active response that is responsible for the transfer of information along the axon. One essential property of this threshold feature is that it stops small random depolarisations (noise) from generating action potentials.

When an axon is depolarised beyond the action potential threshold the depolarisation itself causes large numbers of voltage-dependent sodium channels to open, seen as a large upswing in the sodium conductance G_{Na}, which increases in turn the sodium current, I_{Na}. Once the V_m approaches the maximum a further series of changes occur that make the sodium channels inactive, G_{Na} falls rapidly, I_{Na} follows and V_m drops towards its resting potential.

The behaviour of electrical activity within a neuron does not remain constant but subject to modulation resulting from synaptic and hormonal stimulation.

Information is communicated by frequency of firing of neuron not by amplitude of signal on axon. A resting neuron will fire off randomly or fire with a steady base line frequency. Once the stimulus crosses the threshold, the amplitude of the response is the same no matter what the amplitude of the stimulus was. Once the stimulus crosses the threshold an action potential is produced and the membrane potential rapidly falls back to its resting potential (see Figure 2.6). A latency period follows the cessation of the spike, during which time another action potential cannot be produced. The latency period is associated with the amplitude of the stimulus, the larger the amplitude the shorter the latency period. This produces an amplitude to frequency conversion effect. A low amplitude stimulus will produce a series of low frequency spikes, a large amplitude stimulus will produce a series of high frequency spikes on the axon, with the limit being approximately 1200 impulses per second(see Figure 2.7).

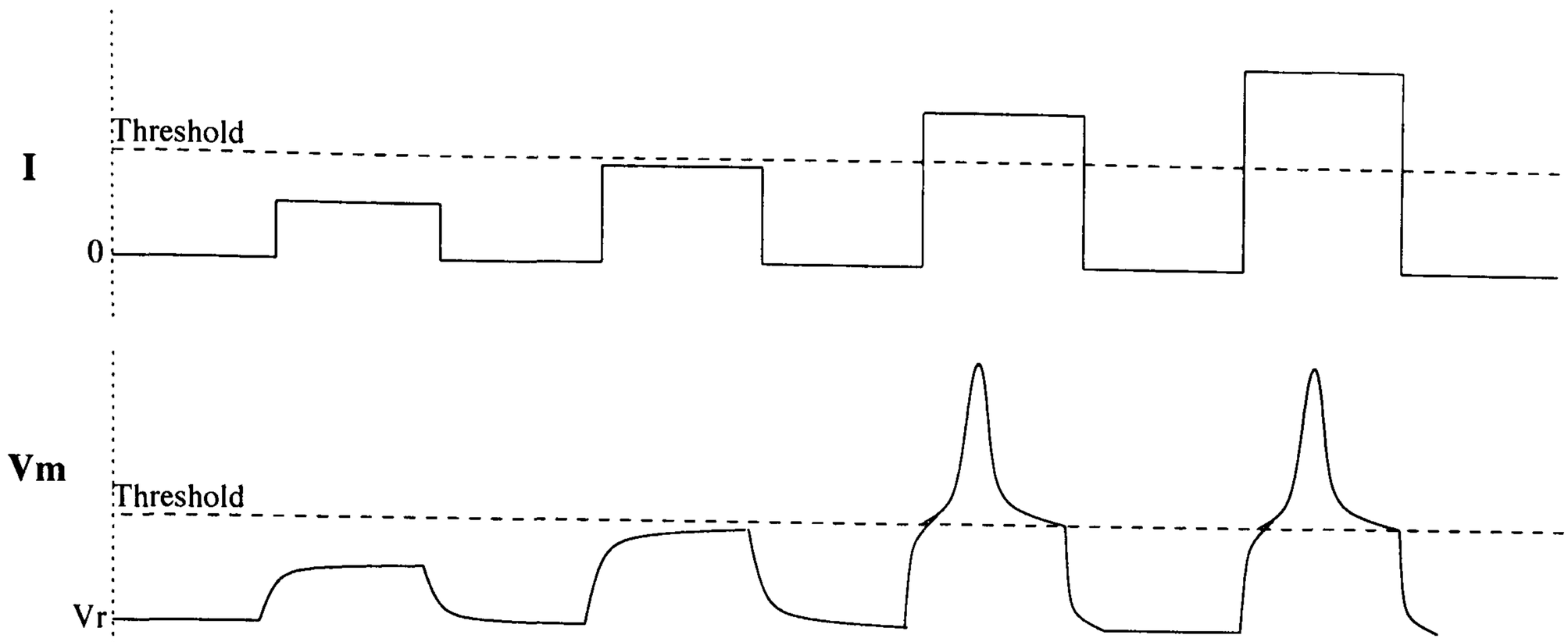
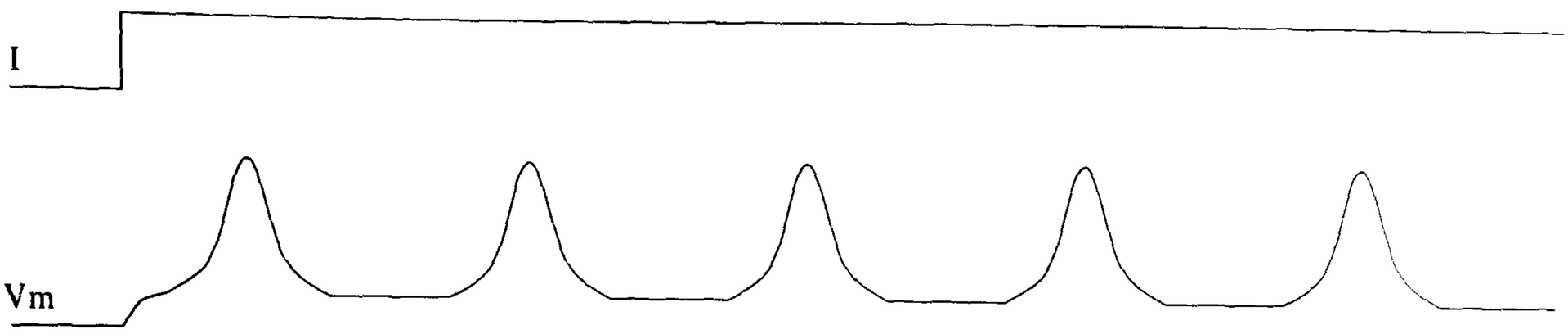


Figure 2.6: Response of a neuron to increasing stimuli(Vr=resting potential of the neuron, I=depolarising current)

The initial action potential is generated at the axon hillock and propagates along the axon towards the axon terminals. The action potential is regenerated at successive axon segments. It spreads from one region to the next because the flow of current associated with this small and rapid change in potential depolarises and thus stimulates adjacent axon segments. The refractory period of an axon segment helps to propagate the signal in the direction required.

Axon diameters can range from 1μm to 1mm and can differ along the length of a single axon. The diameter of the axon is important for determining the speed at which information is transmitted. The different speed of transmission along different axon fibres can act as "delay lines" and may be important for temporal aspects of info processing (Albeck, 1995; Carr, 1993).

Neurons can display different patterns of activity due to their different morphologies, electrical and biochemical properties and the chemical properties of their environment. The different patterns can be seen not only in the shape of individual action potential profiles but also in the pattern of action potentials. For instance, the action potentials of neurons that control fast reflex behaviour can be very different from the action potentials that control slow behaviours such as breathing. Silent neurons produce no action potentials when not being externally stimulated, their output is a non-varying resting potential. Beating (or pacing) neurons fire repetitively at a constant frequency. External stimulation can change



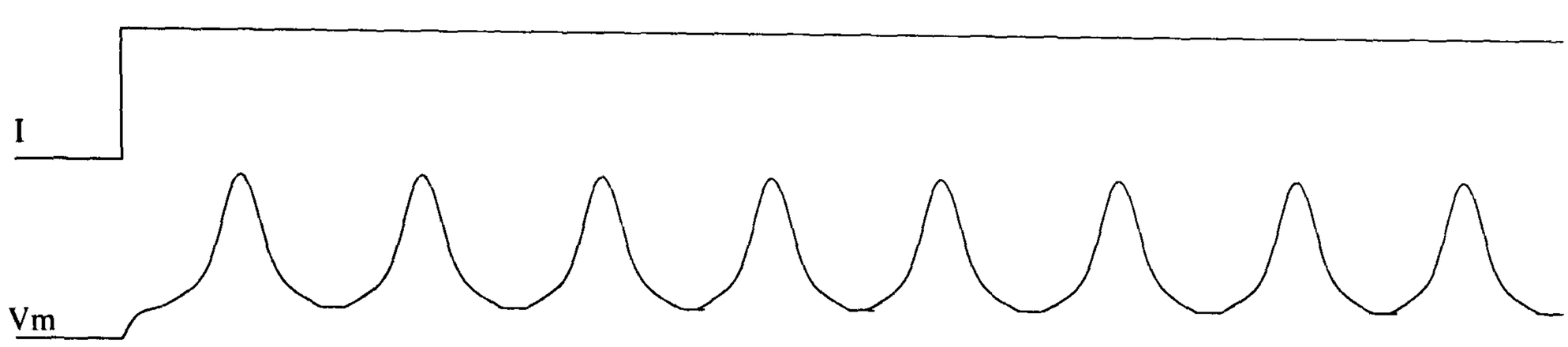


Figure 2.7: Frequency coding in axons(*Vm*=membrane potential of the neuron, *I*=depolarising current)

the firing rate, or inhibit the firing pattern altogether, but the mechanism for the beating is intrinsic to the neuron itself without need for any external stimuli. Bursting neurons generate regular bursts of action potentials that are separated by hyperpolarisations.

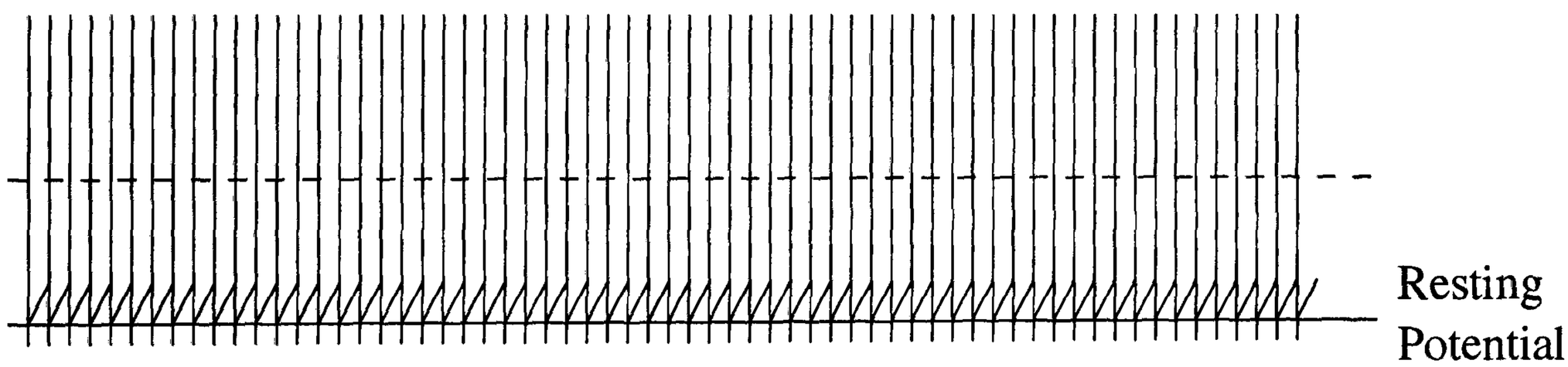


Figure 2.8: Electrical Activity of a Beating Neuron

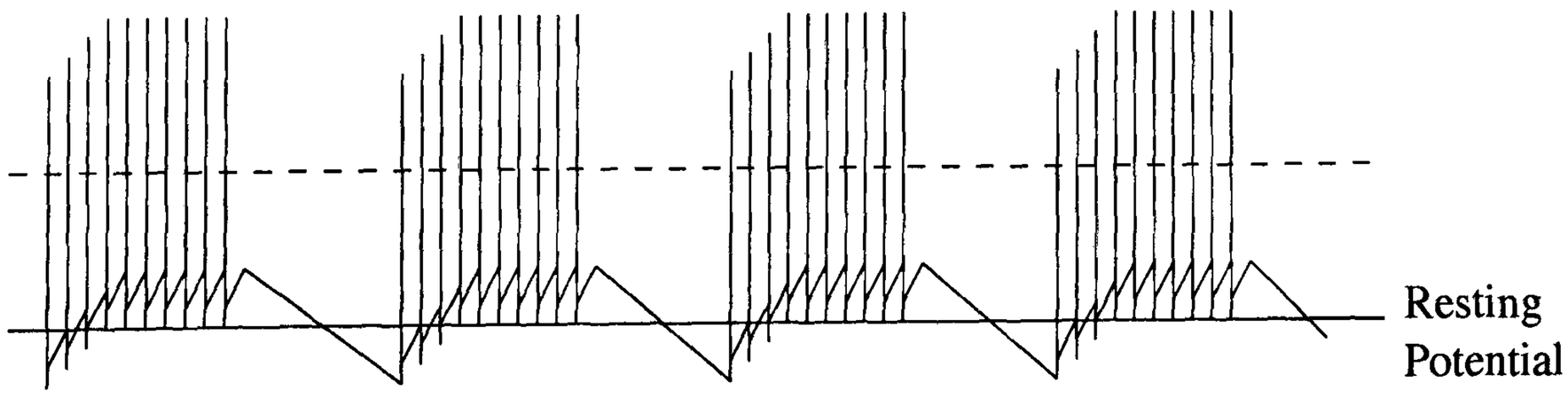


Figure 2.9: Electrical Activity of a Bursting Neuron

There are a number of ways that a neuron can respond when receiving a depolarising

stimulus :

- *Sustained response*. Action potentials are produced during the entire time the external stimulus is present with a frequency related to the strength of the stimulus.
- *Accommodation*. A single action potential is produced at the initial onset of the external stimulus and remains silent thereafter.
- *Delay*. Action potentials are only produced after a period of time from the onset of the external stimulus. Short periods of stimulus fail to trigger any action potentials.

As previously stated, the axon is the channel through which one neuron transmits its information to another neuron. The axon is connected to the dendrites of other neurons via synapses. The activity of presynaptic axon terminals elicits postsynaptic potentials. The effect that a presynaptic neuron has on a post synaptic neuron can be classified in one of two types, either excitatory or inhibitory. When an excitatory presynaptic neuron stimulates the postsynaptic neuron, the membrane potential of the soma in the postsynaptic neuron (the excitatory postsynaptic potential) is pushed towards the action potential threshold required for firing. Generally the combined effect of many excitatory synapses is required to elicit an action potential in the postsynaptic neuron. If many excitatory postsynaptic potentials occur almost simultaneously, these potentials can summate and produce a depolarisation that reaches the threshold and triggers an action potential. The effect of an inhibitory presynaptic neuron has the opposite effect; it pushes the potential of the post synaptic neuron further away from the threshold potential (inhibitory postsynaptic potential) and so makes it less likely that a neuron will produce action potentials.

A typical neuron will have many excitatory inputs and many inhibitory inputs and the function of the soma/axon hillock is to act as a summation and integration device. This means a neuron is able to both summate and subtract input signals. The effect of summation occurs over two dimensions, space and time. Spatial summation occurs when two or more signals arrive at the same time and their combined values are summated. Temporal summation occur because the effect of stimulation is not an instantaneous event but last a

short period of time. The closer two signals arrive together in time, the greater the overlap and the more complete the summation.

Dendrites also play a role in how the signals are processed. Dendrites do not produce action potentials, so the potentials produced at the dendritic spines decay the further they propagate from the spines. Thus information arriving at various parts of the neuron are weighted in terms of the distance and path resistance to the axon hillock.

The actual interpretation of the information encoding in BNNs is still open to much debate. The two primary pieces of information that must be encoded are 'What is the stimulus being perceived?' and 'How strong is the stimulus being received?'. One of the current dominant interpretations is termed the *labelled line* model. This model assumes that the arrangement of connectivities between neurons effectively 'labels' the input lines so as to encode the signal type. The frequency of the signal coming from a particular neuron encodes the strength of the stimulus. The *temporal encoding* model assumes that the information is encoded in the temporal patterning of the action potentials (Cariani & Delgutte, 1996; Reike, Warland, Steveninck, & Bialek, 1997). It is possible that each sensory modality has its own characteristic set of temporal patterns. Other time patterns would then carry information concerning other dimensions of the stimulus (intensity, location, duration, etc).

2.12 Conclusions

The nervous system in many animals is formed from a highly structured network of simple processing elements (neurons). The structure is modular in nature and has a diverse and rich structure to it, with the structure occurring on many different scales. All neurons have the same basic components, a dendritic tree to capture information from a large number of other neurons, a soma to integrate and process this information, and an axon to distribute the result of the integration to other neurons. This leads to a high degree of interconnection of neurons on a local scale. Synapses are the points of connection where information is transferred between one neuron and another neuron. It is believed

that the efficacy, or strength, of these connections are the key elements in controlling the function of the nervous system and it is the efficacy that is changed during learning. At a simple level, a neuron integration and thresholding function on the information from other neurons, but this process can be affected by many internal and external factors to the neurons and can lead to a wide variety of responses from the neuron. Information is transmitted in neurons as a series of electrical spikes or action potentials. It is unclear at the present time whether the frequency of the action potentials to the temporal patterns of the action potentials are the main form of information encoding.

Neurons can be classified by their shape, size location and function, and the classification of neurons can be further used to classify structures on larger scales. Scanning techniques such as MRI and PET scanning can be used to locate the processing of functions within the nervous system and map the modularity of it. The developmental process that creates the nervous system helps to shape the structure and modularity of final system. The process follows a sequence of developmental steps where the neurons may divide, migrate, form connections with other neurons and possibly die. The development process starts with a small number of cells and using feedback both from external environmental factors, such as chemical signals, and internal factors, such as genetic influences, grows to form a nervous system with millions of neurons. The feedback from the environment provides a flexible method for constructing the nervous system as it allows repairs to made to any flaws that may occur during development and allows 'fine-tuning' of the system. This flexibility can be a 'double-edged sword' as external factors can influence the process and lead it astray. Evidence suggests that position, and branching of neurons/axons is not under strict genetic control, other developmental factors play a role. As well as environmental factors, learning overlays hardwired connections. Processing goals correlate with processing levels in that the same processing goals of different modalities tend to be handled at the same processing levels.

In this chapter, we reviewed Biological Neural Networks before moving on to investigating how such networks can be modelled artificially.

3

Modelling Neural Networks

3.1 Introduction

This chapter considers how it has been possible to artificially recreate biological neural networks. ANNs excel in a number of problem domains, especially where precisely defined rules to solve a problem may not be known. In *classification* problems, typically pattern recognition, ANNs can be trained to recognise an input pattern and classify it into one of a set of classes. The network can be viewed as a mapping function, transforming an input pattern into an output pattern according to the function encoded in the weights of the network. An ANN can be used for *function approximation*, where the network has to associate the input pattern, close to that computed by an underlying, unknown function representing the true phenomena. Typical applications for function approximation include time-series prediction and process control. In *feature extraction and compression*, ANNs can be used to reduce dimensionality and noise from data that may be noisy or have a high-dimensionality, either by numerical compression or through extraction of relevant features. These networks are typically used to 'clean-up' data coming from real-world sensors that are prone to noise and high-dimensionality. ANNs are, of course, used to investigate, model and analyse the biological neural networks found in Nature, the area of research that originally spawned the concept of ANNs.

From research on BNNs, many types of neural simulation have arisen. The detail of the simulation, with regards to its realism with respect to BNNs, depends primarily on

the purpose of the simulation and the problem under investigation. For instance, if the purpose of the network simulation is primarily to produce a network capable of solving a classification problem, then a model on the scale of 10^{-4}m (eg Multilayer perceptron model) would probably suffice, as this type of model offers the computational equivalence of a BNN, but has very little biological realism. This model suffices as the primary purpose is to simulate the processing capabilities of BNNs. For the purpose of investigating how assemblies of neurons work together in the brain when modelling the response of networks of different neurons, then a level of simulation on scale 10^{-8}m may be needed. The Figure 3.1 shows how the different levels of detail can be visualised.

Typical Unit of Measurement (metres)	**Typical Structure**	**Type of Model**
10^{-2}	Whole Brain	
10^{-3}	Anatomical Regions	
10^{-4}	Neurons	
10^{-5}	Processes/Somas	
10^{-6}	Synapses	
10^{-7}	Synaptic Cleft	
10^{-8}	Neuron Membrane	
10^{-9}	Ion Channels	

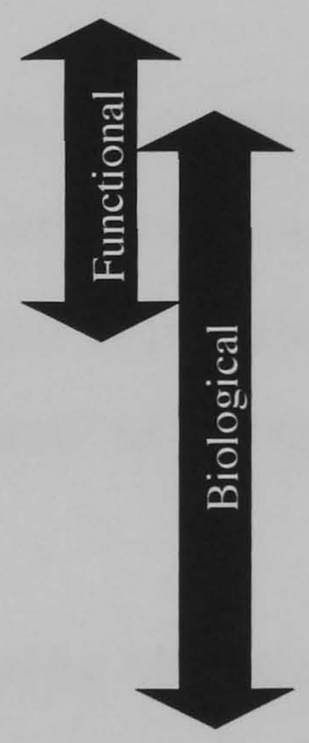


Figure 3.1: Spectrum of detail/realism

As can be seen there is a dimension of biological detail that depends upon what sort of simulation is required. The addition of more detail adds more computation time to the simulation of the model. As yet the computational demands required for a detailed realistic model of hundreds of thousands of neurons cannot be met.

The models can be categorised into two main types of model, those that strive for functional realism only and those that strive for biological realism. Generally functional only models are primarily clustered around the low detail end of the spectrum and biological

models cover the detailed end of the spectrum. It is also noticed that due to the computational demands placed on simulating more realism, functional models tend to be many times larger than biological based models.

Researchers often want to distil the complex shape and behaviour of a real neuron into a simpler model, either to guide a neurobiological experiment or to construct a functional network. Choosing which neuronal properties to keep and which to ignore is heavily influenced by the nature of the problem to be solved and how the information processing capabilities of neurons operate (Softky & Koch, 1995). Spatial detail/resolution is often sacrificed in functional models in order to obtain larger networks or gains in computational speed.

3.2 Functional Models of Neurons

Functional models aim to capture the simplest aspects of neuron information processing abilities. These models often disregard most physical and temporal processing features of neurons but due to their simplistic nature they are computationally the easiest to handle and take up the least processing power per node, and have a 'speed' advantage over other ANN models.

One of the first functional models to be developed was developed by McCulloch and Pitts (1943) who proposed a neural network model using artificial neurons, or nodes, called *Threshold Logic Units* (TLUs). The TLU is based on an abstract representation of a biological neuron and uses a mapping function to translate its input values to a single output value, described in Figure 3.2. The function has two parts, an *integration function* and a *transfer function.* The integration part arithmetically sums the inputs from other connected nodes (x_n), multiplied by their appropriate connection weights (w_n), equivalent to the synaptic efficacy, and the result is termed the *activation* of the node. The transfer function used in a TLU node is a simple thresholding function which outputs a '0' if the activation of the node is less than the value of a threshold θ, otherwise a '1' is output. This is a gross simplification that can be seen to mimic the very basic functioning of signal integration of a simple biological neuron but for the purposes of a functional model,

it is deemed to capture the most elementary and important properties of information processing of a real neuron.

$$a = w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_nx_n$$

or more compactly as

$$a = \sum_{i=0}^{n} w_ix_i$$

the output y is then given as

$$y = \begin{cases} 0 & : \quad a < \theta \\ 1 & : \quad a \geq \theta \end{cases}$$

where:

a is the activation of the node

w is the weight of a connection

x is the input from a node

y is the output of the node

θ is the threshold value of the node

Figure 3.2: Transfer Function for the Threshold Logic Unit

Individual nodes are connected together into a network of nodes, and an example of a typical network is given is Figure 3.4, where there are three layers, an input layer, a hidden layer and an output layer. Each connection has a value associated with it, commonly called a *weight*, indicating the strength of the connection between the connected nodes. The network is trained using a training/construction algorithm, coupled with training data, that sets the magnitude and sign of the weights appropriately for the problem. Once training is complete, examples can then be presented to the input layer of the network, processed, with the answer indicated on the output nodes.

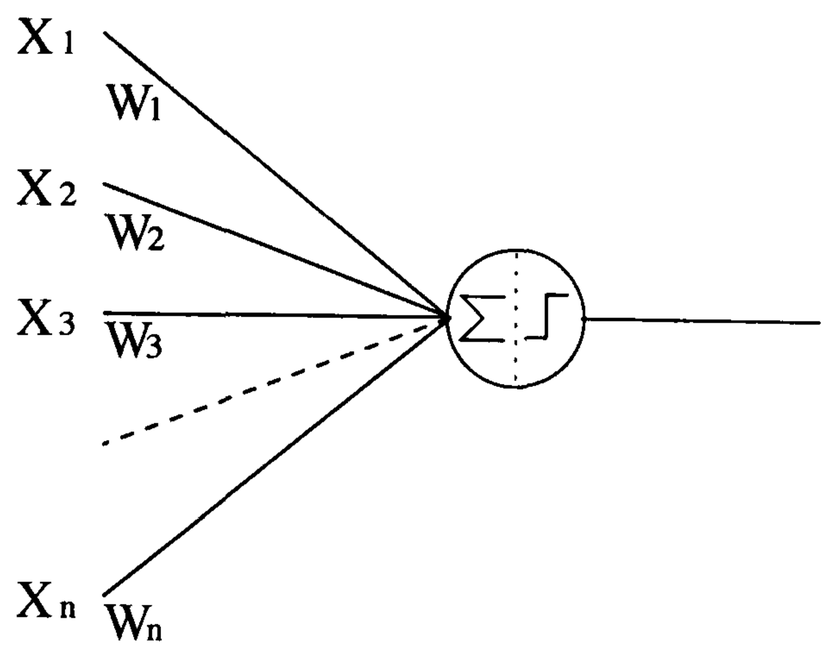


Figure 3.3: Graphical description of a TLU node

3.3 How a TLU network works

As previously stated a TLU can be used as a classifier system, where the input patterns represent points in a multi-dimensional problem space. The weights of the connections between nodes in the network define decision hyper-planes in the problem space, dividing the space into a number of volumes, with each volume representing a category in the classification scheme.

An example of this is to examine how a neural network can be constructed to categorise the following data in Table 3.1. The data represents the logical AND classification function, the data is classified into two classes, one class where the output is zero and another class where the output is 1. The architecture of the network needed to classify the data will consist of two input nodes and one output node. Due to the fact that only two inputs are needed to classify the categories, the problem space can be represented in two dimensions (see Figure 3.5). An n-dimensional problem space is required to represent and n-input problem.

To solve the AND function only one TLU node is required, with two inputs and one output. If the equation used to calculate the value of this node is expanded to give, $w_1 * x_1 + w_2 * x_2$, and rearranged to this can be recognised as the equation of a plane in two dimensions. When the threshold function is applied to the activation of the node, the

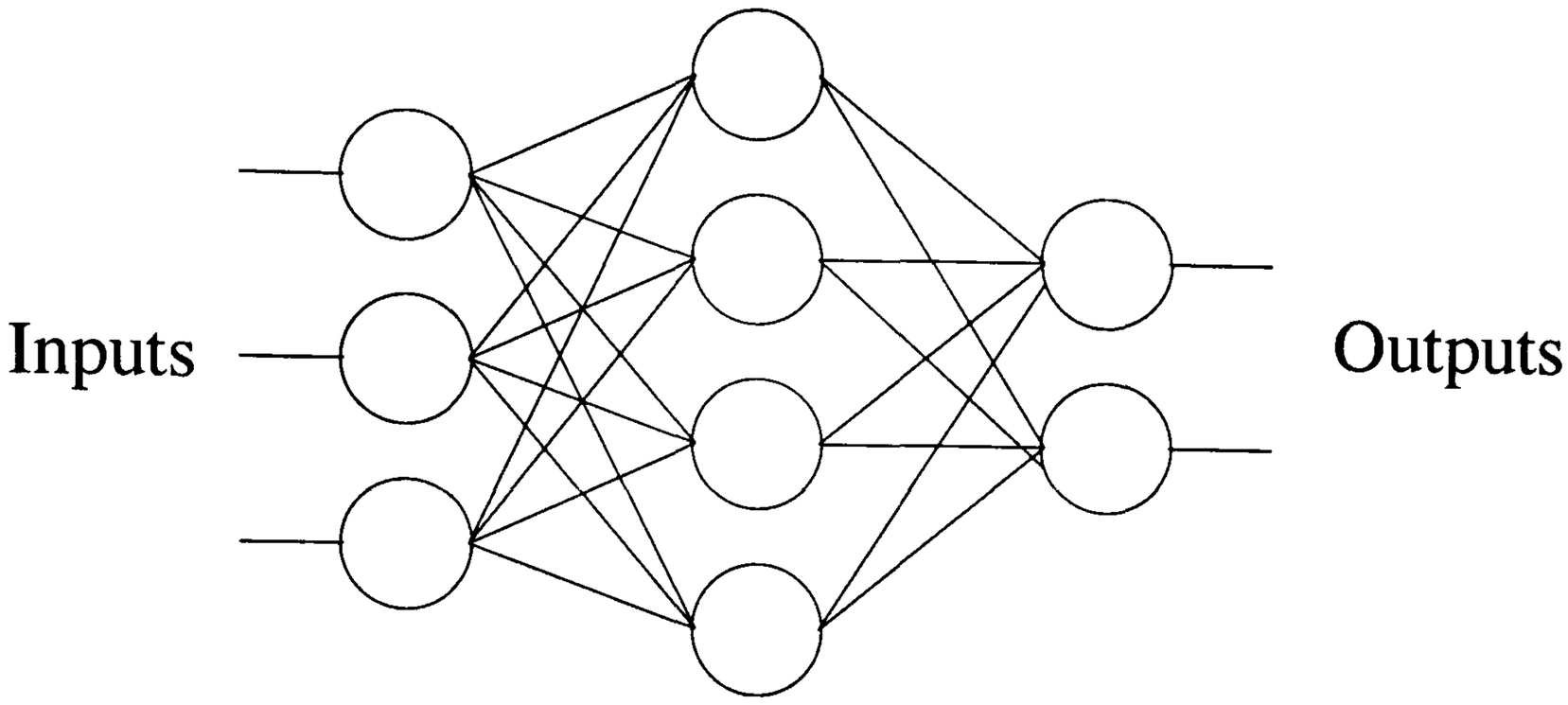


Figure 3.4: Graphical description of a TLU network

Pattern	Input X1	Input X2	Output
A	0	0	0
B	0	1	0
C	1	0	0
D	1	1	1

Table 3.1: AND function input/output patterns

problem space is divided into two areas on either side of the plane defined by the weights and bias. This then forms the basis of a classifier, a method which depending upon the value of the inputs, classifies the input into one of two classes.

In this graphical form of the problem space, the dashed line in Figure 3.5 represents the decision hyper-plane, that the weights must encode in the network. Any data point falling to origin side of the decision plane requires a '0' to be output from the network, any data point falling on the other side of the decision plane requires a '1' to be output from the network.

The weights alter the slope of the decision plane and the bias alters the relative position of the decision plane to the origin. As can be seen from the diagram a large number of weight and bias values could be chosen to position the decision plane to give the correct

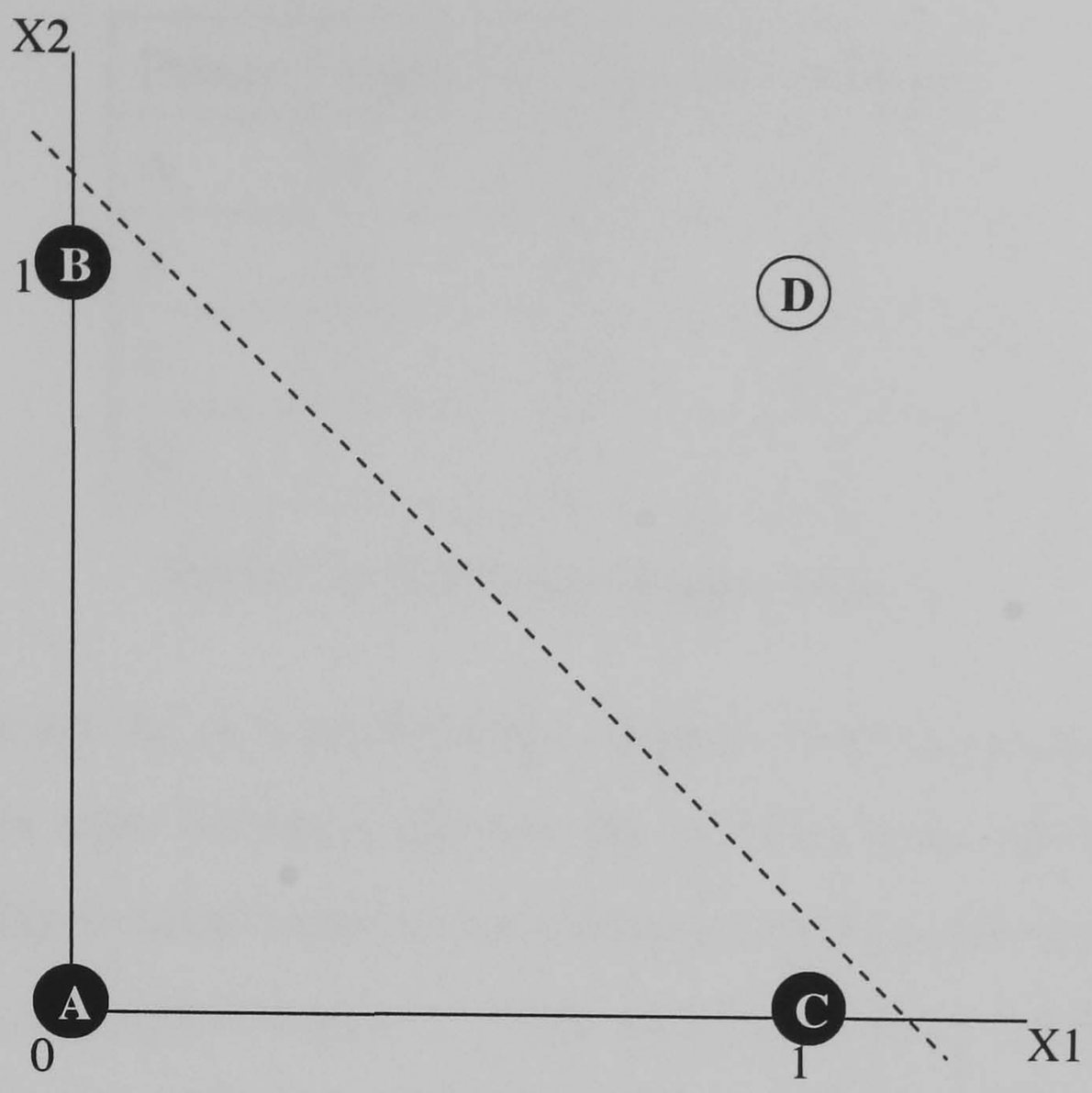


Figure 3.5: AND problem space with decision hyper-plane

output. The output from a TLU can be altered from giving a '0' or '1' to giving a '-1' and a '1'.

3.4 The problem with the XOR problem

As has been previously stated a TLU network can be used to divide the problem space into two classes using a single decision plane, a '1' class and a '0' class, but a TLU network comes 'unstuck' when trying to solve the XOR problem (see 3.2) which, like the AND problem in Section 3.3, is a 'simple' classification of the inputs into two classes. The problem space can be plotted as with the AND problem and is shown in Figure 3.6. Unfortunately no single decision plane can be used to divide the problem space into the two classes, a minimum of two decision planes are needed, with one possible configuration shown in Figure 3.6.

Consider the network topology shown in Figure 3.7. The network shown has three layers, an input layer, an output layer and a layer between them, termed the *hidden layer*. If the node in the output layer has its threshold set so that it turns on only when both of the

Pattern	Input X1	Input X2	Output
A	0	0	0
B	0	1	1
C	1	0	1
D	1	1	0

Table 3.2: XOR input/output patterns

hidden layer nodes are on, it is performing a logical AND function. Since each of the nodes in the hidden layer defines a plane in the problem space, the node in the output layer produces a classification based on a combination of these planes. If one node in the hidden layer is set to respond with a '1' if the input is above its decision plane, and the other responds with a '1' if the same input is below its decision plane, then the network is capable of producing a correct output consistent with an logical XOR function.

By simple extrapolation each hidden node can represent one decision plane and with the addition of more nodes in the hidden layer, the definition of more and more decision planes is achieved. This allows the definition of *convex regions* which classify the problem space into one of two regions defined by the decision planes. A convex region is a region in which any point can be connected to any other point by a straight line that does not cross the boundary of the region.

With the addition of another hidden layer, the output layer will receive as its inputs, not lines, but convex hulls, and the combinations of these need not be necessarily convex. The combinations of these convex regions may intersect, overlap or be separate from one another, producing arbitrary volumes in the problem space. A summary of this is shown in Figure 3.8.

A node typically has the feature that it combines, or integrates, its inputs into a single output. Each node also has at least one internal state, the *activation*, corresponding to the membrane potential of a real neuron.

The transfer function affects the input/output mapping of the node and the choice of mapping function is dependent upon the application of the network. In most cases the output

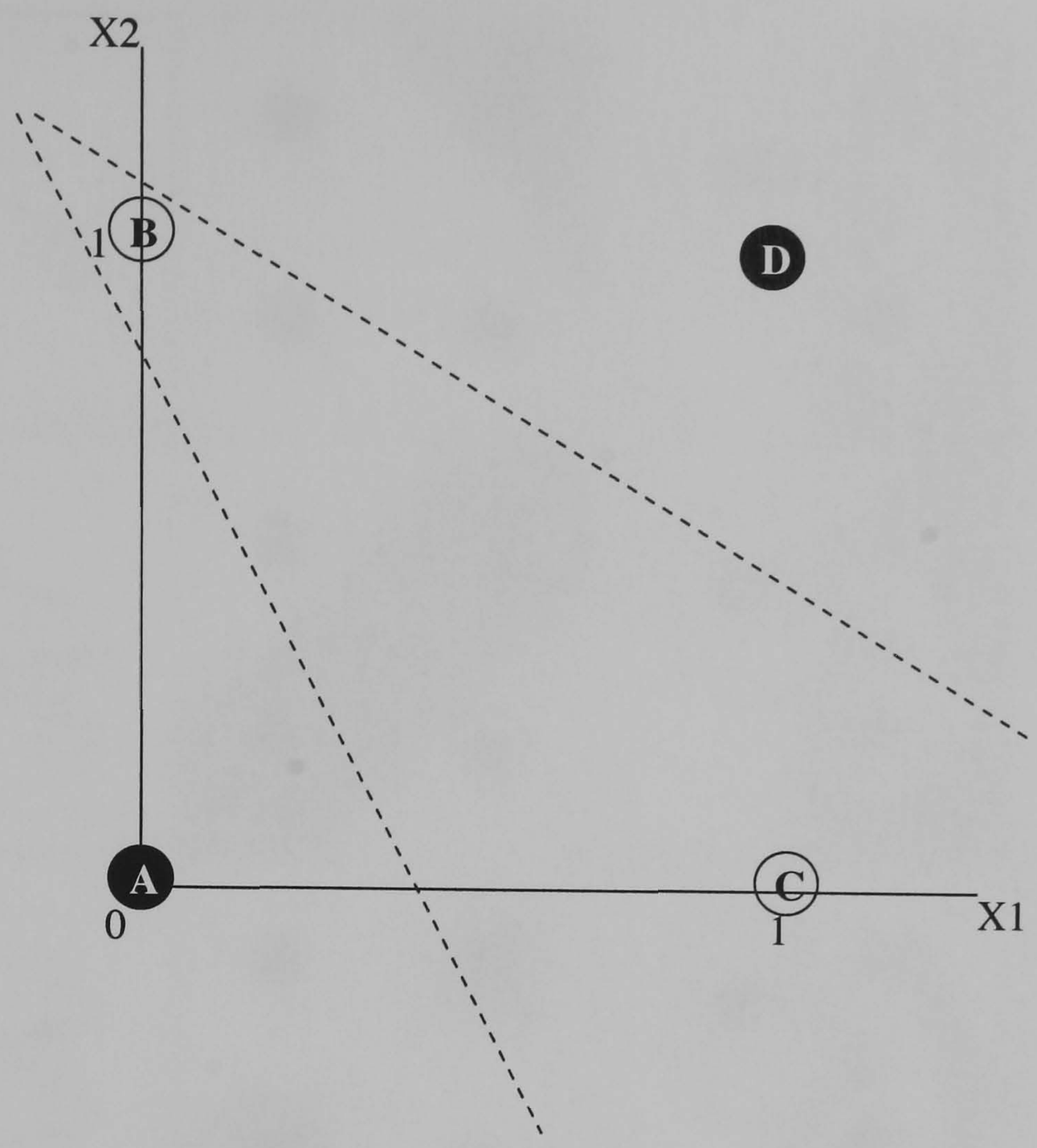


Figure 3.6: XOR problem space with decision planes

is limited to a range of values between a maximum and a minimum value. The positive maximum value is typically '1' and the minimum value is typically either '-1' or '0'. Four typical transfer functions are (see Figure 3.9):

- Threshold Logic functions – The output of the node is a binarised value to either the maximum value or the minimum value. If the activation of the node is greater than a threshold value, the output of the node is the maximum value, else the minimum value is output.

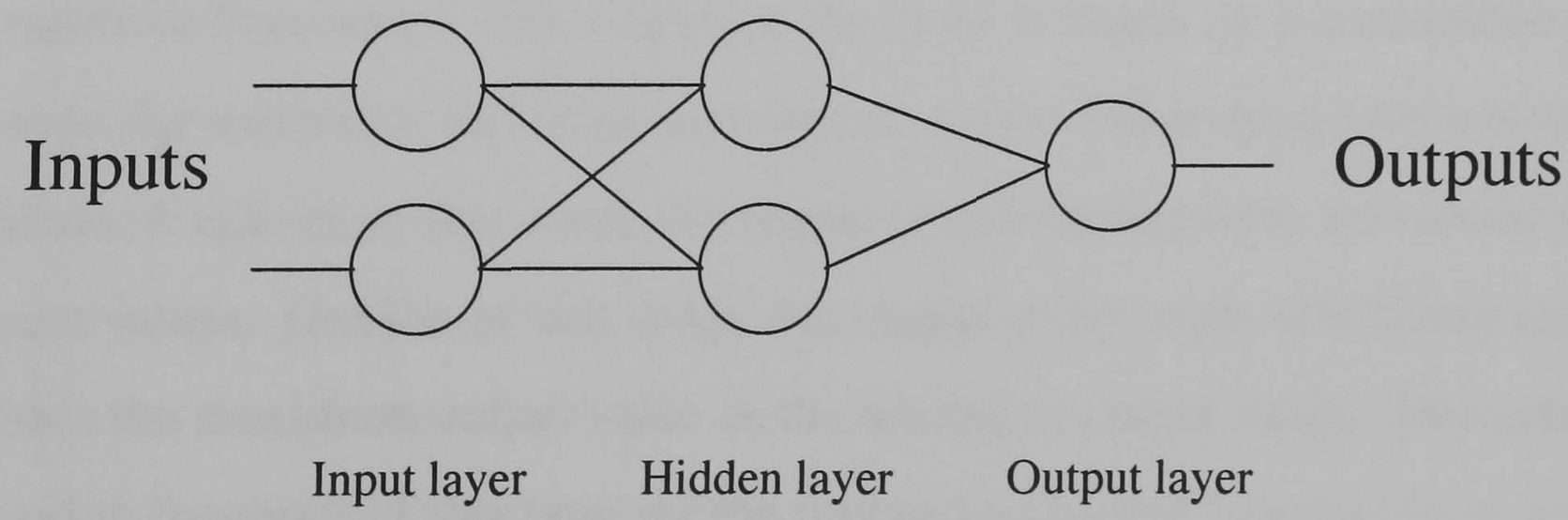


Figure 3.7: XOR network

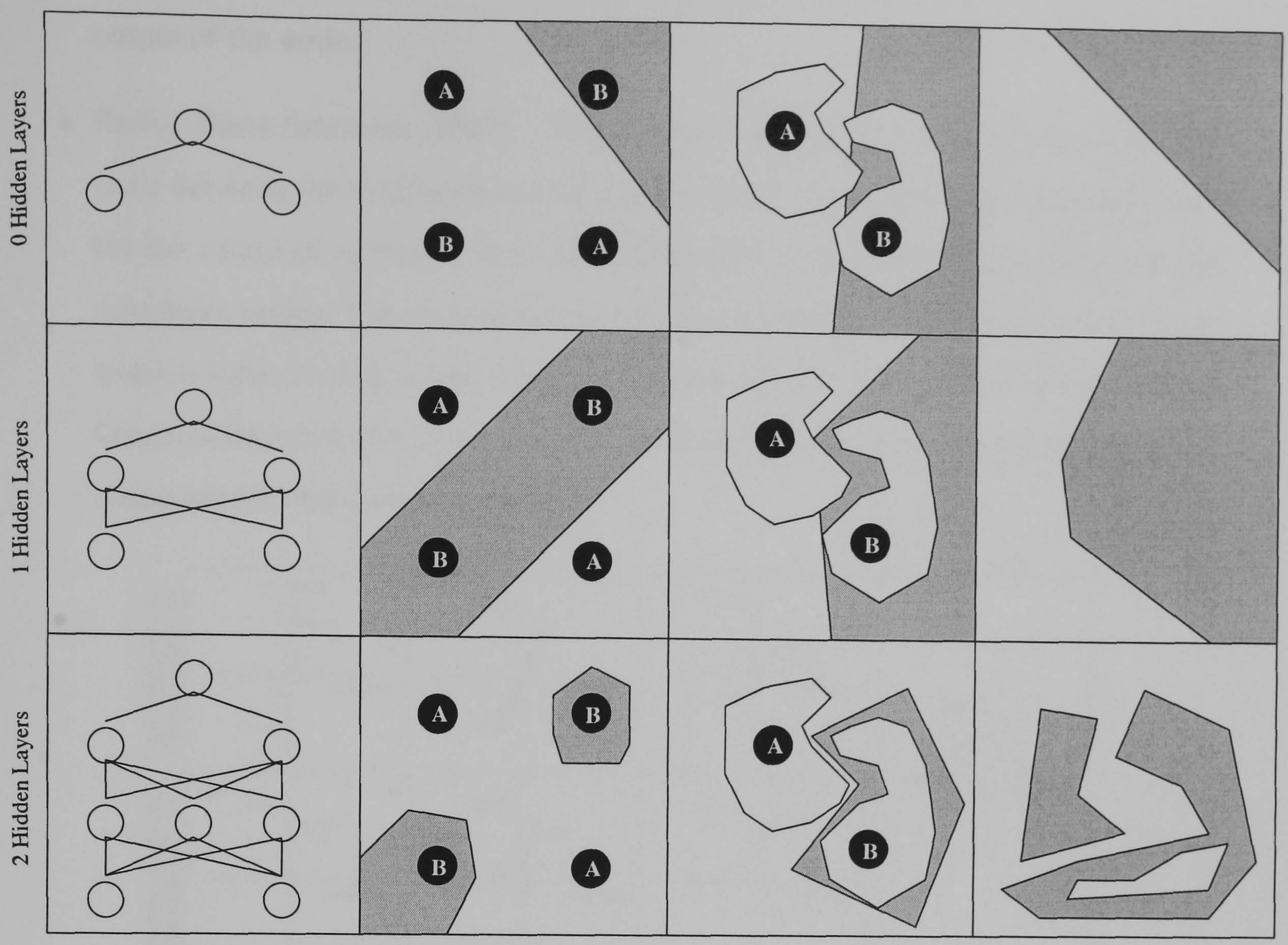


Figure 3.8: Network topology and region classifications

- Semilinear functions – The output of the node is based on a continuous scale between the maximum and minimum values. A linear relationship exists for the values of activation that cause the output to fall between the maximum and minimum values. Outside of this range the output of the node saturates to either the maximum output value or the minimum output value.

- Logistical functions – The output of the node is based on a continuous scale between the maximum and minimum values. A non-linear relationship exists for the values of activation that cause the output to fall between the maximum and minimum values. Outside of this range the output of the node effectively saturates to either the maximum output value or the minimum output value. Two common activation functions of this type are the sigmoid activation function ($y = \frac{1}{(1+e^{-(a-\theta)/\rho}}$) and the tanh function ($y = \tanh(a)$), where a is the activation of the node and y the

output of the node.

- Radial Basis functions (RBF) – The output of the node is based on a continuous scale between the maximum and minimum values. A non-linear relationship exists for the values of activation that cause the output to fall between the maximum and minimum values. The node responds with the maximum output value when the activation value is close to the 'centre' of the function. A typical RBF is based on the Gaussian function (for a one dimensional function, $y = exp(-\frac{a^2}{2\sigma^2})$) and produces a characteristic 'bell-shaped' curve.

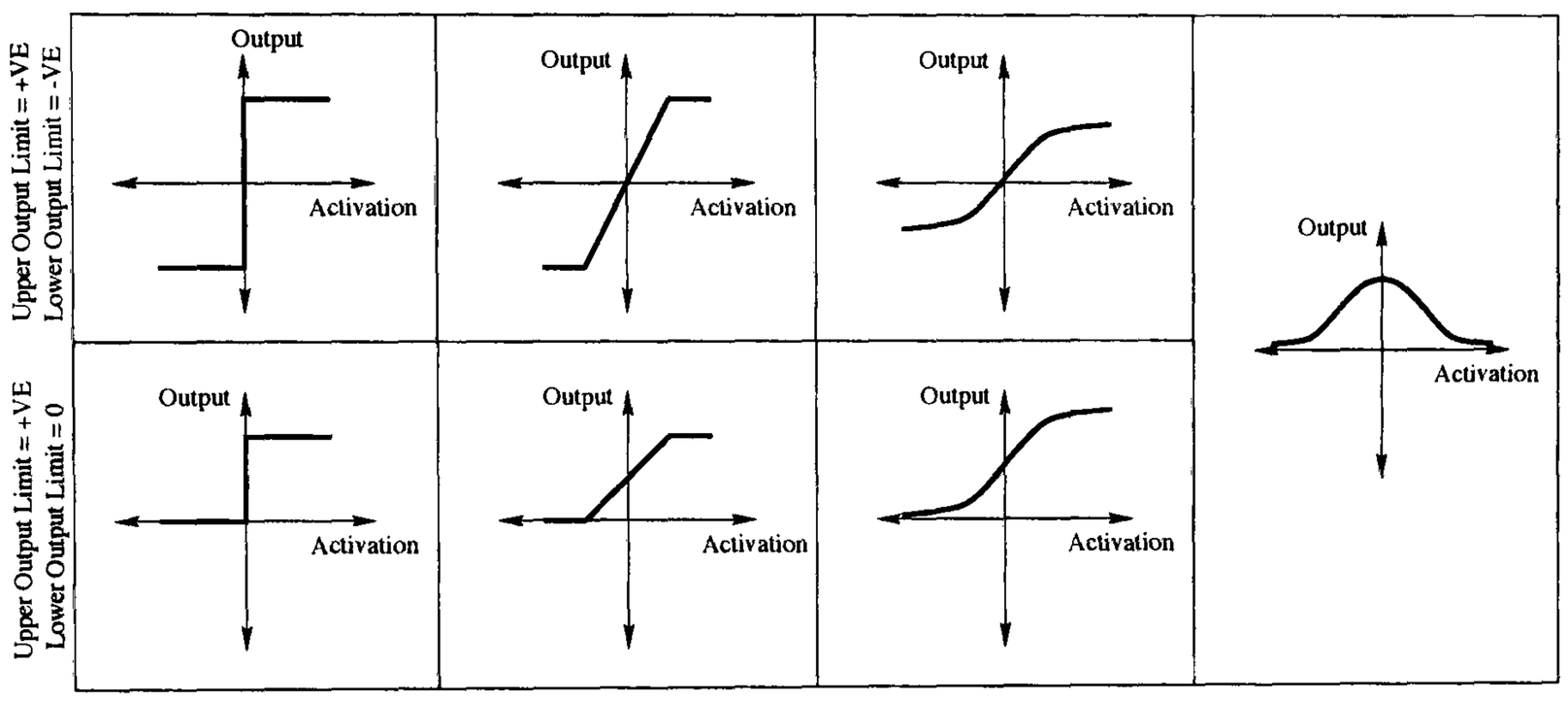


Figure 3.9: Types of Node Transfer function

Logistical type functions are also termed *squashing functions* as they have the ability to 'squash' any input into a range of '0' to '1' with the sigmoid function or '-1' to '1' with the hyperbolic tangent function. Nodes using Semilinear, Logistical and Radial Basis functions operate in the same manner as a TLU in that the inputs are weighted and summed but rather than using the a hard threshold function, the 'softer' functions are used with an advantage of this is the output now covers a range of values between the two output limits as opposed simply an on/off value. The advantage of a range of output values is that the node can now respond with a value correlated to the amount of input it is receiving, equivalent to the frequency of firing of a biological neuron and so can convey a much richer stream of information. These functions are also required for many training algorithms to work correctly, as will be shown in Section 5.2.

The use of different network transfer functions can affect the type of network topology

required to solve a particular type of problem. For instance, consider the MLP network, which employs a semilinear transfer function, that is used to categorise the data represented in the graph to the right of the figure, where points within the region should be classified as '1' and any other points should be classified as '0'. Using an MLP network, the architecture and weights can be used to define a region surrounding the 'cluster' of data. As can be seen the region can be viewed as a convex region defined by a number of decision planes.

By changing the transfer function employed in the hidden layers of the network to a RBF, the network topology can be radically changed. A RBF node works by effectively defining a 'region' in the decision space similar to the convex regions defined by the combination of the two hidden layers of the MLP network.

This idea has been further investigated by a number of researchers. Juedes and Balakrishnan (1998) has been researching the concept of using different transfer functions within the same network. The task to be solved by the network is the classic 2-circles problem, which an MLP network finds difficult to solve. Patterns are to be classified into one of two classifications, where a pattern gives a point in 2d space. The 2d space consists of 2 concentric circles. If the given pattern falls between the two boundaries of the circle the network should generate a '1', and if the pattern falls within the inner circle a '0' should be generated. If the network were to use only semilinear nodes then, in theory, an infinite number of nodes would be required to guarantee 100% accuracy, a circle is essentially an infinite number of decision planes joined together. A large number of semilinear nodes can approximate this boundary though. By allowing different types of node transfer function and specifically a *spherical* transfer function ($t_i = \Sigma_j (w_{ij}.o_j)^2 - w_{oi}^2$), (Juedes & Balakrishnan, 1998) constructed a network, using a genetic algorithm, consisting of only two input units, a spherical node and a semilinear output node, a much smaller network than would be possible with either semilinear nodes or RBF nodes.

3.5 Network Topologies and Structures

The topology of a network is dependent primarily upon the number of nodes in the network, the type of node used and the manner of connection between nodes. The TLU network (presented in Section 3.2) represents an example of a very common class of network topology termed the *feed forward network* (see Figure 3.10). This network topology consists of a number of 'layers' of nodes. In a three layer feed forward network, consisting of an input layer, a hidden layer and an output layer, each node of the input layer is connected to each node in the hidden layer, and each node in the hidden layer is connected to each node in the output layer. The term 'fully connected' is used to describe the situation where each node in one layer is connected to each node in the succeeding layer. The term 'feed-forward' is used to describe the condition where the information in the network flows only from the input layer, through the output layer, to the output layer, in other words the information flows in one direction only.

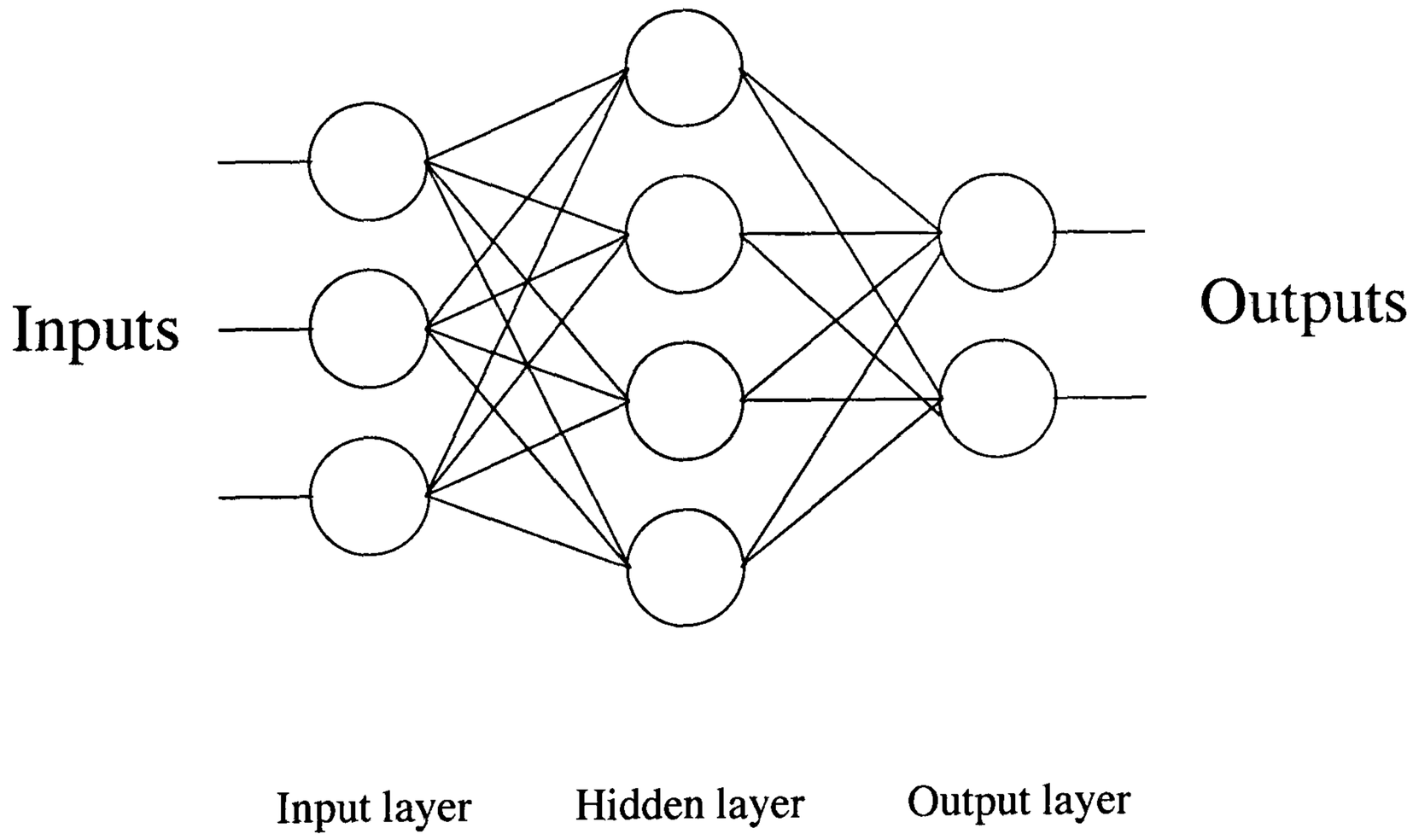


Figure 3.10: Graphical description of a feed forward network

Other classes of network topology exit, such as the *fully-connected network*. This term is used to describe a network where every node is connected to every other node and thus this topology does not show the layered structure of a feed-forward network (see Figure 3.11). A fully-connected network is an example of a network where information can flow in many directions and may indeed flow from output nodes to the input nodes. A network

in which the output depends on a previous value of output is termed a *recurrent network.*

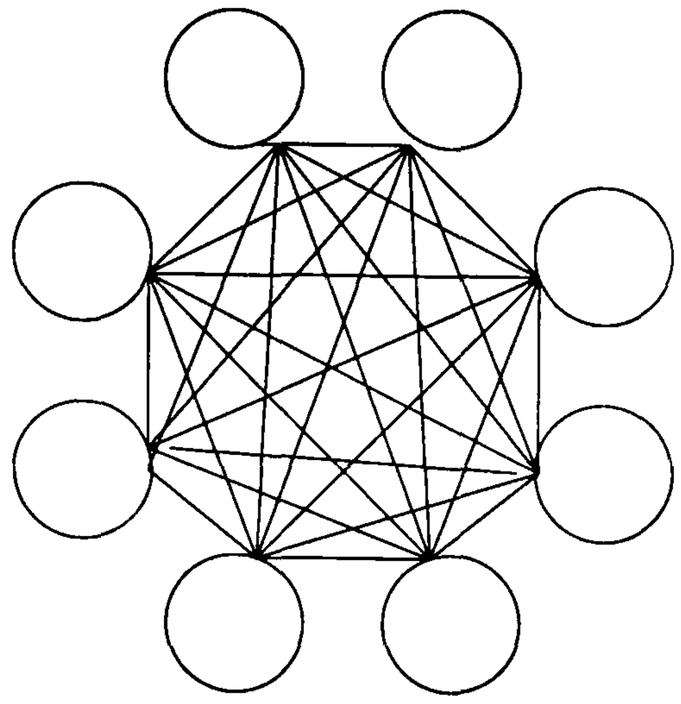


Figure 3.11: Graphical description of a fully-connected network

Maren (1990) suggests using a set of three categories, based on the level of structure, to examine ANN topologies:

- The Micro-level structure – The types of nodes used in the network
- The Meso-level structure – The organisation of nodes and connections in the network
- The Macro-level structure – The conglomeration of different networks to solve tasks

The following criteria at the meso-level structure can be used to define different network topologies:

- the number of nodes
- the type of node
- The type of connections used to connect nodes. Are there lateral connections between nodes in a layer? Are there feed-forward connections only or are there bidirectional connections (recurrent)? Are the connections between nodes symmetrical?
- The number of neurons per layer

- The degree of connectivity between nodes.

By using these criteria the following network topologies can be distinguished:

- Multi-layer feed-forward networks — eg Multilayer perceptron networks
- Single layer laterally connected networks — eg Hopfield networks
- Single layer, topographically-ordered networks — eg Kohonen networks
- Bilayer feed-forward/feed-backward networks — eg Adaptive Resonance Theory (ART) networks
- Multi-layer cooperative/competitive networks
- Hybrid networks — Networks where more than one architecture is combined

Other classification schemes have been suggested by Balakrishnan & Honavar, 1997.

3.6 Biophysical Models of Neurons

As can be seen from Chapter 2 biological neurons and networks have an extremely rich, complex behaviour and structure. Biological based models make an attempt to model this more closely than purely functional models, though many crucial properties of the biological systems remain unknown. Where functional models only tend to model across three orders of magnitude of scale ($10^{-6} \rightarrow 10^{-8}m$), biological models tend to model across six orders of magnitude ($10^{-2} \rightarrow 10^{-7}m$), with detailed modelling going right down to individual ion channels in a cell (Bove, Guigliano, Grattatola, & Massobrio, 1998). Many models exist that capture different aspects at different levels of a neuron. Some models concentrate on modelling individual ion channels, others model the axon or dendritic tree whilst other model the synapses. To make a detailed and complete model at the neuron level, all these elements would have to be combined.

3.6.1 Modelling at the neuron level

One of the first mathematical descriptions that adequately captured the shape, duration and frequency coding of an axon potential was developed by Hodgkin and Huxley (1952). Hodgkin and Huxley (1952) used a four-dimensional set of coupled, nonlinear, partial differential equations to describe the initiation and propagation of action potentials in axons and even today they are treated as exceptionally good models. Simplified versions, such as those developed by FitzHugh (1969), have since been derived that show the same phenomena though their reduced parameters cannot be mapped into the biological realm as well as the Hodgkin-Huxley equations, but due to the simplifications are more mathematically tractable than the Hodgkin-Huxley model. Generally these equations model the complex interaction of the large number of ionic channels involved in propagation, including factors such as linear and non-linear ionic channels, and refractory periods in a single set of differential equations. The model of the neuron does thus capture the dynamic temporal aspects of a biological neuron.

A number of models, collectively called *integrate and fire neuron models* provide a different approach to modelling biological neurons. The underlying strategy of these models employ a neuron which, over time, adds up the incoming pulses it receives (integration) until a threshold is reached, at which point the neuron emits an output pulse (firing). This model relaxes the requirement that a single set of differential equations describe all parts of the neuron. The cell voltage starts at zero, increasing or decreasing according to the input it receives through its synaptic input. When the voltage reaches a threshold value, the cell fires an output pulse, and resets the voltage to zero. A refractory period is usually employed, during which time the cell cannot fire, after which the cell is ready to fire again.

To enhance this type of model to resemble a biological neuron more closely, a 'leaky' component is added. Given a current cell voltage of a neuron, if the neuron receives no input over a particular time period then the cell voltage of the neuron will slowly decrease. ie the cell voltage 'leaks' away if no input is received. This, coupled with the threshold, has the effect of reducing erroneous signal output caused by noise in the input signals.

The neural models so far have not taken into account the spatial nature of biological networks. In BNNs signals take time to propagate along a dendrite, from one neuron to another, and from region to region, and as previously stated this could have consequences for the temporal processing of information. The connectivity produced during development could also be dependent upon the physical distances between neurons. To incorporate this physical aspect, *cable* (Rall & Agmon-Snir, 1998) and *compartmental* (Segev & Burke, 1998) models have been developed to model the dendritic tree of a neuron. In both these types of model the neuron is broken up into a number of 'sections'. When a signal propagates along an axon or dendrite the signal travels from one section to the next and processing is performed in each section. The resolution of an individual neuron is altered by the number of sections that are used to describe it. The greater the number of sections used to describe the neuron, the closer the neuron comes to modelling the actual neuron. The number of sections being modelled largely depends on the processing power available and the reason for requiring the model.

The term 'cable model' comes from the derivation and application of the equations used to model the flow, and the spread of the resultant voltage, of an electrical current in and around a cylindrical core conductor, such as an electrical cable. The model of an axon or dendrite can be considered to be such a conductor. Many applications of cable theory to the modelling of dendrites assume that the dendrites respond in a passive way. The part of the neuron being modelled is assumed to be uniform cylindrical core conductor having a length many times its diameter and so imposes the assumption that along the unit length of the cable their are uniform membrane properties. To model a dendrite, the dendrite is broken down into a number of sub-sections.

The compartmental model approach complements cable theory by overcoming the assumption that the membrane is passive. In the compartmental model, sections that are electrically short are assumed to be isopotential and are lumped together into a single 'R-C' (resistor-capacitor) circuit. Compartments are connected to each other via a longitudinal resistance according to the topology of the structure being modelling and thus differences in physical properties (eg diameter, membrane properties, etc) and differences in potentials are modelled between the compartments rather than within them. This allows

a compartment to represent a patch of neuronal membrane with a variety of voltage-gated and synaptic channels.

By using these types of models it has been shown in simulations that dendritic trees can perform sophisticated processing. It has been found for instance that the location of inhibitory synapses can be crucial as they are more effective when located on the path between the excitatory input and the soma than when placed distal to the excitatory synapse. When strategically placed, inhibitory inputs can specifically veto parts of the dendritic tree and not others. It has also been shown that distal arbors of a dendritic tree can act as coincidence detectors, whereas the soma acts more as an integrator, when brief synaptic inputs are involved.

Although, the models described have been extremely useful, they have some flaws that mean that reservations about their veracity has to be taken into account. These types of models display an unstable nature, the behaviour of any given model can change dramatically when relatively small modifications are made in its parameter values. Therefore a detailed model of a given neuron cannot be viewed as a unique description of it, even if the behaviour of the model accurately matches that of the biological neuron under some conditions.

One aspect that is ignored in many functional models is the use of spiking outputs as opposed to continuously-valued outputs. Up until this point only artificial neural systems using a non-temporal signal have been considered but as has been shown, in biological systems temporal aspects play an important role. Biological systems use 'spike-trains' to encode information travelling from one neuron to another. Spike-trains consist of a group of action potentials that propagate along the axon of a neuron to another neuron. There are at least two possible levels of information encoding occurring in these spike trains:

- the frequency of the pulses within the spike train can encode the information.
- the timing of individual pulses can encode the information

If the frequency alone is considered to encode the information content, then the types of nodes considered until this point can adequately capture this method. The output of the

node in a functional model can be considered to be a direct representation of the frequency of the spike-train which in turn relies on the membrane potential. If the frequency of the spike train is not enough to encode the information but the timings of the spike trains needs to be considered, then other models of neurons must be considered.

3.7 Modelling Biological Networks

As previously stated the modelling of BNNs occurs on many levels and the design of networks involving many model neuron must be addressed.

Again many models exist and this section outlines the general classes that are available. In order to model a BNN each and every connection, synapse and neuron must be modelled, and this information is obtained by looking at the BNN itself. As many of the complexities of a single neuron are still unknown it, it is an impossible task to model a BNN with the certainty that the model is correct but it does provide insights that can be useful, eg it has been found that many different network configurations could produce the same output behaviours (Pudipeddi, Abbot, & Athanas, 1998).

3.8 Concepts in the development of Biological and Artificial Networks

- genotype to phenotype encoding
- dimension of correspondence between genotype and phenotype (loose vs tight)
- developmental growth
- cell division
- axonal growth and branching

3.9 The Organisation of Neural Networks

3.9.1 Structural Organisation of Biological Neural Systems

Identified structures are :

- layers of processing elements — e.g. visual cortex has six layers
- columns of processing elements — layers can often be decomposed into smaller sub-networks
- specialisation of neural tissue into both specific and non-specific systems — e.g. specific systems mediate exact info through the sensory system to the cortex. Other areas are non-specific e.g. association areas, have a general learning ability.
- processing elements — neurons.

As previously stated in Section 3.2 neural networks consist of two basic components, nodes (neurons) and weights (connections, axons, processes). By varying the way these components work and how they are connected it can lead to a vast array of different network types, and there is then the need to categorise them in some manner.Maren, 1990 suggests that neural network architectures can be broken down into a number of levels :

- The Micro-level structure : neural element and transfer function
- The Meso-level structure : how is a network organised?
  - The number of layers in a network
  - The connection pattern between nodes
  - The flow of information around a network
- The Macro-level structure : conglomeration of networks to solve tasks

These levels of can be found in both biological and artificial systems.

Level	**Biological**	**Artificial**
Micro	Neuron, Axon, Dendrite	Node, Weight
Meso	Layers, Columns	Layers
Macro	Cortical Systems	Hybrid Networks

Table 3.3: Defining Levels of Architecture

3.9.2 Micro-level structures

In artificial systems there are the following features :

- Activation — what range of values are output from a node (1–0, -1–1, ...)? Is the activation continuous or is binary?
- Weights — How are they used in conjunction with the node/transfer function? Are there temporal aspects to the weights, e.g. decay ?
- Transfer function
- Bias function
- Gain function
- Threshold function

The transfer function affects the input/output mapping of the node. Four typical transfer functions are (see Figure 3.9 :

- Threshold logic nodes — activation is binary, greater than threshold activation is 1, less than threshold activation is 0 (e.g. Hopfield/Tank networks)
- Hard limit nodes — a defined upper and lower bound with a linear relationship between. If input $\geq$ upper bound activation = 1, input $\leq$ lower bound, activation = 0. If input is somewhere between the two, the linear relationship applies.

- Continuous function nodes (sigmoid) nodes — most commonly used, backpropagation learning relies on it. Most common is sigmoid but can use any function which is smoothly defined over the interval from minus to positive infinity, e.g. arctangent function. The reason backpropagation learning requires a sigmoid type function is that it relies on differentiation of the output activation, threshold logic functions give undefined results at the transition point and zero everywhere else. Sigmoid is defined as $y = 1/(1+\exp(-\alpha x))$.

- Radial basis functions — not so common, typically uses a Gaussian curve. The centres and widths of the functions can be varied which gives them more flexibility than the sigmoid function. Mappings which require two or more layers using sigmoid functions may be able to "squashed" down to one layer using a radial basis function.

Detail of Nodes vs Population of Nodes important (Kolmogorov theorem – any function, no matter how complex, can be represented by a multilayer perceptron of no more than three layers)

3.10 Conclusion

Having investigated ANNs and BNNs the research story moves on to a technique for the construction and development of neural network, i.e. Genetic Algorithms.

4

Genetic Algorithms

4.1 Introduction

In the chapters following this one, techniques for the construction and development of neural networks will be discussed. One of these techniques is the genetic algorithm (GA), and so it is appropriate to introduce the concepts of genetic algorithms at this point.

The biological underpinnings from which genetic algorithms have been developed will initially be outlined, followed by an overview of the procedures and methods used to develop solutions to optimisation problems using GAs. Finally, a brief discussion of the advantages of genetic algorithms over other optimisation techniques will be presented.

4.2 The biology

A genetic algorithm is part of a larger group of algorithms termed *evolutionary algorithms* (EAs), a group of stochastic optimisation techniques, which draw inspiration from two areas of research; the principles of Darwin's theory of evolution and genetics.

4.2.1 The biology of Genetics

Every cell of every living organism contains a "blueprint" of how that organism should be constructed, and the structure that encodes the prescription of how an organism is to

be constructed is termed a *chromosome.* Chromosomes, threadlike bodies in cell nuclei, are composed of genes, linearly arranged, which carry genetic information responsible for the inherited characteristics of the organism. Chromosomes are constructed from deoxyribonucleic acid, or DNA for short, attached to a protein core. All normal cells contain a certain number of chromosomes characteristic of the species (46 in man), in homologous pairs (diploid chromosomes). *Gametes*, the germ cells of an organism, are however *haploid*, having only one of each of the pairs found in normal cells, so that during reproduction when a child organism is formed, from the joining of the two gamete cells of the parents, it will have the correct number of chromosomes.

A chromosome can be subdivided into a sequence of *loci*, or positions on the chromosome, and at each locus a *gene* can reside. The set of genes that are associated with a specific locus are called *alleles*. In humans, where there is a pairing of chromosomes, there will be a two genes associated with each locus on the paired chromosomes. When a cell is being constructed only one of these genes will be used (expressed) depending upon a number of factors. One of these factors is whether the genes involved are *dominant* or *recessive* genes. If one of the genes is dominant then that gene will always be expressed, only in the case where two recessive genes are coded in the DNA will the recessive gene be expressed. Environmental factors can also influence what genes are expressed, as can the influence of other genes. The interaction between genes where one gene influences the expression of other genes is called *epistasis*.

The *genotype* of an organism is the total genetic makeup of the organism, consisting of all the genes received from it's parent or parents. The actual organism that results from the interaction of the genotype and the genotype's environment is called the *phenotype*. Different genotypes may give rise to the same phenotype, or a genotype may give rise to different phenotypes depending upon the interaction of the genotype and it's environment.

The genetic material of an organism is passed on to it's offspring in the process of reproduction. There are two forms of reproduction, asexual reproduction and sexual reproduction. *Asexual reproduction* essentially involves the duplication of a single parent's genetic material and is most often seen in the world of single celled organisms. If no errors were introduced during the process of asexual reproduction, then the child would be

genetically identical to the parent. *Sexual reproduction* is where the genetic material from two organisms is combined to produce a child organism which is genetically different to both parents. In an organism with a diploid chromosome structure, the cells (*gametes*) involved in the transfer of genetic information are typically haploid. During the production of these cells each of the two genes of a particular locus has a 50% chance of ending up at the locus of the resultant gamete. When the gametes of the parent organisms are joined together to produce the first cell of the child organism, the two haploid cells are fused to produce a diploid cell again, thus keeping the same chromosomal structure as the parent cells; ie the two haploid cells join to form a single diploid cell.

The process of reproduction is not an error-free process and the result of these errors introduce *mutations* into the genetic material of the child organism. Two forms of mutation are of particular interest to the study of genetic algorithms, *chromosome mutations* and *gene mutations*. Chromosome mutations involve sections of a chromosome becoming interchanged with other sections of the chromosome or in some cases in diploid chromosomes sections from each of the paired chromosomes become interchanged. This is also called recombination. Gene mutations occur within a single gene and so are localised to a specific locus within a chromosome. If the gene is involved with the switching on or off of other genes (epistasis), this can lead to large changes in the phenotype of the child organism.

4.2.2 The biology of Evolution

As previously stated GAs are a form of EAs which are techniques that share an underlying problem solving mechanism broadly based on theories of evolution, and the principle of 'survival of the fittest' as presented in the seminal work by Darwin (1859). Evolution describes the change in the gene pool of a species over time. Two categories of evolutionary mechanisms can be identified, those that increase the variation of genetic material over time and those that decrease it. Mutation is one of the main contributors of increasing genetic variation but others exist such as gene flow, the introduction of new genetic material from another population of the same species by means of migration. *Natural selection*

and *genetic drift* are just two mechanisms for decreasing the variation in genetic material between individuals within a population. Natural selection is the mechanism where individuals whose genotype/phenotype is best matched to the prevailing environmental conditions are most likely to survive and pass on the genetic material to their offspring. Thus the most favourably part of the variation within the species is preserved.

According to Darwin, evolution occurs when an organism is confronted by a changing environment. A degree of genetic variety is always present in the members of an interbreeding population. Normally, the possession of a variant characteristic by an individual confers no particular advantage on it, and the proportion of individuals in the population with a given variation remains constant. But if it ever arises in a changed environment that a given variation increases the chances of an individual's survival, then individuals possessing that variation will be more liable to survive and breed. The frequency with which the variant character occurs in future generations of the organism will thus increase and over a large number of generations, the general form of the population will change. The name "natural selection" derives from the analogy Darwin saw between this selection on the part of "Nature" and the "artificial selection" practiced by animal breeders.

Genetic drift is the process by which genetic information controlling certain characteristics is lost to the population as a whole because it is not transmitted to the offspring of the population. Generally genetic drift only occurs in small isolated populations. In large populations the genetic material for any specific characteristic is carried by so many individuals that unless it is unfavourable its loss is highly unlikely.

The theory of natural selection is often referred to as "survival of the fittest" but this is a misleading term because it is not the survival of the organism that is the driving force of evolution but rather it is the contribution of the fitter organism's alleles to the next generation's gene pool. Darwin himself did not use the phrase himself until the third edition of "The origin of species" (Darwin, 1859), the phrase being actually coined by the British philosopher Herbert Spencer, a contemporary of Darwin's. The fitness of an organism is just not a measure of it's physical abilities but it's ability to attract mates for breeding and demonstrating it's fitness to potential mates. As Richard Dawkins puts it evolution is like a "blind watchmaker", evolution has no sense of purpose or direction,

there is no overall plan or goal, as time progresses, organisms become better adapted to there current environment.

4.3 Genetic Algorithms

This section briefly describes a type of evolutionary algorithm, the Genetic Algorithm (GA). GAs are based on an iterative process, as shown in Figure 4.1, which mimics the process of evolution. The process begins with the creation of a population of potential solutions to the problem being solved.

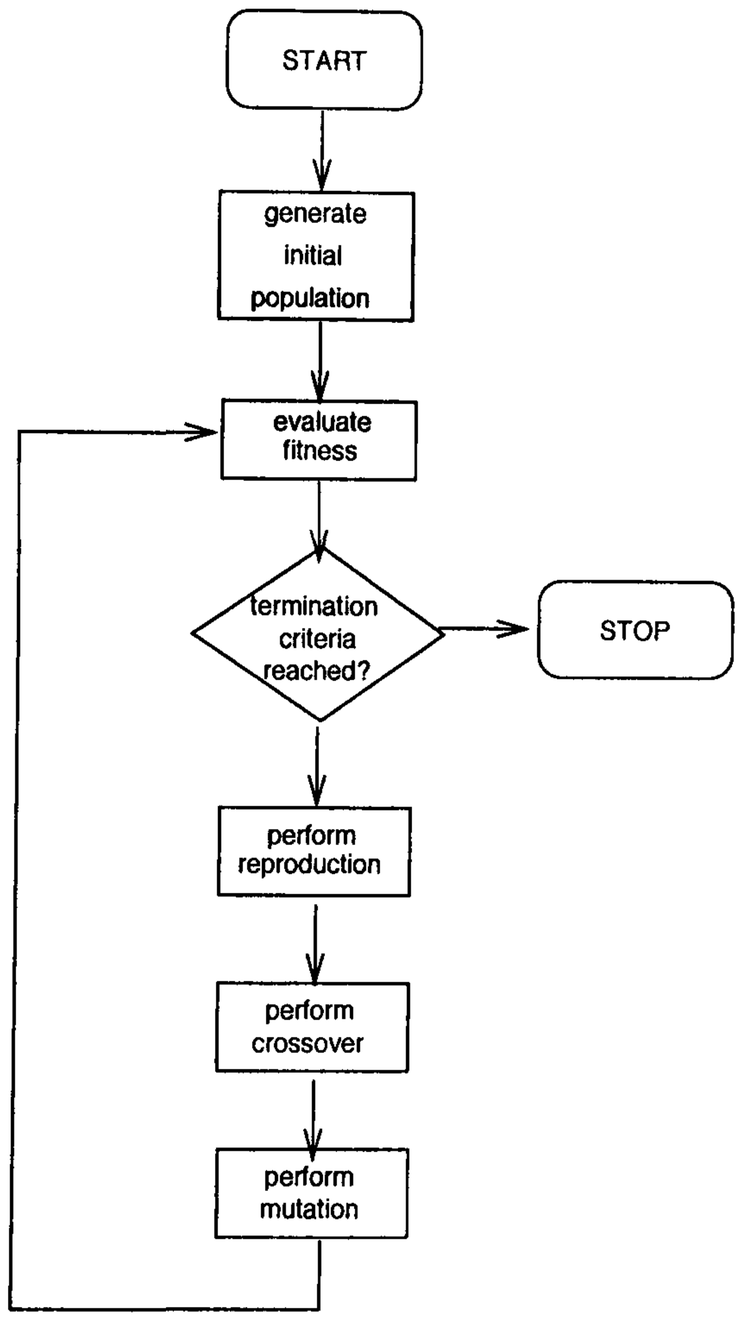


Figure 4.1: Flowchart of the GP process

GAs maintain a population of structures that represent solutions to the problem to be solved from a number of points over the search space. This population is akin to a population of a single species found in Nature. Genetic operators, such as reproduction, mutation and recombination, manipulate the population of structures to create better representations. Individuals are measured on a basis of how well they solve the problem (an

individual's *fitness*), with those individuals with greater fitness surviving in the population longer. Reproduction focuses attention on high fitness individuals, thus individuals with higher fitness reproduce more. Reproduction and mutation provide mechanisms for exploring the search space.

The basic unit of a GA is a representation of a solution is the genotype, or as its sometimes termed the *encoding string*, and is analogous to the DNA found it animals which evolution works through. In nature, DNA consists of sequences of four base pairs and these four base pairs in different combinations encode different proteins and amino acids that are used to construct the cell structures of the animal. In a similar way, GA genotype encodings, in the simplest types of GA, are based on binary string representations. The alleles of a binary string are thus '0' and '1'. Typically, the parameters of the problem to be optimised are turned into binary values and concatenated. The length of the string and the encoding of the string is dependent on the problem and will be discussed at greater length in the following sections. For instance if the value of x was to be optimised for the function $f(x) = x^2$ over the integer range 0–31, then only one parameter needs to be encoded, x, and can be represented by 5 bits, '00000–11111'.

In the first instance, an initial population of genotypes is produced. The actual number of genotypes in the population depends again on the problem. In the example function of $f(x) = x^2$ over the integer range 0–31, a population of 32 genotypes, each different from one another, would in actual fact cover all possible values of x, and is thus not different from an enumerated search algorithm. Thus a balance between having a representative sample of genotypes covering the search space and having too many genotypes and risking the inefficiencies of the enumerated search is required. The size of the populations will depend on the complexity of the problem, and can range from tens of individuals to hundreds of thousands for very difficult problems. As yet there is no definitive algorithm to determine what the population size should be, only rules-of-thumb.

Associated with each of the genotypes is a *fitness value* which is evaluated by the *fitness function*. The fitness value is a measure of how well the solution represented by the genotype solves the problem. The fitness value is used in the *reproductive* genetic operator

to determine which genotypes, and the number of copies, survive from the current generation into the next generation. In nature fitness is determined by an animal's ability to survive and reproduce in its environment by avoiding predators, pestilence and other obstacles. The fitness value is analagous to this. This selection is like natural selection in nature. Individuals with a higher fitness are more likely to be chosen than those with low fitnesses. In natural evolution, those animals which are less capable of surviving in their environment die, and those which are stronger survive and their genetic information propagates to subsequent generations. A number of reproduction schemes exist but the simplest and most commonly used is termed the *roulette wheel* method. In this method each genotype is allocated a number of slots on a "roulette wheel". The number of slots allocated to each genotype is determined by the proportion of its fitness value to the summed fitness of all the genotypes. The greater the fitness value of an individual genotype, and hence its greater contribution to the fitness of the population as a whole, the greater the number of slots on the roulette wheel. Thus if a genotype's fitness is 20% of that of the summed fitness, 20% of the slots on the roulette wheel will be allocated to that genotype. To determine what genotypes will make up the next generation, the roulette wheel is spun and whichever slot 'wins', a copy of the genotype that owns that slot is put into the next generation. This process is repeated until the next generation has the allocated number of individuals. The new generation will consequently have more copies of the fitter individuals than the less fit individuals, but through the stochastic nature of the selection process, less individual members will also be present, thus keeping some variation in the genetic nature of the population. The genotypes destined for survival to the next generation are not immediately copied into the next population but are instead held in a temporary "mating pool" ready for the mutation and recombination genetic operators. The individuals in the subsequent generation of this simulated evolution are created either by directly copying them, by making a composite of two individuals, or by making a mutated copy of a selected individual. The genetic operators used mimic asexual reproduction, sexual reproduction and genetic mutation in biology.

Recombination, also called *crossover*, in the simplest case involves taking pairs of genotypes from the mating pool and swapping parts of their coding structures. The first stage

is to pair up pairs of "parent" genotypes. The second stage is to select a point along the coding structure that will serve a swapping point. This point is usually random selected from along the length of the coding structure. Two new "child" genotypes are then created by swapping the selected portions of the coding structures at the swap point (Figure 4.2. The two child genotypes replace the parent genotypes in the mating pool. This process continues for all pairs of the parent genotypes. The process of recombination follows the theory that certain sections of a genotype code for certain parts of a solution. Some of the sections of the genotype will contribute more to the fitness than other sections, and recombination allows different combinations of sections come together. In the example of the function $f(x) = x^2$, ...

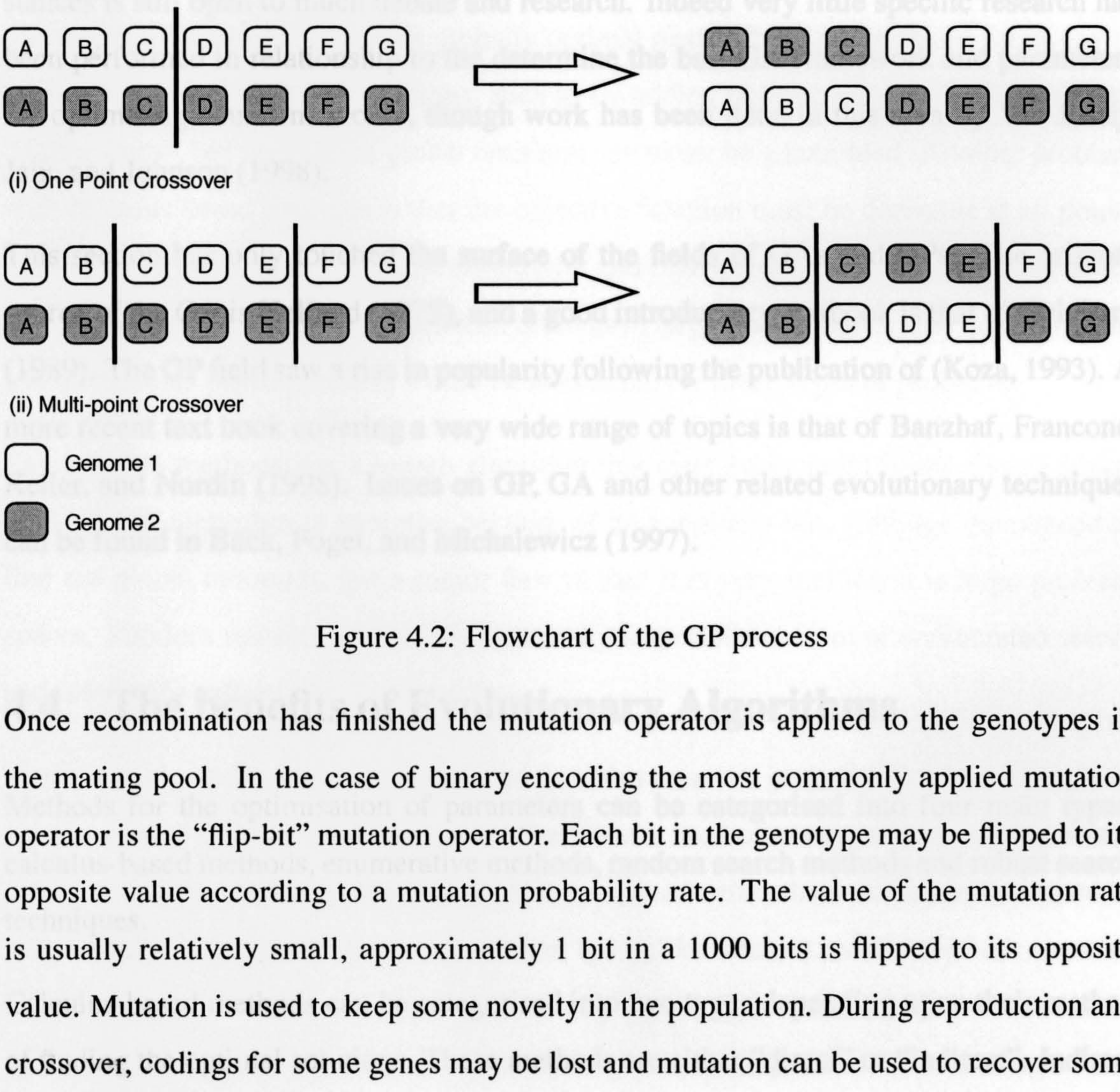


Figure 4.2: Flowchart of the GP process

Once recombination has finished the mutation operator is applied to the genotypes in the mating pool. In the case of binary encoding the most commonly applied mutation operator is the "flip-bit" mutation operator. Each bit in the genotype may be flipped to its opposite value according to a mutation probability rate. The value of the mutation rate is usually relatively small, approximately 1 bit in a 1000 bits is flipped to its opposite value. Mutation is used to keep some novelty in the population. During reproduction and crossover, codings for some genes may be lost and mutation can be used to recover some

of the lost codings.

The process of evaluation, selection and insertion is then repeated until a set of predefined termination criteria are reached, for instance, a solution is found, or a timeout has been reached.

As already stated the above, a GA is one of the simplest frameworks. There are many different types of recombination, reproduction, mutation and coding schemes that have been devised to make GA's more efficient in many different problem areas.

Each GA simulation run is controlled by a set of parameters. These include the size of the population, the frequency with which different genetic operators are applied, and the termination criteria. Guidance for the settings for these values under different circumstances is still open to much debate and research. Indeed very little specific research has been performed in relationship to the determine the best GA framework and parameters for optimising neural networks, though work has been done in this area by Van Rooij, Jain, and Johnson (1998).

This section has only touched the surface of the fields of GAs and GPs. The original source of the GA is Holland (1975), and a good introductory textbook is that of Goldberg (1989). The GP field saw a rise in popularity following the publication of (Koza, 1993). A more recent text book covering a very wide range of topics is that of Banzhaf, Francone, Keller, and Nordin (1998). Issues on GP, GA and other related evolutionary techniques can be found in Bäck, Fogel, and Michalewicz (1997).

4.4 The benefits of Evolutionary Algorithms

Methods for the optimisation of parameters can be categorised into four main types, calculus-based methods, enumerative methods, random search methods and robust search techniques.

Calculus-based methods can be categorised into two types depending upon their method of finding the optimal solutions. These methods are either "direct" or "indirect". Indirect

methods solve the optimisation problem by finding local extrema in the set of equations describing the problem. This is usually achieved by setting the gradient of the objective function to zero and solving the equation to find the relevant parameter values. Direct search methods find local optima by moving in a direction related to the local gradient (the notion of *hill-climbing*). One problem that calculus based methods can encounter is that of *local minima.* Calculus based methods rely on information obtained from the local search space and so are not "privy" to the global picture, and can thus be lured into solutions that are not globally optimal. Consider the function shown in Figure 4.3, where there are a number of peaks and a calculus based search method could be employed to find the maximum value of the function. Using a simple calculus based hill-climbing method, the starting position of the search is all-important; if the search was started at position X, the search will find the locally optimal peak at X1. By starting at a different location the search method could find the globally optimal peak. There are methods for trying to avoid these locally optimal positions, such as by adding noise to the search or by adding "momentum", but finding the global optimum can never be guaranteed. Another problem with calculus based methods is that the objective function must be derivable at all points in the search space but often, with non-trivial and real-world problems, the search space is scattered with discontinuities and can often have large multimodal, noisy search spaces and so calculus based methods would be unsuitable for these types of problems.

Enumerative methods use a search algorithm that tests every point in the search space. This type of algorithm is attractive because of its simplicity but, although guaranteed to find the global optimum, has a major flaw in that it is very inefficient is large problem spaces. Random search methods are essentially a specialised form of enumerated search and ultimately suffer from the same flaw.

The robust algorithm that will be referred to in this case is a *genetic algorithm* (GA). EAs are considered to be more robust than other search methods because they sample many points of the search space, like random search methods, thus the overall search algorithm is less likely to get stuck in a local minima, but unlike random search methods can also concentrate on local areas of particular interest. Working from many points in the search space simultaneously, climbing many peaks in parallel, the probability of finding a false

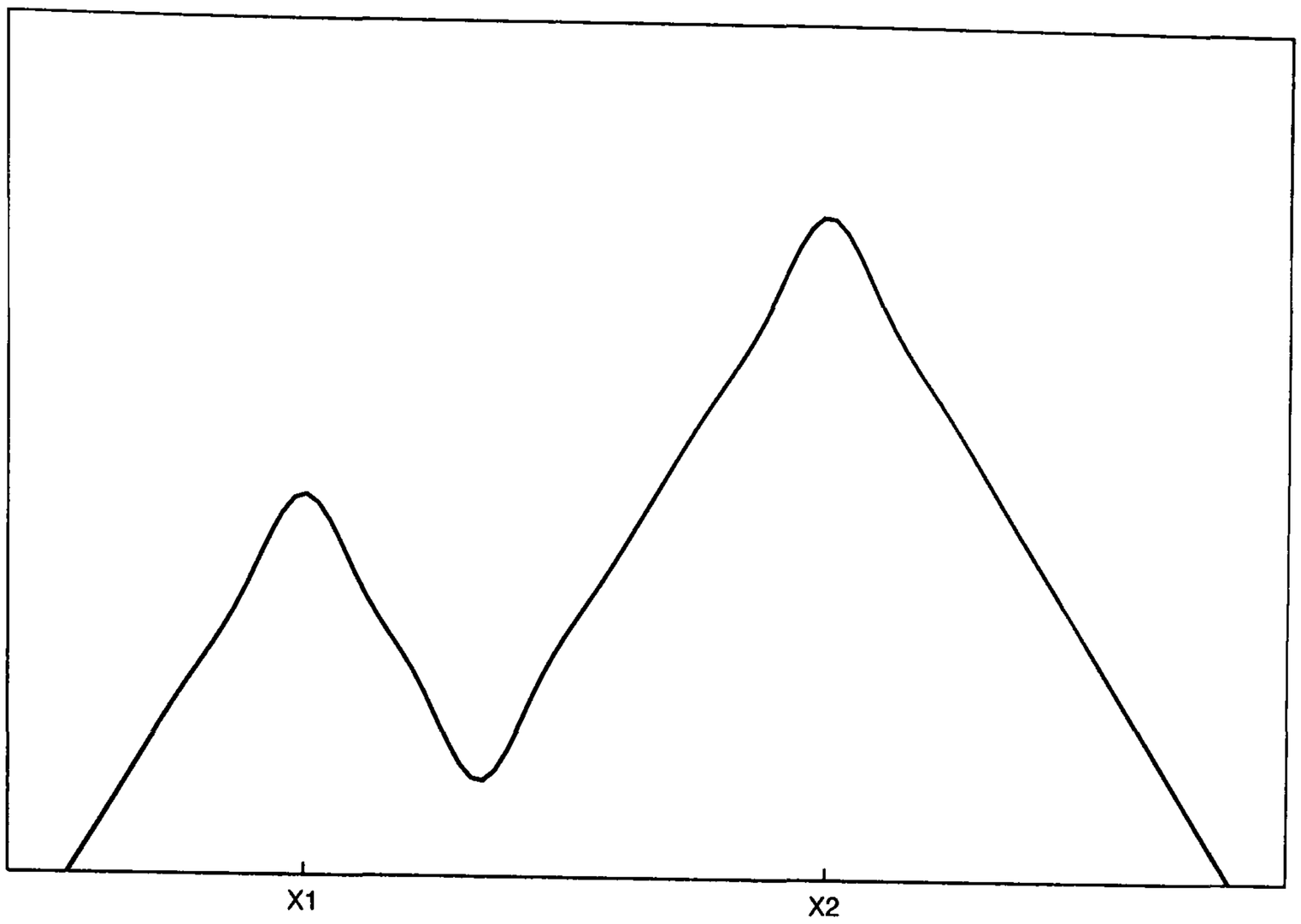


Figure 4.3: Graph of an objective function containing a local minima

peak is reduced. As Goldberg states (Goldberg, 1989, pp9) "there is safety in numbers".

The selection of representation structures to concentrate on does not have to depend on any derivatives of the function being optimised. The EA works with a coding of the parameter set, not he parameters themselves. The genetic operators mutate, rearrange the structure of the representation but it does not use any of the actual values of the parameters. The only value that the EA needs in order to improve the representations is a value of fitness. Probalistic transition rules are used rather than deterministic ones. EAs use random choice as a tool to guide a search toward regions of search space with likely improvement.

The bottom line is that GAs provide a robust search algorithm for complex spaces where it is not obvious what the function governing the problem space is. When the terrain of the search space is unknown, where gradients or derivatives may not exist for all of the search space, where the search space may be very large, EAs can still be used.

It has been shown that GAs are more robust than traditional optimisation algorithms, especially where the objective function to be optimised has a unknown and complex landscape. Goldberg (1989) gives a useful description, Figure 4.4 of how the techniques are related to one another on a relative scale of efficiency. The different types of search method are

compared to one another on the basis of efficiency in three different types of problem landscape. Under the right conditions a calculus based search method outperforms all other methods, but under the wrong conditions its performance is severely limited. An enumerated method performs equally well under all conditions but at a relatively low level of performance. The robust scheme performs well across all situations at a much higher level than both the calculus based methods and the enumerated methods except in a small subset of the search space where it is outperformed by the calculus based method.

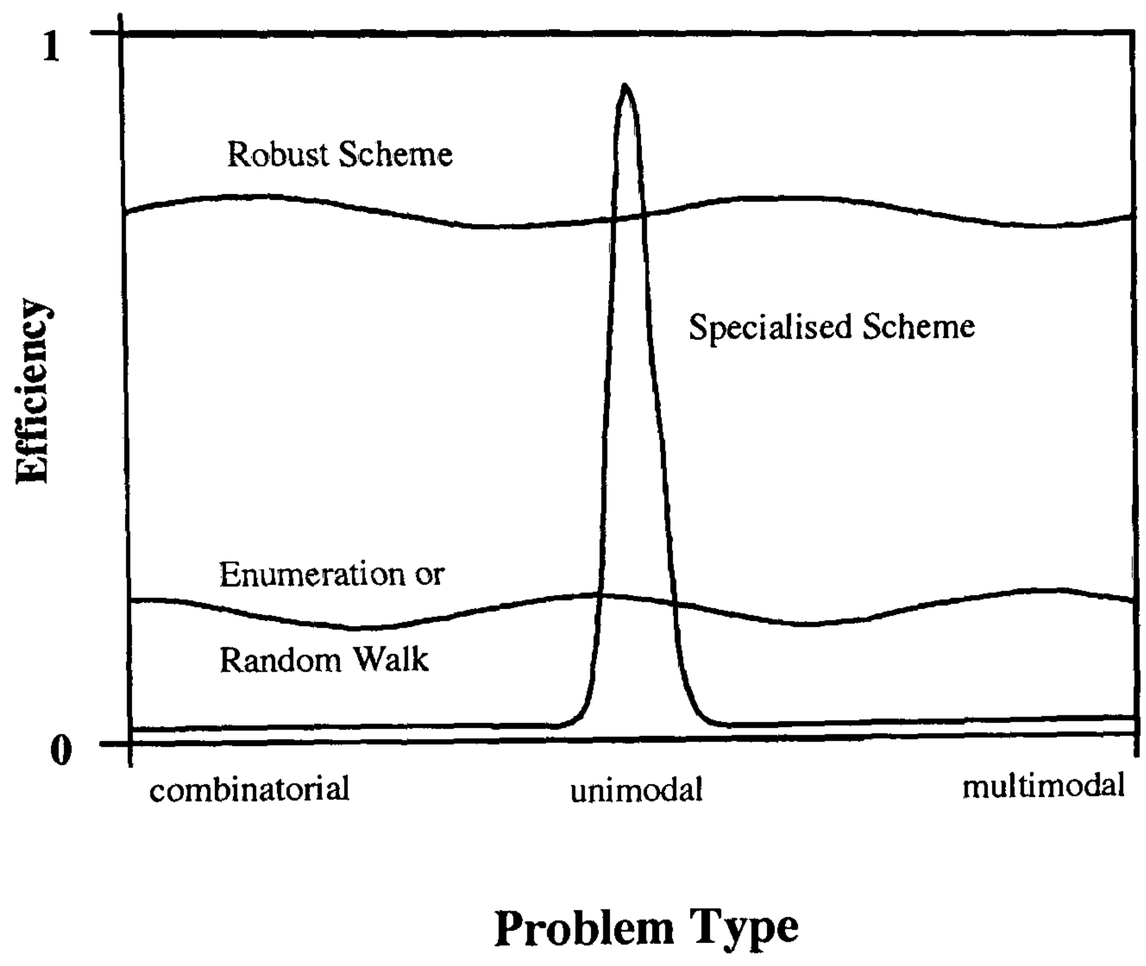


Figure 4.4: Graph of relative efficiencies of search techniques

4.5 Conclusion

The structures underlying biological neural networks have arisen through a process of natural evolution. This chapter has looked at how natural evolution can be applied in a computer to solve complex problems. We now move onto the construction part of the research story.

5

Artificial Neural Network Construction

5.1 Introduction

In this chapter we look at how to construct ANNs in order for them to be most able to solve the problems to which they are applied. We consider different types of construction as a precursor to modelling some of these later in this dissertation.

5.2 Meso-Level Network Design and Learning

The output of a ANN in response to an input primarily depends on the network topology and the strength of connections between the nodes of the network. Networks with the same basic topology can be used to solve different problems by using different weights between the nodes. The changing of the weights to produce a network that has an output response that is closer to the required output is called 'learning' or 'training'.

The task of a neural network training algorithm is to set the weights of the network to represent the hyper-planes.

Formation of the network topology can be achieved through a variety of methods. The network topology can be chosen by the designer of the network and remain static throughout the lifetime of the network. Alternatively, the network topology can vary during the formation of the network. It can be constructed from a small number of nodes and during

the development of the network, nodes and connections are added to the network until the network can solve the problem set to it. The network can start from a large number of nodes and during the development of network nodes and connections are deleted from the network until the network can solve the problem set to it.

The method for setting the strength of the connections between nodes is intertwined with the choice of network topology that is employed. The choice of how a neural network will learn to perform the task it is designed to solve depends primarily on the structure of the network topology involved. In most networks, the network is constructed first to give the desired topology and then the weights between the nodes are adjusted by a learning algorithm. In some network construction techniques the weights are adjusted during construction, performed in a construction/learning cycle until the desired network has been achieved. In other networks the weights are assigned during construction and are not adjusted again and so essentially no learning is achieved during the lifetime of the network, ie the network weights can be considered to be 'hard-wired'.

The methods used to design a network can be broken down into two types, those methods that modify the weights only with the basic topology of the network being left to the designer and those methods which modify the weights and the network topology. The design methods can be broken down along another axis, those methods which search the problem space from one point only and those methods which explore many points in the problem space simultaneously. Typically these latter design methods employ evolutionary optimisation methods.

For each type of the network topologies there are many network formation methods and for each of those methods there are many possible learning algorithms and so it would be impractical to describe each of them. A sample of techniques will be described to give a flavour of the main categories of methods for feed-forward networks and recurrent networks as these will be the main focus of the thesis.

5.3 Defining Static Network Topologies using single networks

The most commonly used network topology is the multi-layer perceptron model (as described in Section 3.2). The simplest form of this model is a static network formation technique where the number of layers and the number of nodes in each layer is preset by the designer. The weights are adjusted after network formation by a learning algorithm, of which one of the most commonly applied learning algorithms is the back-propagation method. The back-propagation method relies on an iterative process to reduce the amount of error between what the network outputs and what the required output should be from a training set of data. The error of the network can be viewed as a function of the weights. By changing the weights, and hence the classifications the network will produce, the error of the network will change. Figure 5.1 shows a graph of the network error (E) against the Weight Space (the domain representing all the combinations of possible weight values for each connection). The network error can be used to guide the search around the weight space to find the set of weights that will give the minimum network error.

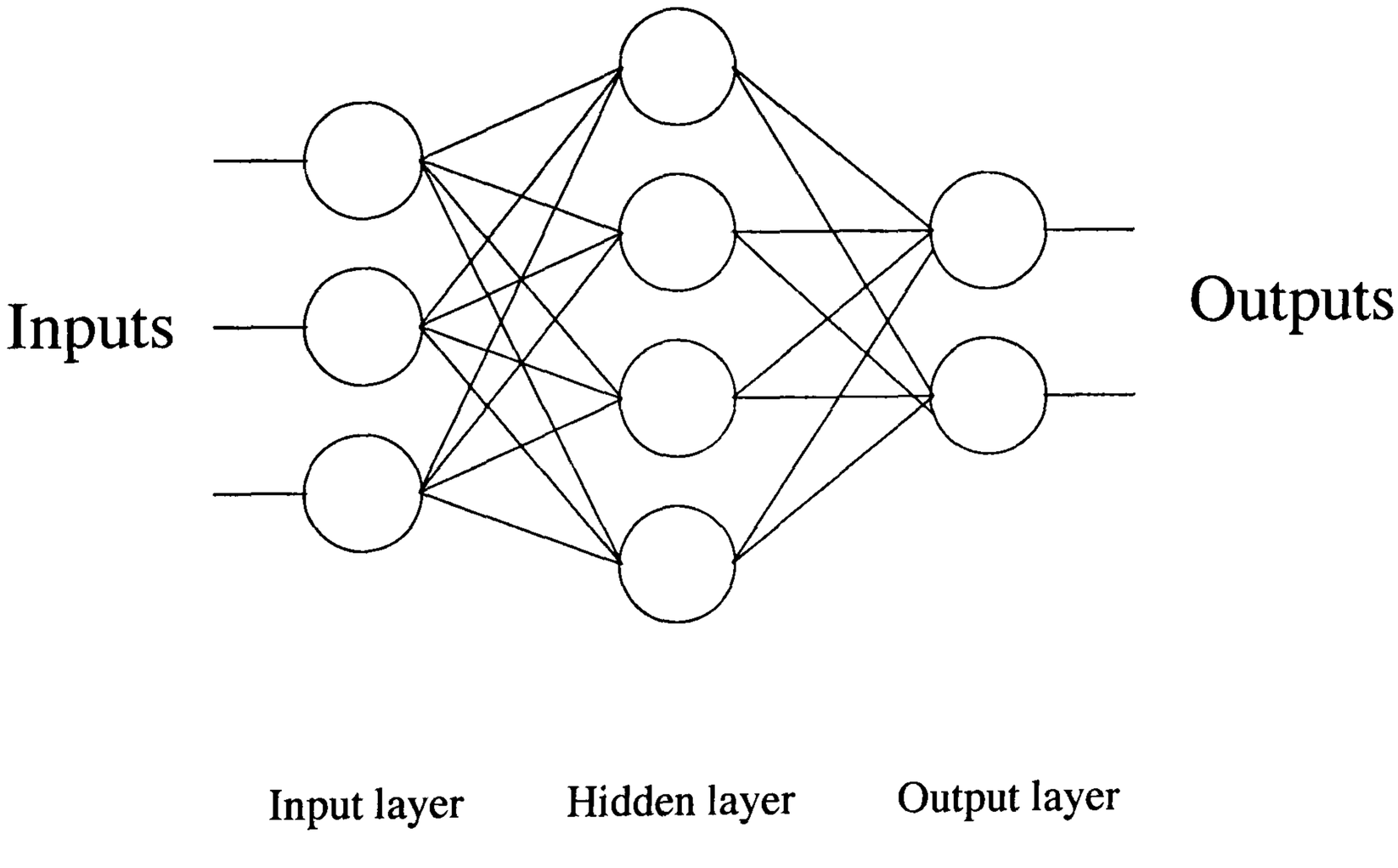


Figure 5.1: Plot of Network Error against Weight Space

The back-propagation algorithm uses a gradient descent method to navigate the weight space to find the optimally lowest point and so the least error. Given a network architecture of one input layer, one hidden layer and one output layer, the algorithm is:

1. tower
2. tiling
3. upstart
4. cascade-correlation
5. Initialise the weights and thresholds of the nodes to small random values.
6. Repeat
7. for each input/output exemplar in the training set
   (a) present the input of the current exemplar to the network and feed forward/propagate this forward from input to hidden layer and from hidden layer to output layer
   (b) calculate the error for each node of the output layer of the network and change the weights connected to each output node in proportion to the error for that node
   (c) calculate the error for each node of the hidden layer of the network and change the weights connected to each hidden node in proportion to the error for that node
8. Until the error is sufficiently low

The weight change for an output node is calculated by $\delta w_i^j = \alpha\sigma'(a^j)(t^j - y^j)x_i^j$, where j is the jth node in the output layer, i is the ith node in the hidden layer. The term $(t^j - y^j)$ gives a measure of the error on the jth output node. The term $\sigma'(a^j)$ gives a measure of how quickly the activation can change the output of the node and so the error of the node.

As can be seen from the algorithm the errors of the hidden nodes are based in part on the errors of the output nodes, or in other words the errors of the output nodes are propagated backwards to the hidden nodes, and hence the name 'back-propagation'.

This scheme can be applied to a network with any number of hidden layers by repeating steps b) and c) on each of the hidden layer nodes. This scheme has also been further developed to cope with recurrent connections.

The number of layers and the number of nodes in each will of course determine what classifications the network can learn. Although the network may learn the classifications the designer has already predetermined what patterns can be classified by choosing the number of layers and number of nodes. One possible, and naive, solution to this is to have a large number of nodes and layers in the network, which will allow the network great flexibility to learn complex pattern sets. Apart from increasing the weight space dramatically and so slowing the learning process down, this greater flexibility can be a 'double-edged sword' Too much flexibility allows the network to 'over-fit' the data and so reduces the ability of the network to generalise. Too few nodes and the network may not be able to classify the training patterns given to it.

Juedes and Balakrishnan (1998) suggest a scheme that uses BP as the back bone of the learning algorithm. Rumelhart and McClelland (1986) has shown that a multilayer network can be trained using a BP algorithm as long as the threshold function is differentiable (e.g. use a sigmoid function) [1].

5.4 Defining Non-Static Network Topologies using single networks

The size of the network (also called the *model complexity*) is related to the performance of the network. If the network has too few degrees of freedom, too few weights, for example, the network may not be able to achieve a good fit to the data. Too many degrees of freedom on the other hand, can lead to a network which is poor at generalisation due to over-fitting the data (memorisation), and slow to learn. For good performance, the network should be sufficiently large enough to solve the problem but not overlarge to avoid memorisation. The designer of the network can of course, through experimentation and experience, vary

[1]Saarinen,Saarinen, Bramley, and Cybenko (1991) (in Juedes & Balakrishnan, 1998) show that back-propagation is essentially the reverse mode of computational differentiation. Yoshida,T (1992) (in Juedes & Balakrishnan, 1998) uses computational differentiation and two-dimensional conjugate gradient search to train neural networks. The algorithm ADOL-C, see Griewank 1996, is a computational differentiation algorithm that can be used in place of the standard BP algorithm

the network size manually on different training runs to find an optimally sized network. The disadvantage of such an approach is that the designer may unintentionally bias the network design and opt for a non-optimal network. Also, by varying the size of the network, training algorithms may prove to be more efficient and so training times may be reduced. Many researchers have been developing automatic methods for choosing the optimally sized network. The advantage of using techniques that vary the topology of the network is that the final network topology is not biased towards a design that the designer has chosen and which may be non-optimal.

The design of non-static network topologies can be broken down into two types, those that construct a network by starting from a small number of nodes and adding new nodes, and those networks that start from a large number of nodes and remove, or 'prune', nodes that are deemed unnecessary. Constructive methods offer a natural starting point for the initial architecture, usually only the required number of nodes needed to represent the input and output. The networks, due to their initially smaller size, can benefit from quicker training. However, the size of the network must be carefully controlled otherwise the network may quickly grow without restriction. By starting with a large network and pruning the size of the network, it is generally the case that the network can easily learn the proper input-output mapping and litle chance that the network is too simple a model. A natural stopping point is easily identifiable when the network error reaches an unacceptable level. The disadvantages of pruning include initially longer training times and the possibility that the network may not reach an optimal structure because of local minima in the error-surface.

The Tiling algorithm is an example of a constructive algorithm, developed by Mezard and Nadal (1989), creates a multi-layered network with each successive layer having fewer nodes than the previous layer, until the final output layer has a single node.

The Dynamic Node Creation (DNC) algorithm constructs networks by adding fully connected nodes to the hidden layer of three-layer networks during back-propagation training (Ash, 1989). The network starts with a single hidden node. Back-propagation training takes place until the desired mapping is learned or until another hidden node need to be added to the network. The procedure is repeated until user-specified stopping criteria are

met. A sliding time history window of a specified width is kept for the network's average output error. The drop in error since the creation of the last hidden node is calculated and if the error has not decreased quickly enough over the width of the history window, a new node is added to the hidden layer.

Fahlman and Lebiere (1990) have developed the Cascade Correlation method to construct neural networks. The network starts out with no hidden nodes and with the input layer connected directly to the output layer. New hidden nodes are inserted into the network one at a time and each new node is connected to all of the inputs as well any of the previously inserted hidden nodes. The network is trained in the first instance using a variation of the back-propagation algorithm. If the network error is deemed to be low enough the algorithm terminates, otherwise if the error has not reached a low enough level after a user-defined period, a new hidden node is added to the network. The new node is selected from a group of candidate nodes that are trained to maximise their output's correlation with the residual error errors on the network outputs. The one that maximises this correlation, and hence has learned a feature that correlates highly with the residual error, is inserted into the network. The input weights for this node are not trained but the weights that connect to the output nodes are retrained in order to incorporate the contribution of the new intermediate node.

Destructive techniques take the form of either removing (pruning) connections between nodes or removing nodes to enhance the efficiency of learning. Generally The network is designed as a fully-connected network with an over-abundance of nodes and connections. During training the network is pruned to remove connections or nodes that do not contribute to the solution. The size of the network is thus reduced over time. One simple technique is that of *weight elimination.* Weight elimination works by using the values of the weights as an indication of how important the connection is the network as a whole. The algorithm works by creating a driving force that will attempt to force all weights towards zero during training. If the input-output mapping of the network requires some large weights, learning will keep pushing up these weights, but ones that are not important will be driven to zero; unimportant weights decay to zero. Weight decay can be implemented by adding an extra term to the weight update of a backpropagation algo-

rithm; $w_{ij}(n+1) = w_{ij}(n)(1-\lambda) + \eta\delta_i x_j$, where λ is the weight decay constant, and the term $w_{ij}(n)(1-\lambda)$ will try to reduce the value of the weight towards zero. Weight values smaller than a threshold value can be eliminated from the network entirely, reducing the overall degrees of freedom of the network. There are other variations on the theme of weight decay (Bishop, 1995; Reed, 1993).

Optimal Brain Damage (OBD) by LeCun, Denker, and Solla (1990) is a post-training method that removes selected weights in order based on a notion of saliency. The OBD method calculates an approximation to the actual effect on the error of deleting each weight in the network and from that derive a measure saliency. The network is trained in the normal way, and then the saliencies are computed for each weight. The weights are then ordered in terms of their saliency, and a percentage with smaller saliencies are discarded. The network is then retrained using the weights of the previous training cycle and this process of training and removing less salient weights is repeated until an optimal network is achieved. This method improves upon the previous method of weight decay due to the concept of how much the weight affects the network rather than the crude measure of weight magnitude only. Again variations on this theme have been developed (Hassibi & Stork, 1993).

Sietsma and Dow (1991) used a network pruning technique to demonstrate that the performance of the network varies with the size of the network. Their techniques relied analysing the response of nodes to the patterns of the training data set and eliminating those nodes and their associated connections that were unused in the classification process.

5.5 Details of Evolutionary Systems

A candidate for this type of network design has already been used but in a very constrained and elementary manner. This method relies on a category of automatic design approaches based on evolutionary techniques (Koza, 1993, 1994; Goldberg, 1989; Michalewicz, 1996). The design of neural networks is essentially the same as *program synthesis* as researched by Uhr, 1973 and genetic programming Koza, 1993 and given the equivalence of

many models of computations (e.g. lambda calculus, Turing machines, Post productions, etc), the choice of model is largely a matter of convenience (Balakrishnan & Honavar, 1997). Though it should be noted that some models are better suited to some types of problems in terms of cost/advantages/efficiency of implementation/design/processing.

The evolutionary approach is based on its analogue in nature. Figure 5.5 gives an overview of how evolutionary techniques can be used to design neural networks.

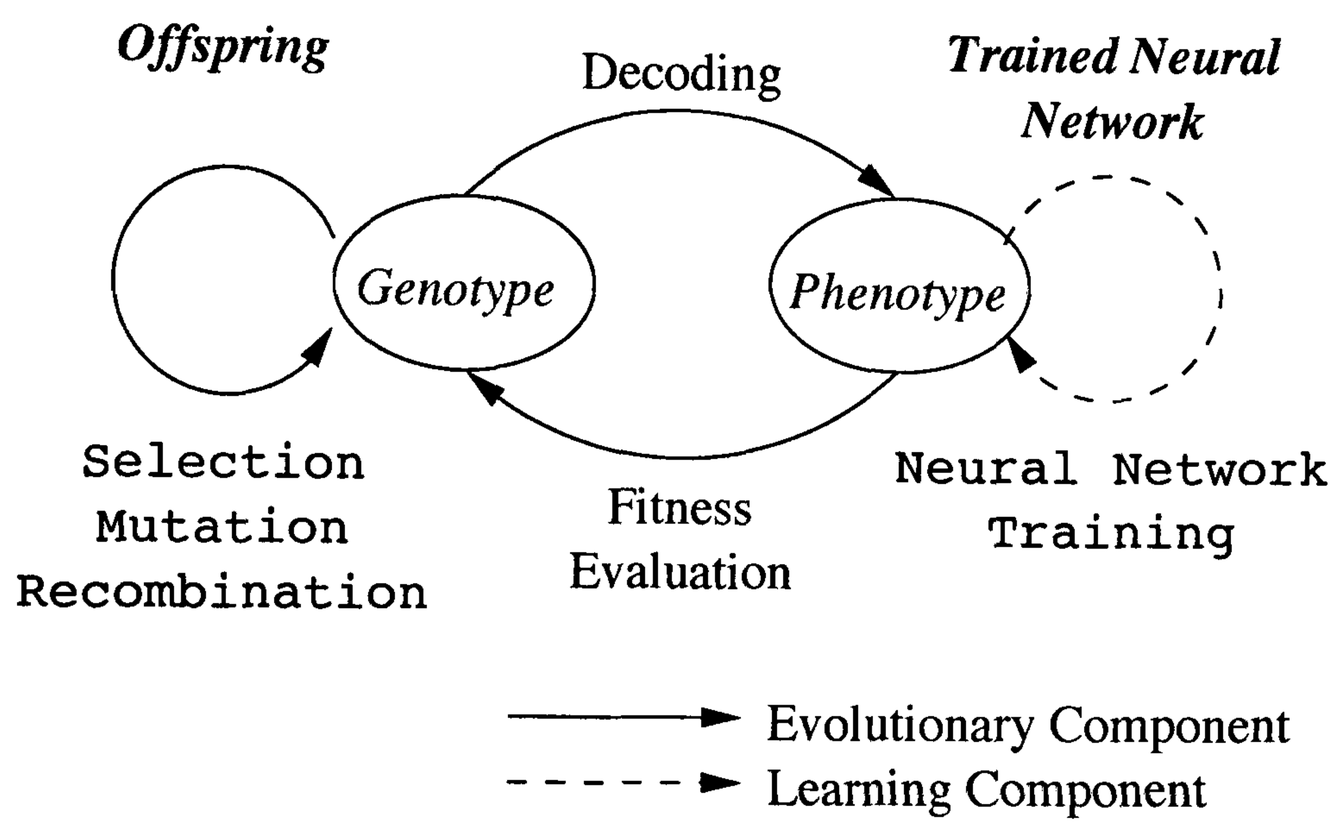


Figure 5.2: Designing neural networks by an evolutionary approach

In most evolutionary approaches, there is an initial population of genotypes, produced randomly, termed generation 0. Each genotype encodes one solution for the problem. In this case each genotype would encode the architecture of a single neural network. Each genotype is then decoded in turn to produce its phenotype, in this case the actual neural network. The neural network is then trained and set to process the problem at hand in the traditional manner. After the neural network has processed the problem, it is given a score on how well it has performed (generally called the fitness evaluation). Those genotypes that encode a network that perform well on the problem (those with a high fitness score), are selected to go into the next generation of genotypes. The genotypes of the current generation have a number of genetic operators applied to them, usually mutation and recombination(Freeman, 1994).

Mutation takes part of the genotype representation and changes a small portion of it to a random value. This ensures a certain amount of novelty survives in the population throughout the generations of development.

Recombination takes two genotypes, picks a point in each of the genotypes representation, splits the genotypes at that point and swaps these parts about. The two child genotypes are placed in the new generation and the two parent genotypes are deleted. Thus the two parent genotypes get mixed together.

This process of decoding, evaluating and applying genetic operators is repeated for each generation of genotypes. The outcome of applying these operators over a number of generations is that the fitness of the genotypes as a population should increase. This will be reflected in the decoded phenotypes, in that the neural networks will be better at solving the given problem.

Two specific evolutionary approaches are Genetic Programming (Koza, 1993, Koza, 1994) and Genetic Algorithms (Goldberg, 1989).

Initial use of evolutionary approaches has been to set the values of weights between nodes in the neural network (Koza, 1994). The starting structure (number of nodes, the type of nodes, etc.) is still pre-determined by the designer. Using an evolutionary approach, a set of weights is found that will solve the problem. In this sense, the evolutionary algorithm replaces the traditional neural network learning rule. This approach uses a from of genotype representation termed direct encoding, in that the transformation of genotype to phenotype is usually trivial and computationally inexpensive.

Another approach (Yaeger, 1996) has been to use the evolutionary approach to determine the general structure of the network and then to use a traditional learning rule to set the weights of the network. In this approach it has been used to set the structure of the network in terms of how many groups of networks are to be connected together and how many nodes are to appear within each group. But the nodes within the network are homogenous throughout all the networks and the scale of the networks was small, typically less than 500 nodes in each network group, with typically less than 10 groups. This approach uses a form of genotype representation termed indirect encoding. Here the task of decoding a genotype to a phenotype is more complex and computationally intensive but it has the advantage of being a more flexible system than direct encoding and captures some of the aspects of modularity found in natural systems.

Other work in this area has been performed by Beer, with his work on the modelling of insect nervous systems, Gruau (1992) in his work with cellular networks, Kodjabachian and Meyer (02/04/97) in an extension of Gruau's work, the work by Boers and Kuiper (1992/1996) on evolving L-system grammars to design networks, Vaarios work (Vaario & Ohsuga, 1992, 1994; Vaario & Shimohara, 1995; Vaario, Ogata, & Shimohara, 1996; Vaario, Onitsuka, & Shimohara, 1997) is a variation and extension of the work by Boers & Kuiper, and the work by Cangelosi, Parisi, and Nolfi (18/8/97) on axonal growth patterns. Dellaert and Beer have produced an evolutionary system that not only takes into account the overall structure of the network but also deals to a limited extent with the mixing of different types of neurons within a single network.

Most of the approaches so far only work at one or possibly two levels of complexity, e.g. determining the modularity of the network structure or developmental nature of the network. None of the approaches tackle all the issues of complexity together in one model, especially in the area of determining the node transfer function. It is in these areas that this research will concentrate.

In the Sections 5.3 and 5.4 the techniques discussed used a single network to explore the problem space. Another class of techniques, evolutionary optimisation algorithms, uses many different networks to explore the problem space. The most commonly used evolutionary algorithm to design ANNs is the Genetic Algorithm (GA). GA's will be briefly described but the details of GA's and their application to neural network design will be described in fuller detail in Chapter 6.

GA's solve a problem by sampling the problem space over many points. This is achieved by having a *population* of representation of solutions scattered randomly throughout the problem space, where each individual in the population represents one possible solution. The representation is typically a one dimensional string of characters or numbers. Each individual is evaluated with respect to their ability to solve the problem and this is termed the *fitness* of the individual. Individuals are picked to survive to the next generation of the process based on their fitness, those with a better fitness value are more likely to be chosen. This forms the basis of competition or survival of the fittest as proposed in

the theory of evolution. In order for the problem space to be explored two operators can be applied to the chosen individuals, crossover and mutation. Crossover takes two individuals and swaps chunks of their representations. Mutation takes an individual and changes a small part of the representation to another value. These two operators are sufficient to explore the problem space. Once crossover and mutation have taken place, the individuals are reevaluated to give their new fitness values and the cycle is repeated. Once the fitness of a network in the population reaches the user defined criteria, this iterative process ceases, with the best representation in the population representing the best solution found.

In the simplest case each individual in the population represents the connection weights of a single network. When an individual is evaluated, the numbers are read from the representation and assigned to a connection between two nodes. The network is then evaluated on a set of test data, with the fitness values being typically the number of correctly identified patterns or the Mean Square Error of the network. After a number of generations the networks represented in the population will produce better solutions than previous generations.

The way in which a genome will represent a neural network depends largely upon what sorts of architectures the neural networks will use.

Belew, McInerney, and Schraudolph (1991) states that there are many ways of representing a neural architecture with a binary string and that it is only limited by the imagination. Todd (1988) suggests one way to look at the problem is to consider a dimension called "developmental specification", which specifies how complete and literal a representation of the network is encoded on the GA string. At one extreme there is a very literal translation of string to network where every weight and connection is specified. At the other the genetic description can be used as input into a complex developmental interpretation process that then constructs the network.

Taking the extreme of encoding every weight of the network into the string : how do you encode the real valued weight into the string and how is the order of connections encoded?

A real number could be encoded directly into the string, i.e. have a s string of real values, not binary digits. Or it could be that a real number, within the range [m,M], is coded into a substring of B bits, giving 2^B intervals. The substring is then Gray-coded to minimise the Hamming distance between indices close value.

Cangelosi et al. (18/8/97) states that what is inherited in real 'organisms' is a genotype which specifies a set of instructions for generating phenotype and that the mapping from genotype to phenotype is a complex and nonlinear process which is affected by developmental processes. An important consequence of this is that the fitness of the organism is determined by the phenotype and not by the genotype, but the genotype is the part affected by the evolutionary process. It is this separation of levels that produces many significant properties and intricacies.

Proximity of alleles is an important issue in the representation of a string. If the two alleles are far apart on the string it becomes less likely that the GA will be able to discover and exploit nonlinear interactions. Merrill and R. (1988) has produced empirical evidence of this phenomena concerning weights in a neural network but which can be countered by using 'punctuated crossover' as developed by s Schaffer and Morishima (1985).

There is a basic problem with representing neural networks with a GA string in that two different networks with the same architecture but with different weight structures can provide two solutions to a problem. If crossover occurs between these two networks, the child may be less fit than either of the parents. The problem is in identifying which parts of the two networks perform similar functions and crossing those portions of the networks. But establishing a correspondence among even two three-layer networks can work out to be computationally intractable. The genetic representations of these varying networks cannot be expected to share the same schemata that the GA needs in order to be effective. A possible get-around is to make the correspondence between the genotype and the phenotype must less direct.

One scheme used by (Miller, Todd, & Hegde, 1993) is to use a 'wiring scheme' with a back-propagation algorithm. The wiring scheme specifies which nodes are connected to

which other nodes but not the weight of the link. The MSE of the BP algorithm is used as the fitness measure. Some points on this scheme are :

- In most learning algorithms a weight of value zero can be used to indicate no connection between nodes. Thus an existing connection can learn to have a value of zero to simulate no linkage but an absent connection (as in a wiring scheme) cannot ever become non-zero.

- The wiring scheme can be widened to allow for more variatio,. e.g. ternary specifications might specify a link was absent, restricted to positive weights or restricted to negative weights.

- By moving from away from finding weights only to finding weights and architecture, the fitness function must change accordingly, e.g. find a parsimonious network

Cangelosi et al. (18/8/97) suggest that a recursive mapping can cause the development of complicated neural structures from very simple genetic instructions (very useful for scalability in networks). This property may allow the search space that the evolutionary process will explore to be increased without a correspondingly large increase in the phenotype complexity. Changes in the context of the cell may affect how the genotype is expressed. Due to the recursive nature of the expression, the same set of instructions may give rise to differing neural structures in the phenotype. If a genotype for a functional group were mutated in a way that an instruction to say when to stop dividing was changed this could change the number of functional groups produced.

Cangelosi et al. (18/8/97) developed a framework for developing neural networks based on the principles of a non-literal mapping and ontogenetic development. The genotype to phenotype mapping takes place in a 2 dimensional space, and not purely in an abstract topological way. Neurons are assigned a physical location in the 2d space, in which axons 'grow' from the neuron. If the growing axon reaches another neuron a connection is formed between them. They also suggest that a framework that describes a neural architecture should include neuron cell division, as in real organisms, and cell migration

(cells moving location to location during the development process). Their framework consists of an originating (egg) cell containing the following information :

1. the cell type (there are 16 cell types in all)
2. Neuron growth process parameters :
    - angle of branching of the neuron's axon
    - the length of the branching segment of the neuron's axon
    - the point of neuron's surface from which the axon is to grow
    - the threshold of the neuron
    - the weight of the departing axon
3. a rule describing cell reproduction, which includes information for each daughter cell on :
    - the daughter cell type
    - the location of the daughter cell
    - changes to be made to the daughter cell parameters inherited from the parent cell

The 2d space in which the neurons grow is divided into three horizontal bands, neurons which end up in the lower band become input neurons, neurons that end up in the upper band become output neurons and neurons in the middle layer become hidden units. Additional input neurons are formed in the shape of 'motivational inputs', these indicate the 'internal' state of a network/organism.

Each cell in a network, during the development process, can divide 5 times, with each division making up to two daughter cells. This means that there can be a maximum of 32 neurons per network.

A GA is applied to the an initial population of 100 genomes and the fitness of the resulting determined in a simulated environment. Two simulation runs were undertaken, one set

involved no evolution of the axon branching and growing, these parameters remained fixed through out all generations, and only allowed cell division and branching. The other run allowed axon growth and branching. It was found that the results between the two runs were very similar. Networks were developed to solve the task. Possible consequences of having a complex genotype to phenotype mapping:

- The evolution of fitness of the networks showed a pattern of 'punctuated equilibria'. Many changes are likely to accumulate at lower levels without being expressed at higher levels which results in periods of stasis. A further small change at the lower level may interact with the other accumulated changes to produce a large jump up in fitness.

- The effect of a single mutation on the phenotype can be much greater if the mapping from genotype to phenotype is complex than if it is a simple one-to-one mapping (can lead to large differences beteween the average population fitness and the best individual's fitness)

Juedes and Balakrishnan (1998) state that mathematically neural networks perform an input to output mapping function (problem space to solution space). This mapping function is generated by the adjusting the weights in the network which is in turn achieved usually using a learning algorithm. Learning algorithms can either be supervised or unsupervised. Supervised learning involves using a training set of data from which the network can generalise. Backprapogation is an example of a supervised training algorithm. They suggest the mapping of input patterns to output patterns is limited by two factors :

- the *function* computed by each of the units in the network
- the *connectivity pattern* of the units in the network.

They also suggest that current training algorithms typically only work with homogenous networks that have nodes of all one type (e.g. sigmoid, threshold) and that the general architecture of the networks are determined throgh largely bruteforce methods coupled

with experience and rules of thumb. They further state that neural network design methods are grossly inadequate for designing compact and efficient networks. Other researchers have argued that that there is considerable reason to believe that higher order networks (e.g. using Product units, Leerink, Giles, Horne, & Jabri, 1995) may help realise this goal. Uhr (1994) further states in order to build compact and efficient networks sub-networks and specialised circuits need to be used.

5.6 Macro Level Design of Neural Networks

Two different aspects of modularity can be considered in both BNNs and ANNs; functional modularity and physical modularity. Functional modularity is where a task can be broken down into a number of sub-tasks and each sub-task is processed independently with the results being combined at a higher level of organisation. In functional modularity there is no regard to the physical nature of the networks involved. Physical modularity is where physically distinct regions of the network process different sub-tasks and the relationship between the modules can be mapped on a physical dimension.

Evidence that physical modularity occur in BNNs mainly comes from three bodies of evidence; studying the electrical activity of the brain using MRI/PET scanning and EEG recordings, physical dissection and identification, and localised brain-damage and aphasias.

Magnetic Resonance Imaging (MRI) and Positron Emission Tomography (PET) are used to map the electrical activity in the brain in an indirect manner (Posner, Peterson, Fox, & Raichle, 1998). In both techniques, the uptake of oxygen into a neuron from the blood stream is recorded and both rely on the assumption that a high metabolic rate, indicated by a high uptake of oxygen, in a neuron indicates a high level of electrical activity. To produce a map of the physical modularity of the brain, a large number of people are scanned whilst performing specific psychological tasks, such as a visual tracking task. MRI and PET scanning are relatively slow techniques as recordings can take several seconds and to improve on the accuracy, these can be combined with EEG recordings (Snyder, Abdullaev, Posner, & Raichle, 1995).

Localised brain damage can be used to locate which areas of the brain are possibly used for the processing of certain functions (Gazzaniga, 1989). When damage occurs locally to an area of the brain, through for instance head traumas or strokes, small portions of the brain are killed. By looking at the resultant loss of function, the areas damaged can be associated with that function. For instance, a stroke that causes localised damage in area known as Broca's area in the left hemisphere can be affect language and speech, and therefore it can be assumed that some aspects of speech are processed in Broca's area.

Using the techniques described multiple parallel processing pathways have been identified (Livingstone & Hubel, 1988; Kosslyn, Flynn, Amsterdam, & Wang, 1990; Van Essen, Anderson, & Felleman, 1992; Zeki & Shipp, 1988). On a gross anatomical scale the brain is split into two hemispheres, with some task being performed in both hemispheres but some tasks are primarily performed in a single hemisphere. A classic example is that of writing and speech which occur in the left hemisphere. The visual system in humans is probably one of the best understood regions of the brain. Within it multiple processing modules have been identified, concerned with visual aspects such as form/shape, motion, colour and position. These separate processing streams are integrated in higher cognitive structures to produce the unitary image that is consciously perceived. Possibly individual functions can be further subdivided into different subtasks and these more specialised functions can be localised into anatomically separate regions. Though it would be naive to take this viewpoint to its logical extreme and say that each and every part of the brain is dedicated to a single unique activity. It has been shown that different regions of the brain can be "re-trained" to cope with the loss of other areas of the brain through damage and demonstrates that a modular network can indeed produce fault tolerant systems (Zeki & Shipp, 1988).

It is thought that BNNs employ a modular architecture structure for a number of reasons. It is thought that the human genome cannot hold all the information necessary for a "blueprint" of the brain on a one to one correspondence of component to gene. It is therefore been suggested that a recipe is required whereby the recipe details a number of structures and sub structures that can be put together in variety of ways, ie a sort of LEGO building block system.

A viewpoint that has been put forward that individual neurons are not by themselves the basic processing unit but instead small groups of neurons, several hundred, known as minicolumns are in actual fact the basic processing unit (Eccles, 1981).

The wiring structure of the modules tend to lead to dense within module connections and sparsely interconnected modules.

In biological systems it is evident that there must be an correlation between the physical modularity of the networks and the functional modularity.

It has been shown that if all neurons were fully connected as all nodes are connected in an ANN, the brain would be many times larger than it actually is (Murre, 1993; Nelson & Bower, 1990).

An idea that has surfaced is that not only many neurons compete with one another but also modules may compete to perform functions (Edelman, 1981). This is the model proposed by Gerald Edelman under the name of "Neural Darwinism".

Three aspects must be considered in relationship to modularity; (i) the reasons for the task decomposition (ii) the method for accomplishing the decomposition, and (iii) the relationship between the modules (Sharkey, 1997).

Modular neural networks can be used to investigate the BNN's themselves in a form of "reverse-engineering"; by building networks that look like their biological equivalents it is hoped that information about the biology can be gleaned.

Many connectionist approaches use fully connected neural networks, e.g. Hopfield networks whilst others impose some structure in the form of layers (input, hidden, output layers) as in those used in feedforward networks where each layer is fully connected to one previous and after it. The advantage of these systems is that they are extremely flexible and elastic, to the extent that such systems, given enough resources in terms of hidden nodes can approximate any function. The disadvantage of such a scheme is that it could potentially take a very long for any learning scheme to find the correct weights for the input-output mapping to be learned. By using a modular approach it is possible to break down the task into smaller chunks with each chunk being easier to learn.

A possible advantage of modularisation is the minimisation of mutual interference between simultaneous processing and execution of different tasks. Modular networks have been shown to enhance a networks generalisation, scalability and learning speed.

Happel and Murre (1994) argue that problems arise from using fully connected networks in that the possible probability distribution of possible network configurations is extremely high. They suggest that the reduction of the number of possible configurations is affected by the network dynamics and the network architecture and that by the number of configurations can be constrained through the use of modular networks.

Anand, Mehrotra, Mohan, and Ranka (1995) designed a modular network scheme and tested it on a number of test problems including a set of problems that are used as benchmarks. They concluded that learning was improved due to: it was easier to learn a set of simple functions separately than to learn a complex function which is a combination of the simple functions. Conflicting signals from output nodes was reduced as the learning algorithm only modified those weights directly associated with that output node of a module whilst generalisation of the problem was increased. Their learning scheme could independently learn in parallel each of the subtasks for each module. Examination of the network to see which parts performed which task was easier because a structure was evident and easily identifiable. The speed of learning was over a magnitude faster than with a standard non-modular network trained with back-propagation.

Nolfi (1997) tested several different architectures, including simple feed-forward networks and modular networks, on task involving a robot finding objects in an arena and placing them outside of the arena. A genetic algorithm was used to optimise the network architectures. It was found that modular networks consistently outperformed the non-modular architectures and that the solutions were more robust (when tasks were transferred from the robot simulation to the real world robot). But they also found that it was not always easy to identify what modules within a network performed what subtasks. They found no clear cut correspondence between one module, or combination of modules, and the behaviours required in different environmental situations. Calabretta, Wagner, Nolfi, and Parisi (1997) found in a robot navigation task, that typically a geno-

type that gave rise to a modular type network outperformed equivalent type non-modular network genotypes.

Gruau (Gruau & Whitley, 1993; Gruau, 1992) defines a language for designing neural networks called *cellular encoding*. Gruau defines a number of symbols that can be used to construct the networks. These symbols are used to construct essentially a program that defines the network with the program held in a tree like structure. The symbols include symbols for sequential and parallel cell division, symbols for modifying connections strengths, symbols for pruning connections, symbols for modifying the threshold of a node, a symbol for repeating a section of the construction program. Gruau claims that the resulting scheme has the following properties:

- completeness - any neural network can be encoded using cellular encoding
- compactness - topologically and functionally more compact than other representations. The use of short codes means the shorter the codes, the smaller the search space, and the more efficient
- closure - cellular encoding always develops meaningful architectures
- modularity - if an neural network can is composed of sub nets, the code reflects this by concatenating the codes of the subnets and a code describing how these are connected - using building blocks - makes the final architecture easier to understand
- scalability - a fixed size code encodes a family of neural networks, if the neural network needs to increase in size to solve a bigger problem of the same family then the same code can be used but with an appropriate size parameter
- power of expression - cellular encoding's power of expression is greater than that of Turing machines or cellular automata - cellular encoding can be seen as an neural network machine language
- abstraction - is possible to compile a cellular automata program into the code of a neural network that simulates the computation of the program - reduces the search

space by allowing the GA to look at transition rules for rewriting a graph describing a net rather than specifying the architecture of the network directly.

Not only could modularisation be used for the advantages mentioned but it may have been necessary for the human brain to evolve as it has. It is easier to modify/duplicate self-contained regions of the brain rather than having to deal with a large homogenous whole.

Different modular schemes have been proposed to develop modular networks. Happel and Murre (1994) use a genetic algorithm to optimise the structure a modular network made up of CALM (categorizing and learning module) modules. A CALM can consist of a four different types of nodes which operate in different ways; representation nodes (R-nodes), veto nodes (V-nodes), arousal nodes (A-nodes), and external nodes (E-nodes). Categorisation takes place through the resolution of an interactive winner-takes-all competition in the R-nodes activated ay the input. The resolution of the competition is mediated through the V-nodes and A-nodes The inputs to a module are either the inputs to the network or the outputs from the outputs of the R-nodes in other modules. From testing it was found that the networks were better able to generalise to instances never before presented to the network. There also demonstrated the ability to learn patterns in a sequential manner without much interference with existing patterns.

5.7 Learning Strategies in Neural Networks

Ackley & Littman, 1992 suggest a scheme called Evolutionary Reinforcement Learning (ERL) to improve learning in an individual neural network. The ERL algorithm allows not only inherited behaviours to be specified but also inherited goals that are used to guide learning. A population of genomes is prepared, with each genome representing one individual. Each individual consists of two neural networks, an *action* network and an *evaluation* network. The action network maps sensory input to a behaviour, and its *initial* weights are specified by the genome. The weights of the action network change over the lifetime of the individual in accordance with learned behaviour. The evaluation network

maps sensory input to a scalar value representing "goodness" of the current situation. By learning to move from "bad" situations to "better" situations — modifying the action network weights in the process — an individual achieves the goals of learning passed down from previous generations. A reinforcement signal, based on the scalar value, is used to change the weights in the action network. The weights of the evaluation network do not change in the lifetime of the individual.

The ERL algorithm works in the following manner :

1. Present the input info to the input layer of the action network and propagate through to produce a behaviour
2. Present the input info to the input layer of the evaluation network and produce a reinforcement signal to change the weights of the action network
3. Perform the behaviour

In a comparative study between evolutionary learning coupled with reinforcement learning and pure evolutionary learning, it was found that the evolutionary/reinforcement learning performed significantly better and took fewer generations to produce. This outcome suggests that it is easier to specify goals than implementations, or it is easier to generate a godd evaluation function than a good action function.

With further analysis of the following effects were found to be present :

- Baldwin Effect : As generations pass, ways of anticipating and avoiding the problem become incorporated and instinctive
- A successfully inherited ability to avoid a problem means that learning to avoid the problem confers little advantage
- Shielding effect : Genetic information related to learning the ability is less constrained functionally; mutations can accumulate without affecting fitness. In this case a well adapted action network can shield a maladapted learning network from the fitness function.

- Goal Regression : if the inherited ability did arise via a Baldwin Effect, shielding will preferentially affect the original learning ability that the Baldwin Effect relied upon. A successfully inherited ability acquired via the Baldwin Effect will not shield all learning genes equally, it will preferentially affect the learning genes that were responsible for the Baldwin Effect to begin with.

5.8 Conclusion

Having created and developed an intellectual infrastructure for the subject, we now move on to the application of the network models to demonstrate the power of the infrastructure.

6

Network Models

6.1 Chapter Introduction

This chapter will discuss the issues surrounding the development of increasingly complex models. Each model will de developed tested and the drawbacks of each model will be highlighted. These drawbacks will be remedied in the model following it. This concept follows the hypothesis that there is no *a priori* level of reduction of model complexity for a particular problem. The designer needs to develop a deeper understanding about the critical parameters that govern the behaviour of the system being studied, which can only come from repeated simulation at different levels of resolution with various models of the system. Systematic increase of model complexity while keeping the essential computational capability of the system is an excellent example of this process (Segev, 1992).

6.2 Task to be solved

In order to determine if the network is using a modular approach to solve a problem which consists of a number of tasks, the inputs patterns must reflect this. Therefore each pattern presented to the developing network must consist of a number of sub patterns that can possibly be processed separately. The task must also be of a sufficient difficulty that it is

not easy for a network to solve the task. The task chosen therefore consists of three sub patterns in every input pattern:

- a pattern recognition task
- a relative position task
- a location task

The pattern recognition task requires the network to learn a selection of patterns which may appear in a variety of rotations (rotated 0^o, 90^o, 180^o and 270^o). Each pattern is presented to the network as a grid of seven-by-seven digits of zeros and ones. This is combined with the position task in which the letter in the recognition task can appear in one of nine positions in a three-by-three grid. The network will be required to indicate in which position the letter is placed. These two tasks are combined with the relative position task in which a 'bar' can appear above, below, to the left or to the right of the pattern. The network will be required to indicate in which of these realative positions the bar appears relative to the presented pattern.

As examples of the patterns that the network will be required to learn, some of the patterns are shown in Figure 6.1. Figure 6.1 (i) shows the overall input area divided into nine possible subsections where a pattern could appear. (ii)—(v) shows one of the patterns with a 0^o rotation with the four possible relative positions of the bar relative to the letter (below, right, above, left respectively). (vi)—(ix) shows another of the patterns with the relative position bar in the below position with a 0^o rotation, with a 90^o rotation, with a 180^o rotation, with a 270^o rotation. Each input pattern consists of 196 inputs with each output pattern consisting of 17 outputs (4 outputs indicating the pattern displayed, 4 outputs required to indicate the relative position of the bar to the letter and another 9 outputs to indicate the position of the letter in the three-by-three grid).

6.3 Evaluation

As previously stated the network will be evaluated to determine the following issues:

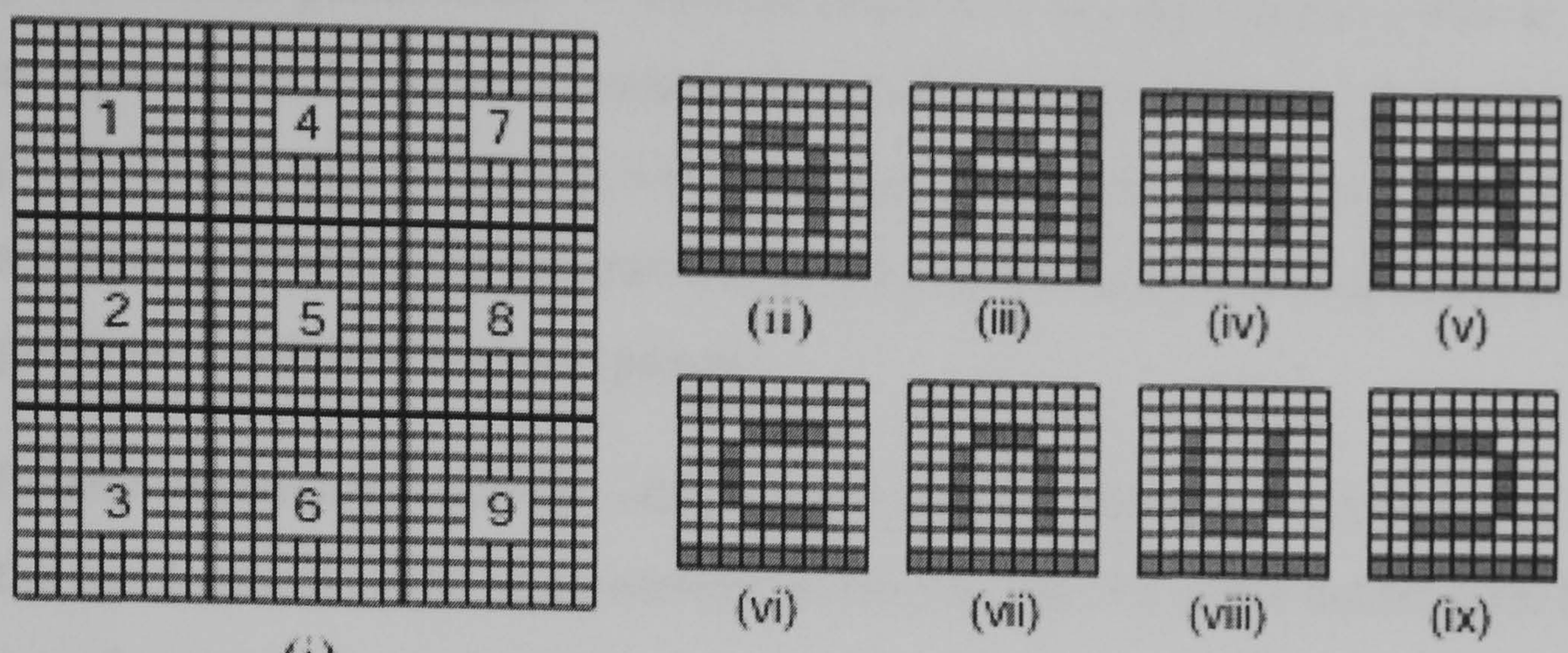


Figure 6.1: Examples of input patterns

- Does a modular network out-perform a non-modular network in training on a difficult problem (training performance)?
- Does a modular network out-perform a non-modular network in accuracy on a difficult problem (accuracy performance)?
- Does a modular network out-perform a non-modular network in generalisation on a difficult problem (generalisation performance)?
- If modularity does arise spontaneously in the produced networks, is there a relationship in terms of their spatial physicality where the functional modularity maps on to the physical modularity of the network (functional/physical mapping)?

The term non-modular network in this case refers to a simple multi-layer feed forward perceptron network trained with standard backpropagation.

To evaluate how well the network has performed the measures used will be:

- training performance — will be measured on the number of training cycles required to train the network so that the network error falls below a specified level or a maximum specified number of training cycles has been reached
- accuracy performance — will be measured by comparing the network error

- generalisation performance — selected portions of the input patterns will be used as input. Only the patterns (without the relative position bars) or only the relative position bars (with no pattern) will be presented, and a measurement of the network accuracy will be used to determine if the network can still accurately determine the presented portion of the input pattern

- functional/physical mapping —this measure must be described in conjunction with the development model of the network and in the case of a single hidden layer MLP neural network does not apply

6.4 Obtaining a Baseline

In order to make a comparison against statistics must be gathered for the case where the network is non-modular. As previously stated comparisons will be made against a simple multi-layer feed forward perceptron network trained with standard backpropagation.

The network contained 196 input nodes, 16 output nodes and 30 hidden nodes. The number of hidden nodes was set at 30 as this allowed a reasonable amount of scope for generating groups of nodes with a reasonable number of neurones in each group, whilst allowing the network to be trained in a reasonable time. By restricting the number of hidden nodes to 30, its also increased the difficulty of the task harder and so could possibly mean the network could be forced to be more efficient due to the limited resources. Increasing the number of nodes would necessarily increase the training times which when being performed in a GA could add up to a considerable increase for a single solution to be found by the GA. These restrictions are in place to limit the search space of the GA. To make the comparison meaningful across all experiments, the number of nodes must be kept constant between the different models. Figure 6.2 shows the average network errors (n=20 runs per group) for networks with 20, 30 and 40 hidden nodes.

The training performance, as specified earlier is measured by examining the number of training cycles required to train the network so that the network error falls below a spec-

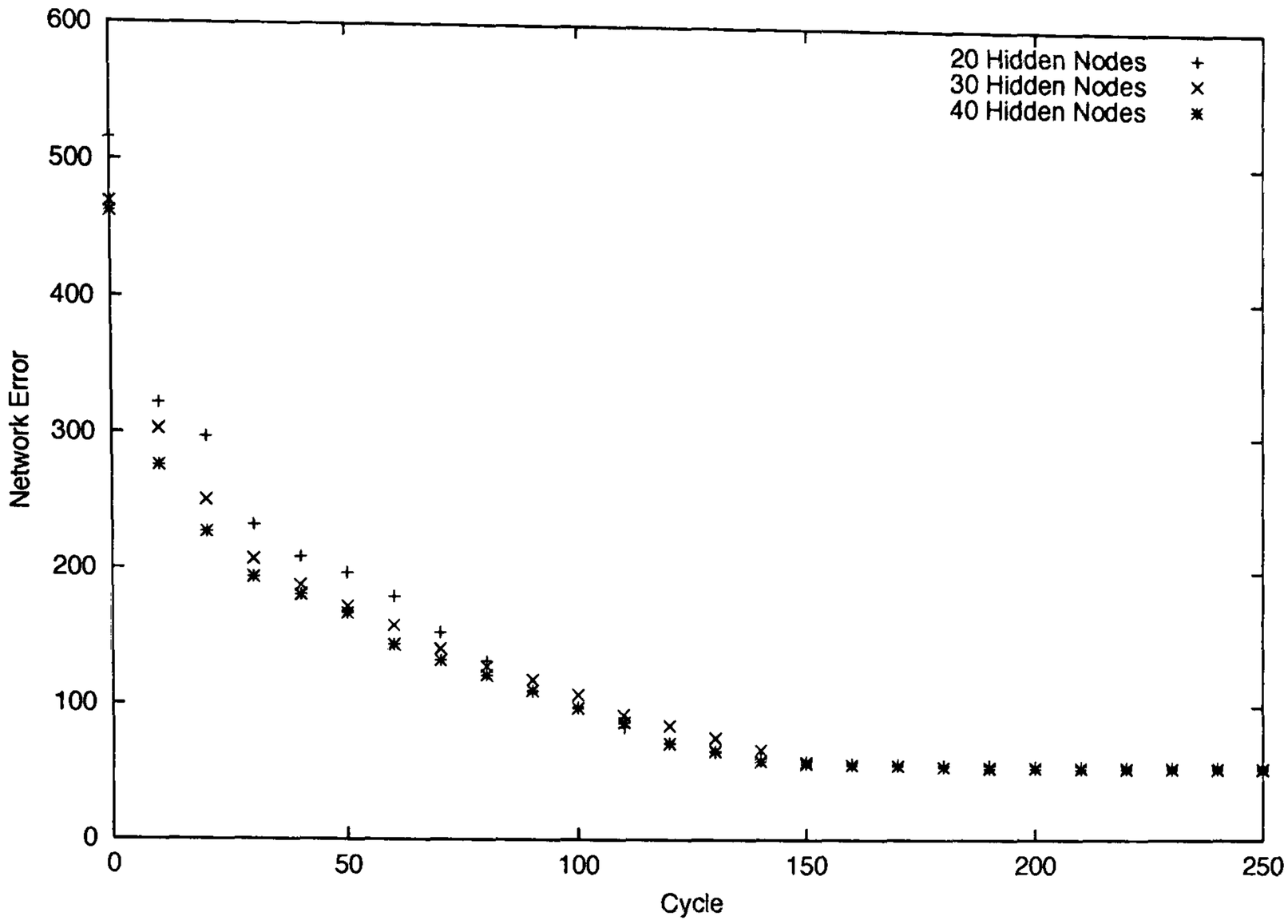


Figure 6.2: Average network error for the input patterns (n=20 per group)

ified level (where the network error falls below 1.0) or until the maximum set number of network training cycles has been reached.

6.5 Choosing the GA Parameters

One of the main problems with GAs, and EAs in general, is the lack of experimental evidence and guidance on what the parameter settings of the GA itself should be (number of phenotypes in the population, mutation probability, crossover probability, etc). Of course these parameters themselves could be optimised by a GA; thus there would be a GA operating within a GA!

Van Rooij et al. (1998) have performed a comprehensive study of the effects of manipulating the parameters of a genetic algorithm and their effects on the training of a neural network. Parameters that are most likely to influence the efficiency of the GA are:

- Crossover Probability - the probability that a genome in a population will be invloved in a crossover operation

- Mutation Probability - the probability that a gene in a genome will be mutated
- Genomes per generation - the number of genomes that a population contains
- Number of generations - the number of iterative cycles, consisting of selection, crossover and mutation, that are performed

From this study, the Table 6.1 shows the values chosen for the parameters that were likely to yield "good" results. These values are based on using a Simple GA with 1-point crossover and mutation.

Parameter	Value
Crossover Probability	0.9
Mutation Probability	0.1
Genomes per generation	50
Number of generations	500

Table 6.1: GA Parameters

The number of genomes per generation and the number of generations have been kept low considering the complexity of the task and thus may produce less fit genomes to keep the time taken to produce a solution to a reasonable time scale (a single solution should be produced in days rather than weeks)

6.6 Introducing a Physical Structure Representation - Model 1

Evolutionary techniques have been shown to be better at producing novel and efficient designs and at coping with local minima in the search space of the problem domain than other neural network design methods. This lends weight for the argument to use evolutionary techniques to design networks that mimic biologically networks. To investigate this further the aspect of the biological network issues, a network will be produced that

uses a simulacrum of a physical environment to determine what neurons are grouped together into potential modules and the connections between those groups of neurons. This simulacrum of a physical growing environment will be termed the "substrate".

When using a GA to design neural networks a simple technique is to define every connection between the nodes in the neural network. This approach is usually termed a "fully encoded" method. By doing so, the design of the network allows very little interaction with the network's environment, in this case the substrate. Therefore it has been decided to take an approach where the initial starting point and direction of the connection growth is optimised by the GA, but the ongoing development of the connection is guided by the substrate. This level of interaction between environment and network growth will allow more complex and novel solutions to be produced.

Each neural network is associated with a genome to control the location and sizes of the regions in the substrate.

Networks in natural systems grow from a small initial group of neurons in a biological substrate that supports and guides the developing neurons. These precursor neurons divide, migrate, orientate themselves in their environment, grow connections in predetermined directions and may eventually die off. The orientation, migration and growing of connections initially depend on the properties of the substrate of cells that the neurons grow on. Chemical attractants, generated by the substrate, are used to guide the growth of the connections between neurons, and the cells that produce the chemicals can be viewed as "landmarks". This leads to a neural network that has a complicated structure consisting of regions and layers differing in size and the amount of interconnectedness between these layers and regions.

The Substrate Genome, Figure 6.3, encodes a series of two dimensional coordinates. A discrete two dimensional grid, in this case representing the biological substrate, is divided into regions using a dissection based on a set of control points. Each control point represents the locus of a single Voronoi polygon. A Voronoi dissection was used as it easily allows the substrate to be divided into a number of separate and distinct regions using a relatively small amount of information, in this case, a number of control points. The

consequence of this is the genome encoding the information can also be relatively small. An example of a substrate divided into ten regions, using ten control points, is shown in Figure 6.4. The regions will be used to partially constrain the connections between groups of nodes. This will mimic the modular structure found in natural systems where only certain modules are connected to other modules, where groups of neurons are more likely to be connected to other groups of neurons that are physically closer together.

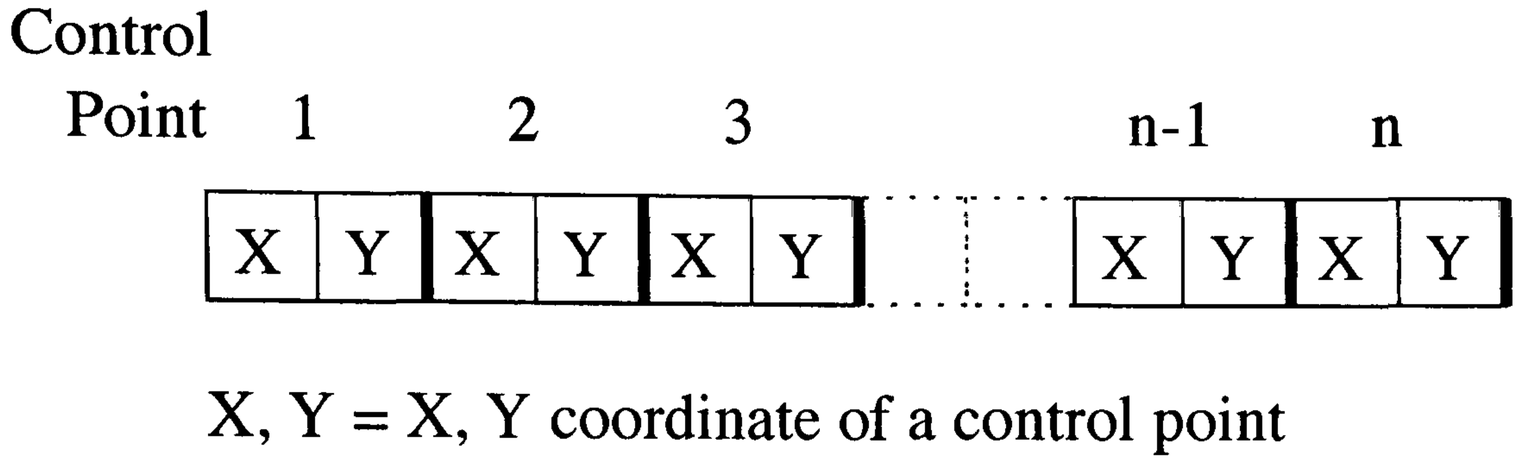


Figure 6.3: The encoding structure of the Substrate Genome

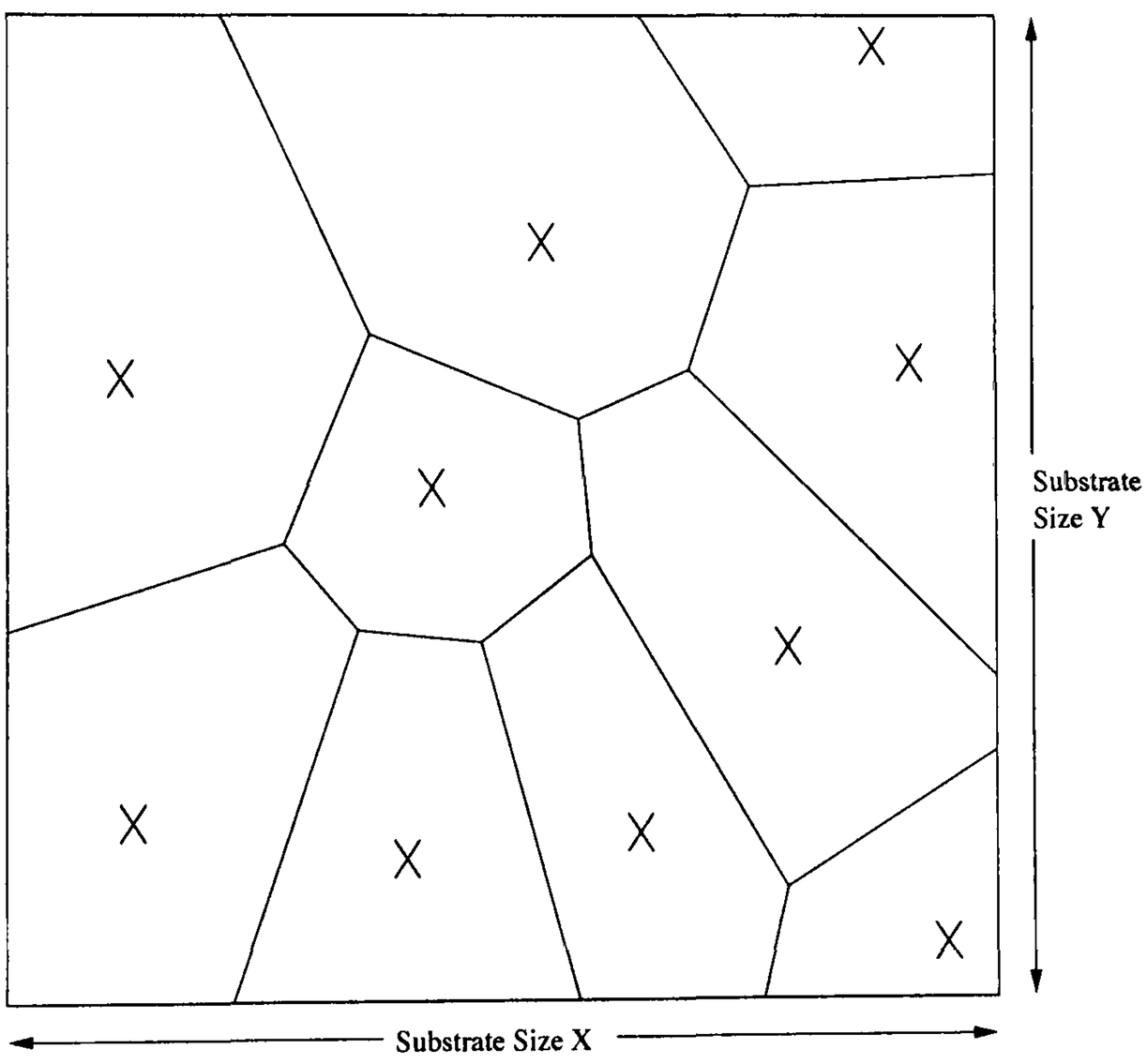


Figure 6.4: Substrate division using a Voronoi dissection

6.6.1 Constructing the Networks

To produce the neural networks that will solve the task, there are two major stages of development; the development of the genomes over time, using a GA, and the development of a genome into a neural network.

A genome is developed into a neural network by decoding the substrate chromosome first to give a substrate divided into different regions. The percentage coverage of each region in the substrate is calculated and this is used to determine the number of neurons assigned to each region. Thus if a particular region covered 20% of the substrate area then 20% of the total number of neurons used in the network would be assigned to that region/module. If the sides of the substrate are labelled 'North', 'South', 'East' and 'West' then the inputs of the neural network are always connected to those regions in the adjacent to the 'North' side of the substrate and the outputs are connected to the regions adjacent to the 'South' side of substrate.

The regions within the substrate are then connected. Using the control point for each region as a reference point, the regions directly to the 'South' of the control point and adjacent are connected to the current region.

Once the regions have been connected the neural network is trained using standard back-propagation.

A simple GA is used to develop the genotypes over time. An initial population of 100 genomes with random is produced, which is evaluated to produce fitness scores for each network. Selection, crossover and mutation are applied to the initial population to produce the next generation. Crossovers use a one-point crossover operator which crosses the appropriate chromosomes in the parents chromosomes, substrate chromosome with substrate chromosome, nodes chromosome with nodes chromosome and connections chromosome with connections chromosome. The mutation operator used is a simple flip-bit operator which is applied to all three chromosomes in a genotype. One thousand generations were used per run to develop the solutions.

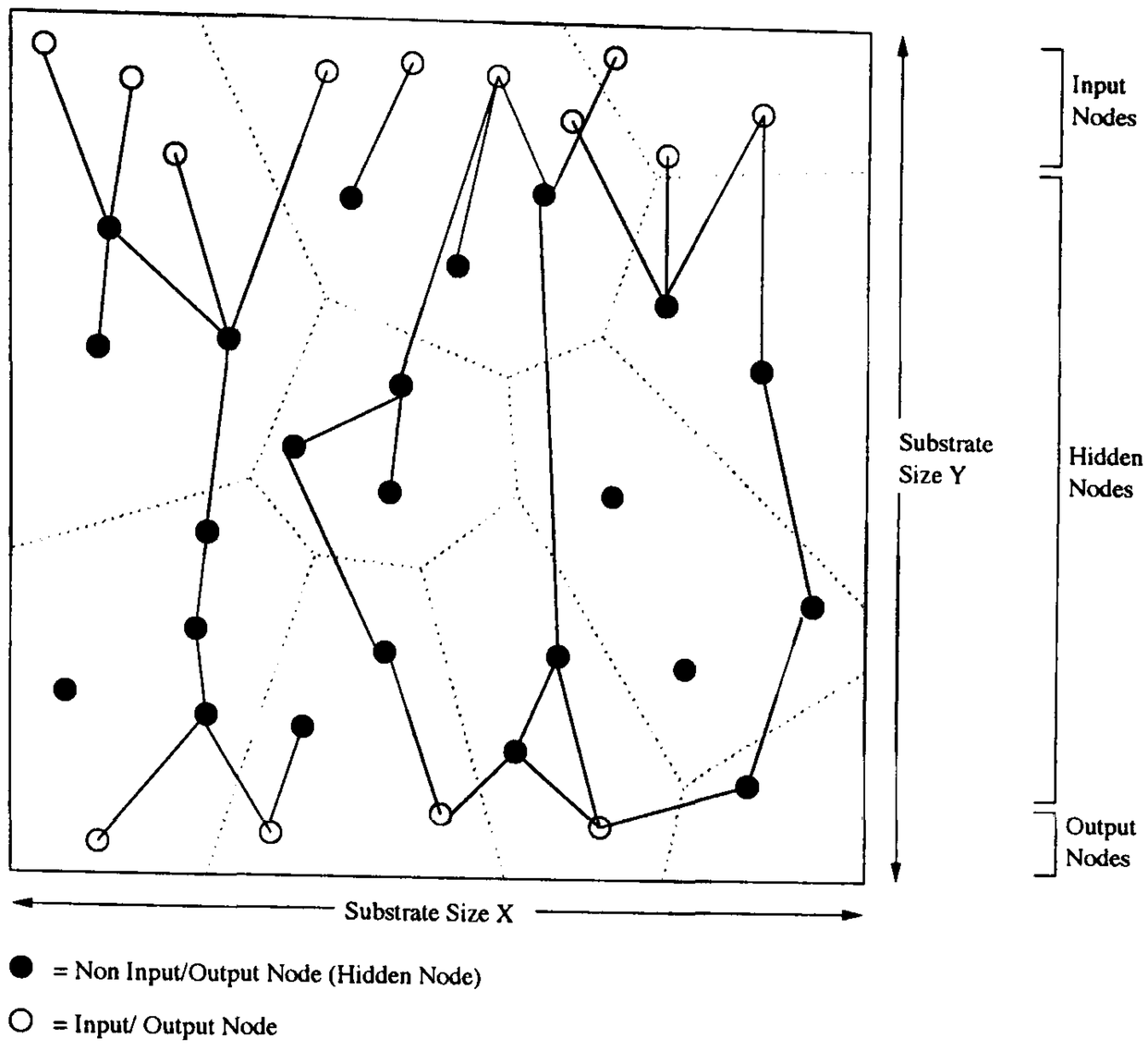


Figure 6.5: Connection Inputs and Outputs to the Developed Neural Network

6.6.2 Results

The development method will be used to solve a visual tracking problem, which is to track a target object in a simulated 2-dimensional scene. There is only one object in the scene, the target, which moves in a straight line until at random points in time a change in direction occurs. Only a small section of the visual scene can be viewed by the neural network at any one time and it is this "view" (visual aperture) that forms the input to the neural network. The neural network can control the position of the visual aperture in the scene by using two outputs, one output controlling the horizontal movement of the visual aperture, the other controlling the vertical movement of the visual aperture. The neural network has additional inputs indicating if the visual aperture has reached the limits of the visual scene in both the x and y axes.

The target is placed at a random location within the visual aperture. The image as seen in the visual aperture is given to the neural network as input and the neural network is allowed to process this data. The values of the output nodes are used to move the visual aperture by the appropriate amount. The fitness function is calculated by summing a score for how close the centre of the visual aperture is to the target for each frame.

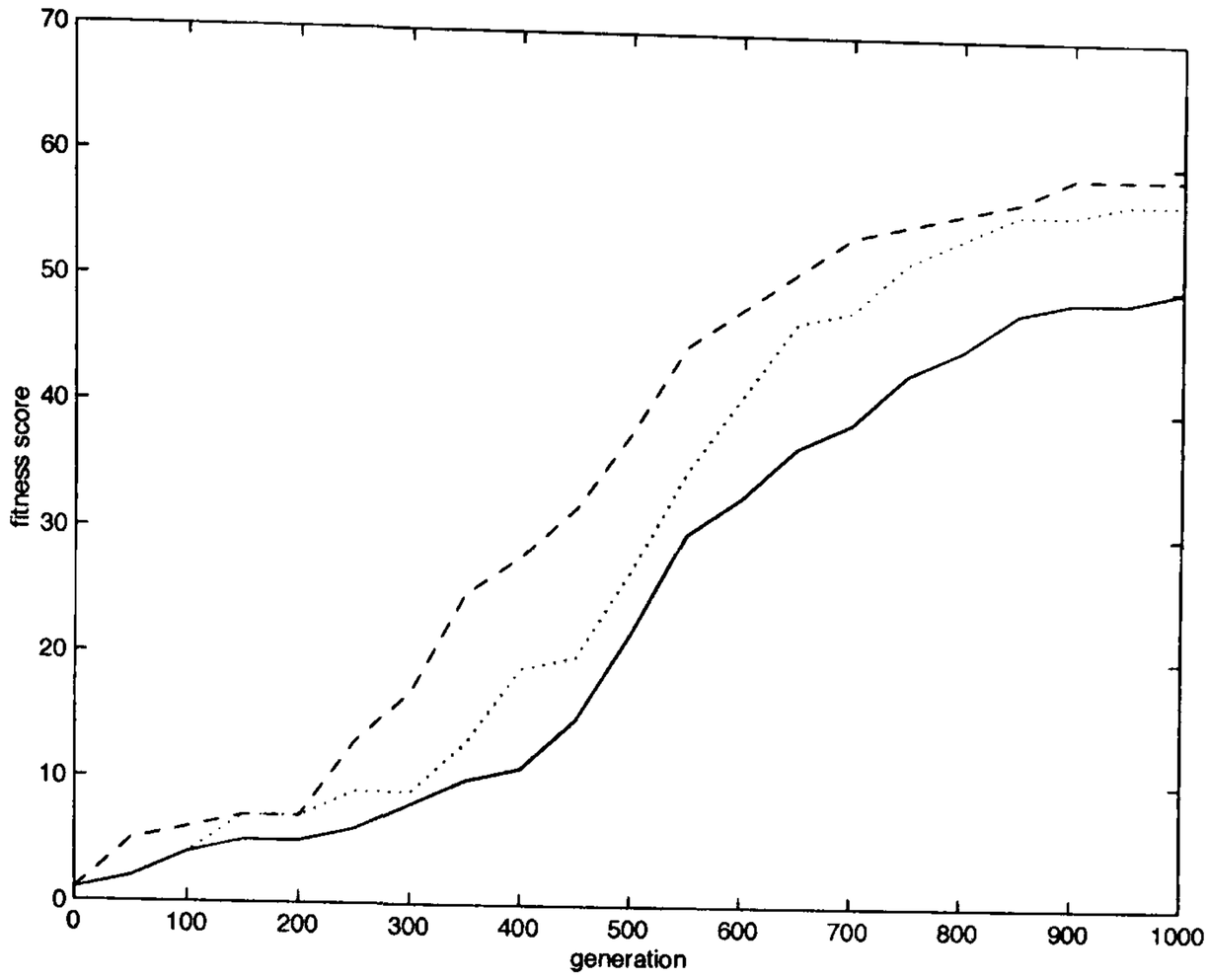


Figure 6.6: Results for visual tracking task: Solid line - One region scenario; Dotted line - Three region scenario; Dashed line - Five region scenario

The development process was tested on the visual tracking task in three different scenarios :

- the substrate contained one control point, i.e. effectively no subdivision of the substrate
- the substrate contained three control points
- the substrate contained five control points

The maximum score that can be achieved given the visual aperture size and the number of frames used in the testing is 60. Figure 6.6shows the average for each scenario with ten runs per scenario.

As can be seen from Figure 6.6 significantly better fitness scores were achieved overall in the scenarios (three & five control points)which allowed the defining of regions within the substrate than the scenario with no substrate regions (one control point). It can also be seen that the evolution of better solutions to the problem were generated more quickly when the definition of regions was allowed than when not allowed. The five control point

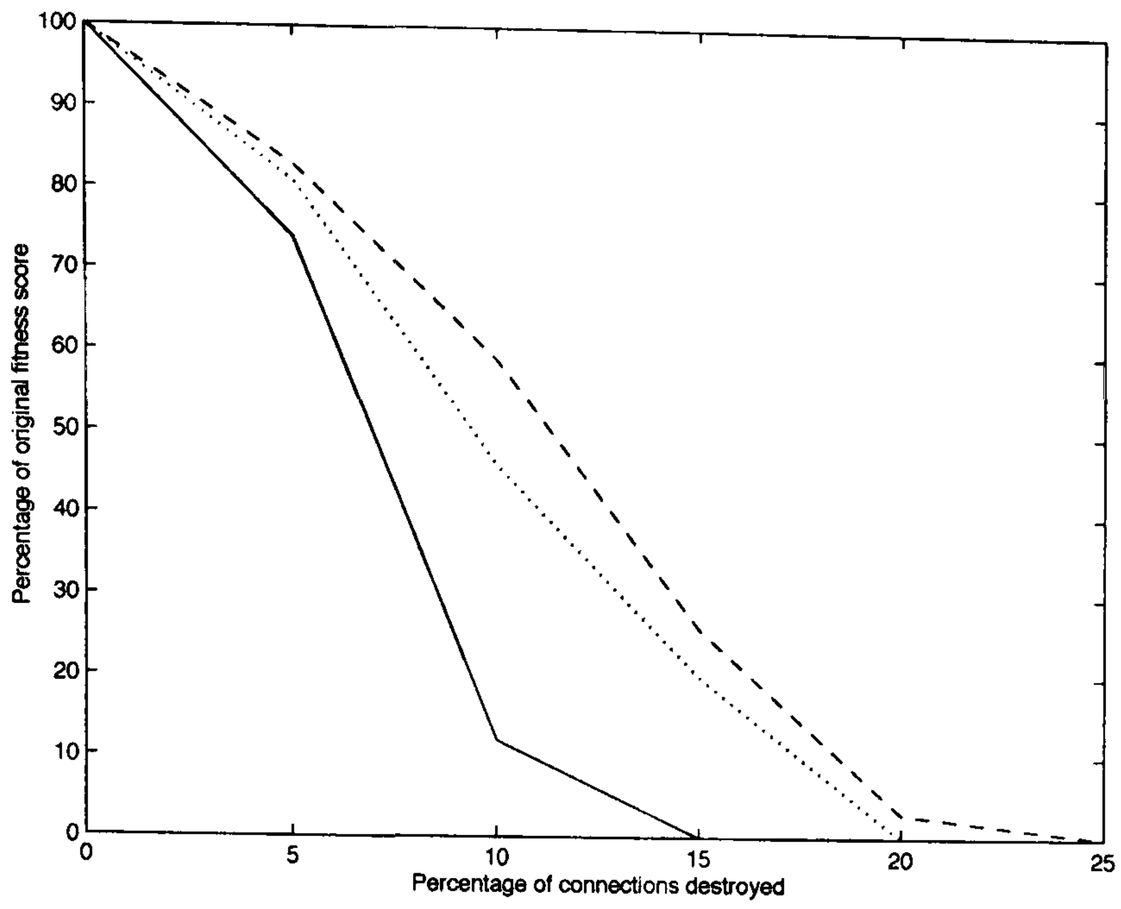


Figure 6.7: Results for network robustness: Solid line - One region scenario; Dotted line - Three region scenario; Dashed line - Five region scenario

scenario again seems to produce better solutions than the three control point scenario though only marginally.

The robustness of the networks for each scenario were tested by destroying connections between nodes. Each network was tested on the task with 5%, 10%, 15%, 20%, and 25% of its connections destroyed. The scores that each network obtained on the tracking task are plotted as percentages of the scores they originally obtained in their non-destroyed condition. As can be seen in Figure 6.7, again the networks that were divided into regions (three & five control points) showed gentler degradation in their scores than the networks with no regions (one control point).

6.6.3 Conclusions

The specific advantages of constructing networks with multiple regions are:

- the resultant networks are quicker to process - as the network is not fully connected, there is no need to process every connection
- the networks are quicker to design than their non-regional counterparts

- the resultant networks perform better on the task that they have been designed to solve.

- the resultant networks are more robust than their non-regional counterparts

The use of evolutionary techniques means that only the method for constructing the network and what constitutes a "good" network need be defined. No specific method for optimising the network parameters need be devised.

6.7 A Finer Grained Representation - Model 2

In the previous model groups of nodes were created and connected together in a manner that required no development. In nature neural networks are being created, the following points that occur during the creation of a network which. The previous model ignored this development of neurons and the fact that neurons are connected not only to the nodes which are closer to them physically but have more discrimination. Chemical markers and chemical gradients are used to guide the connections of neurons from one area to another area of neurons. The previous model although modelling the physical aspect of the growth in terms does not model this use of chemicals. Thus in the previous model all neurons in one are were connected to all neurons in adjacent areas; there is no fine grained discrimination. In some respects the previous model imposed regions or groups and thus there was no spontaneous creation of neurons into groups. By producing a model that models the creation of individual neurons and growth of connections the process that occurs in nature can be examined in more detail and an analysis can be made as to whether a network created in this manner will spontaneously create groups of neurons that act as a module to process a single aspect of a multi-process task.

Aho, Johnson, and Ullman (1977) has developed a model that simulates most of the aspects required for the wanted model but the results of the model do not concentrate on the aspect of producing modular neural networks. Thus this model will be expanded and the task used in the Section—6.6 will be used to examine this model to determine if a network

produced in this way produces a better network as given by the criteria in Section 6.3. The model produced by Aho et al. (1977) uses a technique based on the use of *Lindenmayer Systems* for determining the positioning of the growing neurons and the growth of connections form the neurons and so to understand the model Lindenmayer Systems will be discussed in detail in the following sections.

6.7.1 Details of Lindenmayer Rewriting System

A Lindenmayer system (*L-system*) is a string re-writing method that was originally conceived to investigate the mathematical development of plant structures (Prusinkiewicz & Lindenmayer, 1990). L-systems differ from many other grammar systems in that each letter in the string, termed a *word* in L-system parlance, is re-written in parallel, unlike Chomsky-like grammars that have their letters rewritten in sequence. The purpose behind the parallel rewriting scheme is intended to capture the process of cellular divisions in multicellular organisms, where many divisions can occur simultaneously. Context-free L-systems (also called *OL-systems*) can produce some languages that cannot be produced by context-free Chomsky grammars due to the nature of the parallel rewrite.

In the simplest type of L-system, *Deterministic Context-free L-systems* (DOL-Systems), each L-system consists of an alphabet of allowable letters, a number of re-write rules, *productions*, and a starting word consisting of one letter, an *axiom* (ω). The productions are used to re-write, in the first iteration, the axiom into a derived word, and for each subsequent iteration, the letters in the current word into a new word. Each production (ρ) consists of a *predecessor* and a *successor*, when a letter in the currently derived word matches the predecessor, that letter is replaced by the letters specified in the successor. To take a simple example consider an l-system that consists of two productions:

1. a rule that rewrites the predecessor letter 'a' in a word to the successor letters 'ab'

2. a rule that rewrites the predecessor letter 'b' in a word to the successor letter 'a'

In this example the axiom will consist of the letter 'b' and the derived word is shown for six rewriting iterations in Figure 6.8, along with . a more compact representation as shown in Figure 6.9

Step	L-system word
0	b
1	a
2	ab
3	aba
4	abaab
5	abaababa

Figure 6.8: Example of a DOL L-system derivation

$$\omega : b$$

$$\rho_1 : a \longrightarrow ab$$

$$\rho_2 : b \longrightarrow a$$

Figure 6.9: Example of a DOL L-system description

It is assumed that for any valid letter in the L-system there is at least one production to rewrite that letter. If no production is explicitly stated for that letter, then it is assumed that the letter is rewritten in the new word as itself. For a DOL-system it assumed that there is one and only one production rule for rewriting each letter in the alphabet being used in the l-system.

The basic design of the DOL-system can be extended by the introduction of *stochastic*, *context-sensitive*, *parametric* and *environmentally sensitive* L-systems. Each of these extensions can be combined to provide the varied levels of complexity, with a combination of all four L-system being the most complex. Figure 6.10 shows a table outlining the syntax that is commonly used to describe rules of the l-systems.

Type	Rule syntax
DOL	$\rho_n : predecessor \longrightarrow successor$
Stochastic	$\rho_n : predecessor \overset{probability}{\longrightarrow} successor$
context-sensitive	ρ_n : left context ¡ predecessor ¿ right- ontext$\longrightarrow$ successor
parametric	
environmentally sensitive	

Figure 6.10: Example of a DOL l-system derivation

6.7.2 Translating l-system words into visual models

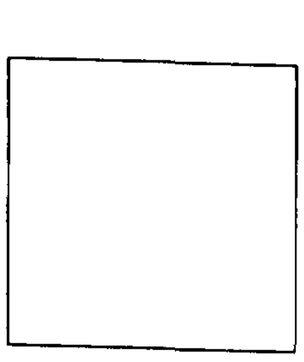
As stated previously, the original intention of L-systems was to investigate and produce 'life-like' models of plants and part of this investigation was to produce realistic images of plants in two and three dimensions. This is achieved by interpreting the derived words to control the production of an image. (Prusinkiewicz & Lindenmayer, 1990) developed a system whereby a derived word was interpreted as a series of instructions to a LOGO-like turtle. The turtle is positioned at an initial position (x,y) with an initial heading (α). A subset of L-system letters are used to instruct the turtle to move forward or rotate about one of its three axes (or two axes for two dimensional images).

$$\omega : F - F - F - F$$

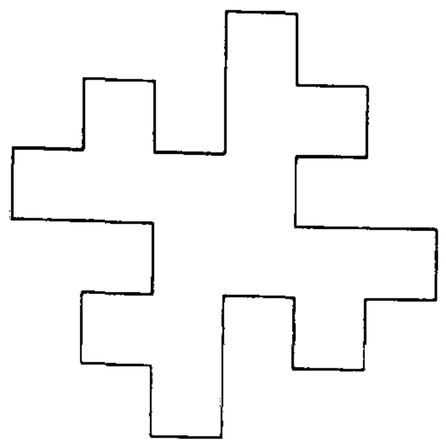
$$\rho_1 : F \longrightarrow F - F + F + FF - F - F + F$$

where	
F	Move forward distance d, (x,y,α) changes to (x',y',α)
	where $x' = x + d\cos\alpha$ and $y' = y + d\sin\alpha$.
	A line is drawn between (x,y) and (x',y').
f	Move forward distance d, (x,y,α) changes to (x',y',α)
	where $x' = x + d\cos\alpha$ and $y' = y + d\sin\alpha$.
	No line is drawn between (x,y) and (x',y').
+	Rotate turtle by $+90^o$
-	Rotate turtle by -90^o

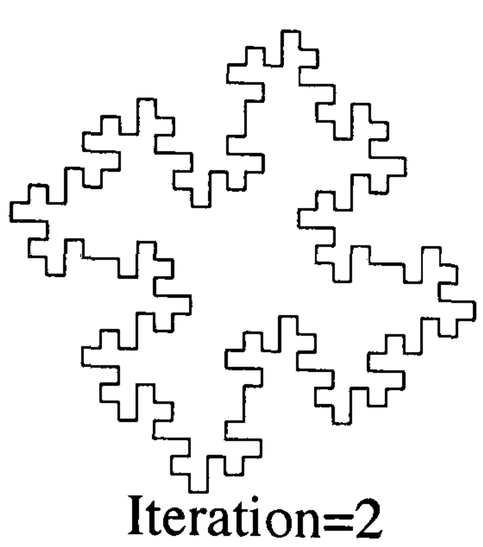
Figure 6.11: Example of a simple DOL L-system used for turtle interpretation



Iteration=0, F-F-F-F

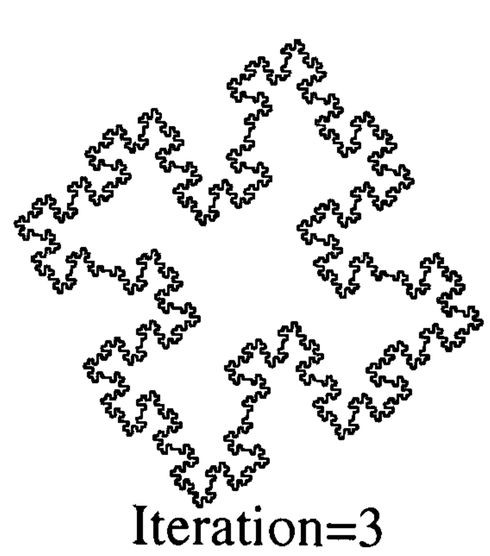


Iteration=1, F-F+F+FF-F-F+F



Iteration=2

F-F+F+FF-F-F+F-F-F+F+FF-F-F+F+...[88 modules]



Iteration=3

F-F+F+FF-F-F+F-F-F+F+FF-F-F+F+...[920 modules]

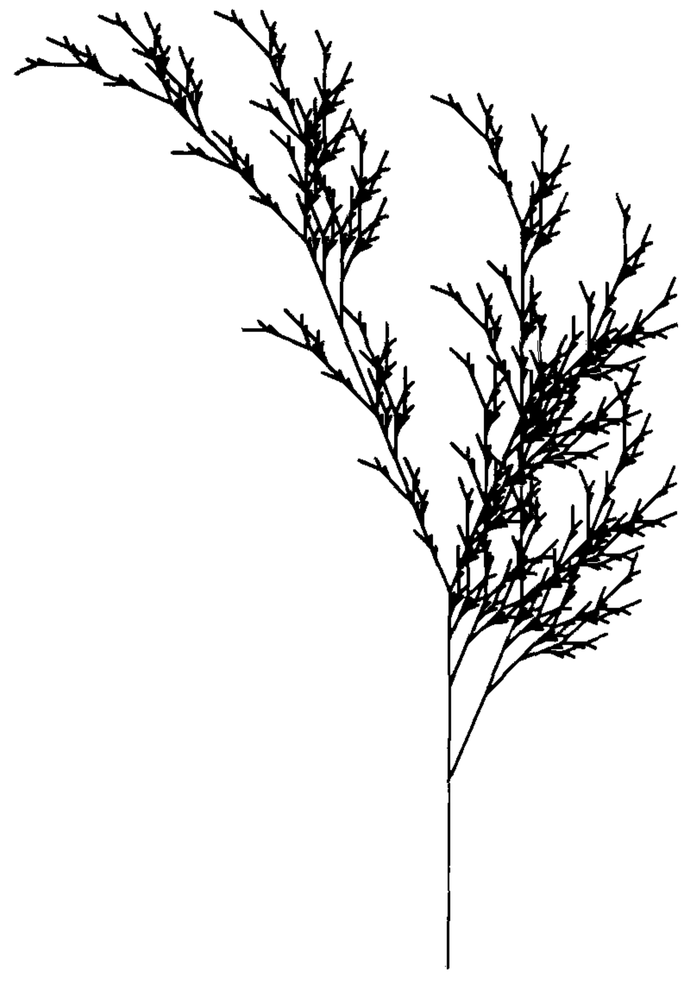
Figure 6.12: Example of interpreted L-system words

This principle can be extended to produce three dimensional images by increasing the number of letters in the subset of letters used to control the turtle.

6.7.3 Branching Structures

The L-systems discussed so far have only been able to produce structures consisting of joined and unjoined line-segment sequences. However many structures in nature require branching structures and a method for achieving this is to use a *Bracketed D0L-system.* Two letters are used to delimit a branch, typically the letters "[" and "]". When used in conjunction with a turtle drawing system during interpretation of a derived word, the "[" letter when encountered is used to push onto a stack the current attributes of the turtle, including its position and heading . The "]" letter is interpreted to mean pop the top-most set of attributes off the stack and assign them to the turtle; this may change the location and heading of the turtle. The bracketed D0L-system is derived in the same manner as an D0L-system, except that the branch delimiters are always rewritten as themselves. With

the addition of branching structures, very simple L-systems (see Figure 6.13) can give rise to impressively realistic models of branching plant-like structures in only a few iterations (in the example given, five iterations).



$\omega : X$

$\rho_1 : X \longrightarrow F - [[X] + X] + F[+FX] - X$

$\rho_2 : F \longrightarrow FF$

Figure 6.13: Example of a Bracketed D0L-system and interpretation

6.7.4 Stochastic L-systems

Any attempt to produce a scene involving many plants using the same D0L-system leads to a very artificial looking image, due to all the plants having essentially identical structures. By using a stochastic application of productions, a variation in topology and geometry of the plants can be achieved. For each letter in the alphabet of the L-system, a set of productions exist, where each production in that set has a probability value associated with it. All the probabilities within a set must sum to unity. During derivation one production from a set is chosen stochastically using the associated probabilities. Thus, different productions with same predecessor can be applied to various occurrences of the same letter in one derivation step. An example of a simple stochastic l-system is shown in Figure 6.14.

$$\omega : F$$
$$\rho_1 : F \xrightarrow{0.33} F[+F]F[-F]F$$
$$\rho_2 : F \xrightarrow{0.33} F[+F]F$$
$$\rho_3 : F \xrightarrow{0.34} F[-F]F$$

Figure 6.14: Example of a Stochastic L-system description and interpretation

6.7.5 Context sensitive L-systems

An extension of the standard D0L-system is to introduce the concept of context into the system. This mechanism is useful for modelling the effects of nutriments, hormones, interacting parts of a plant/organism, or cell lineage. The over all class of context-sensitive L-systems are termed *IL-systems* with its syntax shown in Figure 6.15. Each production in a IL-system has a *left context* and a *right context* associated with the predecessor and for that production to be applied to the current target letter, the pattern of letters specified in the left/right context must match the pattern of letters in the current word to the left/right of the target letter. Each context may have 0-n letters associated with it, but if either the left or right context has zero letters associated with it, then the context to that side is ignored. Sub-classes of context sensitive L-systems are sometimes referred to a 1L-systems, where the is either a left or right context consisting of one letter. or 2L-systems, where there is both a left and right context each consisting of one letter. A 0L-system is a L-system where no context is specified.

$$\rho_n : leftcontext < predecessor > rightcontext \longrightarrow successor$$

Figure 6.15: IL-system syntax

The interpretation of a L-system word becomes more involved when IL-systems and Branching L-systems are combined due to the nature of what constitutes a neighbouring letter. Consider the word 'ab[i+FFFj]cd', when interpreted the 'distance' from letter 'b' to letter 'c' should be represented as one unit and the 'distance' from letter 'b' to letter 'h' should be represented as four units. The right context for the letter 'b' should thus be 'c'. Therefore any context matching must skip over any intervening branch delimiters. Also the letter 'i' does not have any left context and the letter 'j' any right context.

IL-systems can be used to allow the potential of communications between nodes of the growing network. Productions in OL-systems are not reliant on the what has happened in previous derivations. However, it may be important that production application may also depend on the predecessor's context. This is useful in simulating interaction between different parts of the growing network, e.g. signalling of different concentrations of guiding chemicals in the network. To illustrate this, consider the IL-system presented in Figure 6.16, and its derivation over four iterations as shown in Figure 6.17. The letter 'b' moves along the word from left to right, as if a signal were propagating along the length of the word. In the final derivation the letter 'c' is changed to 'd' which could be interpreted to mean change the cell from type 'c' to 'd'. This combined with environment querying can create a enormous wealth of possibilities for network construction. Expand with info from pages 30–36 in (Prusinkiewicz & Lindenmayer, 1990).

$$\omega : baaaaaaaa$$

$$\rho_1 : b < a \longrightarrow b$$

$$\rho_2 : b \longrightarrow a$$

$$\rho_3 : b < c \longrightarrow d$$

Figure 6.16: Example of a Context Sensitive l-system capable of representing chemical signalling

Step	L-system string
n	baaaac
n+1	abaaac
n+2	aabaac
n+3	aaabac
n+4	aaaabc
n+5	aaaabd

Figure 6.17: Derivation of chemical signalling l-system

6.7.6 Parametric L-systems

A number of problems arise in using the L-systems that have been described so far:

- it is often very difficult to create an L-system where the growth of the derived string is constrained to a reasonable size. Looking at nature most structures follow a sigmoidal growth pattern; the structure starts off being small, goes through a phase of rapid expansion and then growth tails off when the structure is mature. Often if ill-designed, an L-system can lead to exponential growth patterns where the growth never tails off.

- the power of modelling structures using a D0L-system can be limited by the fact that many modelling instructions are limited to unit lengths .....

These problems can be overcome by using *parametric L-systems*, whose syntax is shown in Figure 6.18. Each letter in the valid alphabet of the L-system has a number of parameters associated with it. The combination of a letter and a parameter list is termed a *module*. If a production is to be applied to a module, the letter of the predecessor module must match the letter of the target module, the number of parameters of the predecessor module and the target module must be the of the same arity, and the condition statement must evaluate to be true when the actual values of the target module parameters are used into the production. By the inclusion of parameters and a conditional clause the situations under which the rule is selected can be controlled. A simple example is shown in Figure 6.19, where the derived string will never grow bigger than 6 modules.

$$\rho_n : predecessor(parameter_1 ... parametern) : condition \longrightarrow successor$$

$$condition \rightarrow conditionexpressionlogicaloperatorconditionexpression$$

$$conditionexpression \rightarrow conditionexpression|arithmeticexpression$$

Figure 6.18: IL-system syntax

By combining IL-systems and Parametric L-systems, a much more powerful system can be envisaged for signalling.

$\omega : a(5)$

$\rho_1 : a(i) : i > 0 \longrightarrow a(i-1)b(i)$

Step	L-system string
0	a(5)
1	a(4)b(5)
2	a(3)b(4)b(5)
3	a(2)b(3)b(4)b(5)
4	a(1)b(2)b(3)b(4)b(5)
5	a(0)b(1)b(2)b(3)b(4)b(5)

Figure 6.19: Example of interpreted l-system strings

6.7.7 Environmentally Sensitive L-systems

One aspect of the L-systems described so far that has been lacking is the interaction between the model described by the L-system and the environment in which the model is embedded in. In some cases this can be safely ignored because there is no environment surrounding the model. but in other cases the interaction between the two can be of great importance. This type of L-system was primarily developed to show how various environmental factors, in the case of plants factors such as temperature, resources, obstacles and interacting sub-structures of the model, can affect the growth and behaviour of the plants being modelled. In particular this type of L-system can be used to understand how competition and cooperation between substructures can affect the resulting overall structure. To achieve this in L-systems a two-way communications channel must be established; the object being modelled must be able to affect the environment in some manner, through the reception of information, processing the information and generating a response, and the environment must be able to affect the plant in some manner, through manipulation of environmental factors. In the L-systems described so far there is a mechanism for the object being modelled to affect the environment through the use of a turtle interpretation, the object can grow and move through the environment. This leaves the other side of the communications channel. By introducing a special reserved symbol that can be used to

interrogate the environment, the L-system can receive information from the environment and act upon it.

Unlike previously described L-systems where the interpretation of the L-system word did not take place until after the final derivation, in an environmentally sensitive L-system the L-system word must be interpreted at each derivation, and indeed must be interpreted during the rewrite process of the word.

6.8 Combining L-Systems and Neural Networks

As previously stated this model is based on the work done by Aho et al. (1977). The model uses a map like L-System to control the growth and positioning of the neurons on the network substrate. Each rule (see Figure 6.20) in the L-System consists of a predecessor, condition and successor but unlike a string based L-System, the predecessor and the successor consist of a 3x3 grid, termed a "Cell-Matrix". This concept of a grid is based the idea that in nature the way a neurons grows and develops is influenced by the surrounding tissues of the neuron. In this way the L-System is being used as a context sensitive L-System, with the eight cells surrounding the central cell of 3x3 cell-matrix providing the context of the central cell.

A L-System rule is fired when a predecessor cell-matrix matches part of the substrate and the condition is fulfilled. A successful match results in the cell which maps on to the centre cell of the predecessor cell-matrix being changed to the value given by the successor of the rule. The central value in the predecessor cell-matrix must match the equivalent value in the substrate but the surrounding eight values of the cell-matrix may contain either fixed values or one of three wildcard values. If a fixed value is used then value in the equivalent substrate cell must match exactly for the predecessor cell-matrix to match. The three wildcard values are:

- "NoCell" indicates that the equivalent cell in the substrate must be empty for a match to occur.

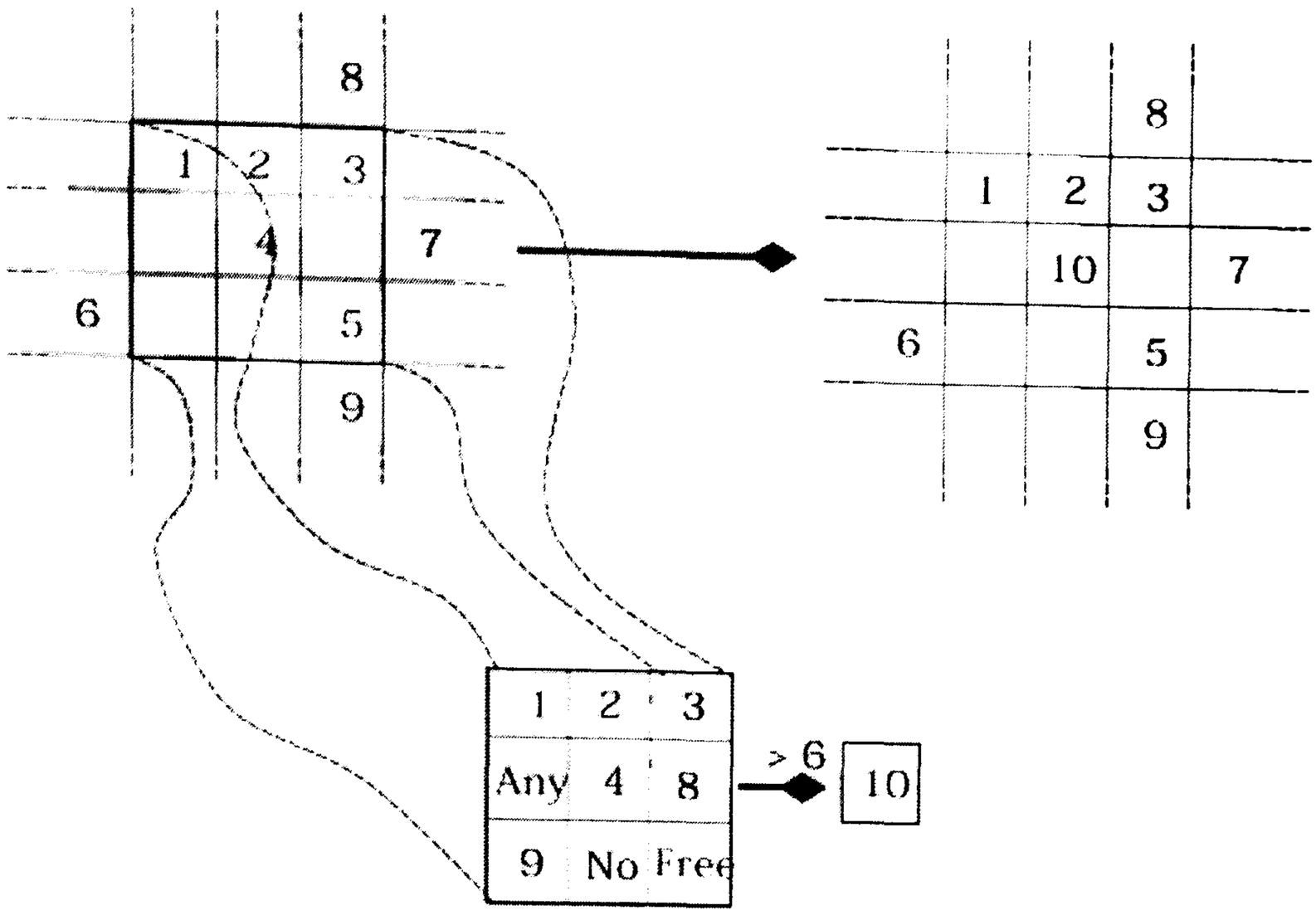


Figure 6.20: Example of an lsystem rule

- "FreeCell" indicates that the equivalent cell in the substrate must *not* be empty for a match to occur but the value can be of any value.

- "AnyCell" indicates that the equivalent cell in the substrate may be empty or filled for a match to occur.

In the original model as described by Aho et al. (1977) the condition for a rule could be one of four types, an AND condition, OR condition, LESS-THAN condition and a MORE-THAN condition. The AND condition required that every cell in the predecessor cell-matrix matched every equivalent cell in the substrate before the successor could be applied. The OR condition required that one cell in the predecessor cell-matrix matched its equivalent cell in the substrate before the successor could be applied. The LESS-THAN condition required that at less than a specified number of cells in the predecessor cell-matrix matched their equivalent cell in the substrate before the successor could be applied. The MORE-THAN condition required that at least a specified number of cells in the predecessor cell-matrix matched their equivalent cell in the substrate before the successor could be applied. This has been altered to make the conditions simpler but still having the equivalent expressiveness. The model developed in this thesis uses only the

LESS-THAN and MORE-THAN conditions. The AND condition can be expressed by either using a MORE-THAN condition with a value of nine, ie every cell has to match. The OR condition can be expressed by either using a MORE-THAN condition with a value of one, ie only one cell has to match.

Figure 6.20 shows a rule with a predecessor cell-matrix, a MORE-THAN condition and a successor. The predecessor cell-matrix has central value is four, and surrounding values of (given a clockwise ordering from the top-left corner) “one”, “two”, “three”, “eight”, “Free”, “No”, “nine”, and “Any”. The MORE-THAN condition specifies that at least six matches must occur before the successor can apply, with the successor having a value of ten. Using Figure 6.20 as an example, the rule can be applied to the section of substrate in question because at least six of the cells match. Thus the cell in the substrate (matching the equivalent centre cell in the predecessor cell-matrix) is changed from a four to a ten.

Each rule has an associated “age” limit which restricts the number of times the rule can be applied in the creation of a neural network. One parallel rewriting step can only consume one unit of age. Thus if the rule is applied several times during one step only one unit is used. This restriction is used to stop a run-away growth process and allows the rewriting stage to come to a “natural” stop.

To generate the nodes in the network a blank substrate is created and a seed cell placed in the centre of the substrate. The L-System rules are then applied until no rules are fired. This iterative process determines the number and location of the nodes on the substrate.

After the nodes have been placed on the substrate, the connections between the nodes must be determined. Each node has two parameters associated with it, a scent value and a scent target value. The scent target value is used during the growing phase to direct the axon growth of a neuron towards other nodes whose scent value matches the scent target value of the node. The number in the substrate determines the node type of the node which in turn determines the scent value and scent target values for the node.

Two types of connections are grown, a single axon and one or more dendrites. Axons are grown in the direction of the input area of the substrate towards the output area of the substrate, and dendrites are grown in the direction of outputs towaRds inputs. Axons use

the scent codes to determine their individual direction in the substrate, growing towards the area that has the highest concentration of the scent target that it is looking for.

A range of five values have been used for the possible values of the scent codes in this model. This restriction is in place to limit the search space of the GA.

Once the axons for each node have been grown, the dendrites for each node are grown. A neuron grows a dendrite towards the input side of the substrate until such time as it reaches its maximal length which is determined by values associated with each node type called the *distance* and *spread* values. The distance value is a range between 0 and 1 with increments of 0.2. This determines the maximal distance a dendrite can grow from the node and is a relational distance to the substrate. A 0 indicates that the dendrite cannot grow at all, a 1 indicates the dendrite can grow to a length equivalent to the full length of the substrate, and 0.2 would indicate that the dendrite could grow to a length equivalent to one fifth the length of the substrate. The spread value indicates how far reaching to either side of the node that the dendrite can spread to. Again this value is a range between 0 and 1 with 0.2 increments. A value of 0 indicates that the dendrite can only grow and connect to axons that lie in a direct path between the node and the input area of substrate in a band one substrate cell high. A value of one indicates that the node can search a space for connections equivalent to the full width of the substrate along its distance. A spread value of 0.2 would indicate that the node could connect to any axons found with an area equivalent to one fifth of the width of the substrate along the distance as indicated by the distance value. Thus a neuron with a distance value of 1 and a spread value of 1 which was placed close to the output end of the substrate would connect to every axon in the substrate.

Once the stages of placing the nodes and growing the node processes has been achieved the input and output nodes are connected. The model as described by Aho et al. (1977) selects "Ni" nodes closest to edge of the substrate to become input nodes, where Ni is the required number of input nodes, and "No" nodes on the opposite edge of the substrate to become output nodes, where No is the required number of output nodes. Only those nodes with connections to other nodes are selected as this is more likely to produce working

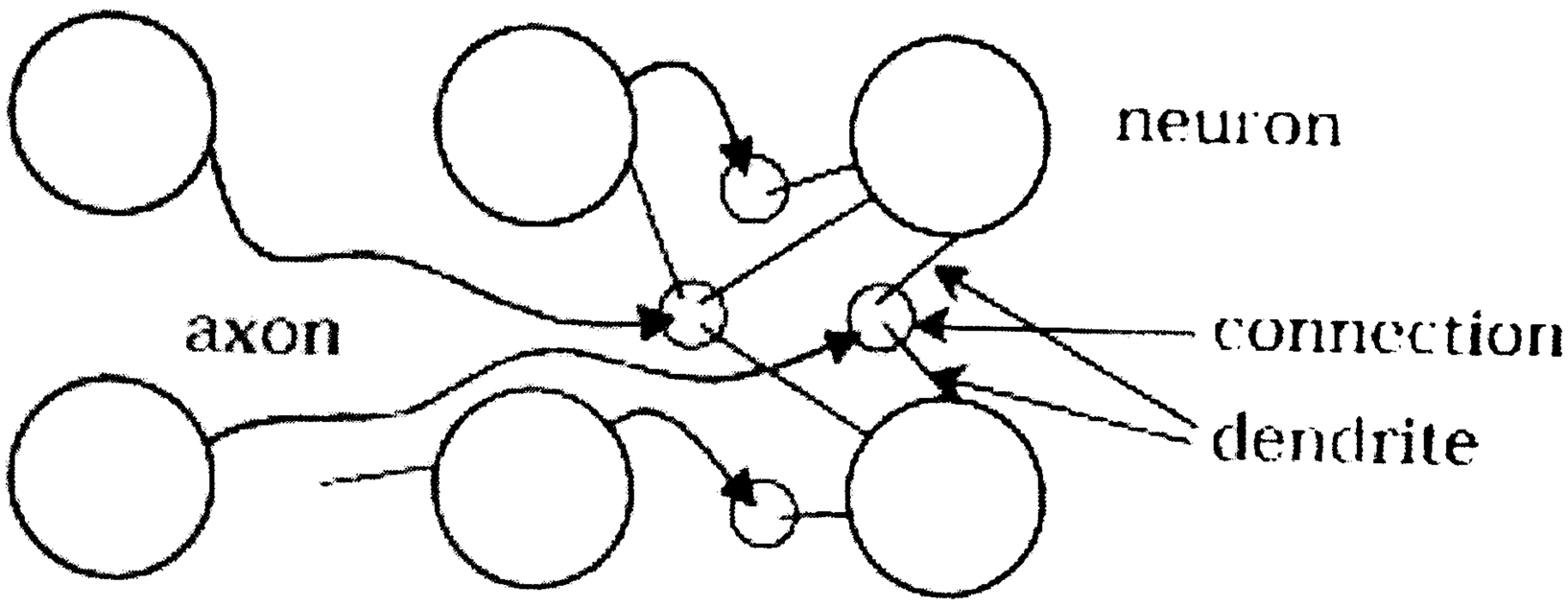


Figure 6.21: Example of the result of neuron growth controlled by the lsystem

networks. Thus the network determines both the placing of the input layer, hidden layer and output layer nodes.

The model presented in this thesis uses a modified version of this process. This model divides the substrate into sections, with an area the full width of the substrate and one quarter the length of the substrate, adjacent to one edge, being designated the “input area” and similarly an area, again the full width and one quarter the length of the substrate, but on the opposite edge of the substrate being designated the ”output area”. Any nodes or dendrites located in the input area are connected to a layer of input nodes, any nodes or axons located in the output layer are connected an a layer of output nodes. Thus the nodes in the substrate only consist of hidden nodes. The input layer nodes and the output layer nodes in effect are outside of the growth process and form a constant and fixed structure.

6.9 Conclusion

In this chapter we demonstrated the power of the intellectual infrastructure through its application in the two models discussed in sections 6.6 and 6.7. The two models we introduced in this structure; a physical structure representation and a finer grained representation. There are many other topics that could be used to demonstrate the power of the

intellectual infrastructure such as axon-dendrite growth and neural network dynamics but these are left as further research beyond the temporal embrace of the PhD.

7

Summary and Conclusions

7.1 Summary

In this dissertation we have taken a variety of bodies of knowledge and created and developed a new intellectual infrastructure for them to produce an almost continuous understanding of the subject. Chapters 2-4 took these topics in turn to develop this infrastructure.

In chapter 2 we looked at the fundamental understanding of biological neural networks which for many researchers is the analogous system that underlies their work in neural networks.

In chapter 3 we took the biological neural networks and demonstrated how they could be modelled as neural networks. In chapter 4 in order to construct artificial neural networks we chose to use the Genetic Algorithms tool, not as a principal of optimality, but as a reasonably appropriate approach to take which matched the development of the developmental infrastructure and the personal intellectual priorities of the researcher.

In chapter 5 we see the fulfilment of the intellectual infrastructure in the structure of the artificial neural network.

In chapter 6 we exemplified the intellectual infrastructure's benefits through its application to two models.

7.2 Conclusions

This research has concentrated on original investigation in order to obtain knowledge and understanding of the subject area of neural network construction. In order to achieve this it has combined biological concepts with neural network modelling using genetic algorithms as a tool in artificial neural network construction. The result of this effort has been the creation and development an intellectual infrastructure for this approach to artificial neural network construction. As a work of knowledge and understanding it is in some respects merely a minor step in the quest for a more complete understanding, however such full understanding usually requires the contributions of many minor steps and I am pleased to be one of the contributors to these.

Chapter 6 has demonstrated that when used in practice the intellectual infrastructure has commendable substance.

7.3 Critical Review

I set out on my research journey like many aspiring PhD students wishing to make a noticeable contribution to the body of knowledge in a subject. Like the vast majority of PhD students I have learnt the business of doing research and therefore in my intellectual development have been humbled as to the contribution that a three year apprenticeship can make. This dissertation I would have wished to have ended more substantially but in the quest to understand all the relevant research I, in the time available, have only been able to make my contribution in the area of improved understanding and knowledge.

Suitably chastened I look forward to fulfilling more modest ambitions as well as sharing my experience with coming generations of excited, over ambitious, intelligent, would-be PhD students.

References

Ackley, D., & Littman, M. (1992). Interactions between learning and evolution. In C. Langton, C. Taylor, J. Farmer, & . Rasmussen (Eds.), *Artificial life ii.* Wokingham,England: Addison-Wesley.

Aho, A. V., Johnson, S. C., & Ullman, J. D. (1977). Code generation for expressions with common subexpressions. *JACM*, *24*(1), 146–160.

Albeck, Y. (1995). Sound localization and binaural processing. In M. Arbib (Ed.), *The handbook of brain theory and neural networks*. London: The MIT Press.

Aleksander, I., & Morton, H. (1995). *An introduction to neural computing*. London: International Thomson Computer Press.

Anand, R., Mehrotra, K., Mohan, C., & Ranka, S. (1995). Efficient classification for multiclass problems using modular neural networks. *IEEE Transactions on Neural Networks*, *6*(1), 117-124.

Ash, T. (1989). Dynamic node creation in backpropagation networks. *Connection Science*, *1*, 365-375.

Bäck, T., Fogel, D. B., & Michalewicz, Z. (Eds.). (1997). *Handbook of evolutionary computation.* Oxford: Oxford University Press.

Balakrishnan, K., & Honavar, V. (1997). *Evolutionary design of neural architectures - a preliminary taxonomy and guide to literature.* World Wide Web Document. Available : http://www.cs.iastate.edu/ balakris/papers/TR95-01.ps. (20/02/97)

Ballard, D. (1990). Modular learning in hierarchical neural networks. In E. Schwartz (Ed.), *Computational neuroscience.* Cambridge, MA: MIT Press.

Banzhaf, W., Francone, F. D., Keller, R. E., & Nordin, P. (1998). *Genetic programming: An introduction.* San Francisco, CA, USA: Morgan Kaufmann.

Beale, R., & Jackson, T. (1992). *Neural computing : An introduction.* Bristol: IOP Publishing Ltd.

Belew, R., McInerney, J., & Schraudolph, N. (1991). Evolving networks : Using the genetic algorithm with connectionist learning. In C. Langton, C. Taylor, J. Farmer, & . Rasmussen (Eds.), *Artificial life ii.* Wokingham,England: Addison-Wesley.

Bishop, C. (1995). *Neural networks for pattern recognition.* Oxford: Oxford Press. (CHECK PUBLISHER!)

Boers, E., & Kuiper, H. (1992/1996). Biological metaphors and the design of modular artificial neural networks. (master's thesis, leiden university.). In J. Kodjabachian & J. Meyer (Eds.), *?????*

Bove, M., Guigliano, M., Grattatola, S., & Massobrio, G. (1998). Dynamics of networks of biological neurons: Simulation and experimental tools. In C. Leondes (Ed.), *Neural network systems techniques and applications (1): Algorithms and architectures.* London: Academic Press.

Brown, T., Zador, A., Mainen, Z., & Claiborne, B. (1992). Hebbian computations in hippocampal dendrites and spines. In T. McKenna, J. Davis, & S. Zornetzer (Eds.), *Single neuron computation. neural nets: Foundations to applications.* Boston: Academic Press.

Calabretta, R., Wagner, G., Nolfi, S., & Parisi, D. (1997). Evolutionary mechanisms for the origin of modular design in artificial neural networks. In Y. Bar-Yam (Ed.), *Proceedings of the 1997 international conference on complex systems.* ?

Cangelosi, A., Parisi, D., & Nolfi, S. (18/8/97). *Cell division and migration in a 'genotype' for neural networks.* World Wide Web Document. http://kant.irmkant.rm.cnr.it/nolfi.html.

Cariani, P., & Delgutte, B. (1996). Neural correlates of the pitch of complex tones I. pitch and pitch salience. *Journal of Neurophysiology*, *76*(3), 1698-1716.

Carr, C. (1993). Processing of temporal information in the brain. *Annual Review of Neuroscience, 16*, 223-243.

Darwin, C. (1859). *The origin of species.* London: John Murray.

Durbin, R., Miall, C., & Mitchison, G. (Eds.). (1989). *The computing neuron.* London: Addison-Wesley Publishers Ltd.

Eccles, J. (1981). FINS OUT TITLE. *Neuroscience, 6*, 1839—1855.

Edelman, E. (1981). Group selection as the basis for higher brain function. In F. Schmitt (Ed.), *The organisation of the cerebral cortex.* New York: MIT Press.

Fahlman, S., & Lebiere, C. (1990). The cascade-correlation learning architecture. In D. Touretzky (Ed.), *Advances in neural information processing systems 2.* San Mateo, CA: Morgan Kaufmann.

FitzHugh, R. (1969). Mathematical models of excitation and propogation in nerve. In H. P. Schwan (Ed.), *Biological engineering.* New York: McGraw-Hill.

Freeman, J. (1994). *Simulating neural networks.* Wokingham: Addison-Wesley.

Gardner, D. (1993). Towards *neural* neural networks. In D. Gardner (Ed.), *The neurobiology of neural networks* (p. 1-11). London: MIT Press.

Gazzaniga, M. (1989). Orgainisation of the human brain. *Science, 245*, 947—952.

Geman, S., Bienenstock, E., & Doursat, R. (1993). Neural networks and the bias/variance dilemma. *Neural Computation, 4*, 1-58.

Getting, P. (1989). Emerging principles governing the operation of neural networks. *Annual Review of Neuroscience.*

Goldberg, D. (1989). *Genetic algorithms in search, optimization and machine learning.* New York: Addison-Wesley Inc.

Goodman, C. (1996). Mechanisms and molecules that control growth cone guidance. *Annual Review of Neuroscience, 19*, 341-377.

Gould, E., Tanapat, P., McEwen, B., Flugge, G., & Fuche, E. (1998). Proliferation of granule cell precursors in the dentate gyrus in monkeys is diminished by stress. *Proceedings of the National Academy of Sciences U.S.A., 95*, 3168-3171.

Gruau, F. (1992). *Cellular encoding of genetic neural networks.* World Wide Web Document. Available : http://www.cwi.nl/ gruau/node4.html.

Gruau, F., & Whitley, D. (1993). *The cellular developmental of neural networks: the interaction of learning and evolution.* World Wide Web Document. Available : http://www.cwi.nl/ gruau/node4.html.

Happel, B., & Murre, J. (1994). Design and evolution of modular network architectures. *Neural Networks, 7*(6–7), 985–1004.

Harris-Warric, R. (1988). Chemical modulation of central pattern generators. In A. Cohen, S. Rossignol, & S. Grillner (Eds.), *Neural control of rhythmic movements in vertebrates.* New York: John Wiley and Sons. (check page numbers!)

Hassibi, B., & Stork, D. (1993). Second order derivatives for network pruning: Optimal brain surgery. In S. Hanson, J. Cowan, & C. Giles (Eds.), *Advances in neural information processing systems 5.* San Mateo, CA: Morgan Kaufmann. (check page numbers!)

Haykin, S. (1999). *Neural networks: A comprehensive foundation.* Upper Saddle River, NJ: Prentice-Hall Inc.

Hecht-Nielsen, R. (1990). *Neurocomputing.* New York: Addison-Wesley Inc.

Hille, B. (1992). *Ionic channels of excitable membranes.* Sunderland, MA: Sinauer Associates.

Hodgkin, A., & Huxley, A. (1952). Currents carried by sodium and potassium ions through the membrane of the giant axon of Loligo. *Journal of Physiology, 117*, 500-544.

Holland, J. (1975). *Adaption in natural and artificial systems.* Ann Arbot: The University of Michigan Press.

Juedes, D., & Balakrishnan, K. (1998). *Generalized neural networks, computation differentiation, and evolution.* World Wide Web Document. Available : ftp://ftp.math.tu-dresden.de/pub/ADOLC/PAPERS/juedes.ps.gz.

Kaczmarek, L., & Levitan, I. (1987). *Neuromodulation.* New York: Oxford Uni. Press.

Keynes, R., & Cook, G. (1992). Repellant cues in axon guidance. *Current Opinion in Neurobiology, 2,* 55-59.

Koch, C., & Bernander, O. (1995). Axonal modelling. In M. Arbib (Ed.), *The handbook of brain theory and neural networks.* London: The MIT Press.

Kodjabachian, J., & Meyer, J. (02/04/97). *Evolution and development of neural networks controlling locomotion, gradient-following and obstacle-avoidance in artificial insects.* World Wide Web Document. Available at : http://www.biologie.ens.fr/AnimatLab/perso/meyer/kodja/loco2D.ps.gz.

Kosslyn, S., Flynn, R., Amsterdam, J., & Wang, G. (1990). Components of high level vision: A cognitive neuroscience analysis and accounts of neurological syndromes. *Cognition, 34,* 203—277.

Koza, J. (1993). *Genetic programming I.* London: MIT Press.

Koza, J. (1994). *Genetic programming II.* London: MIT Press.

LeCun, Y., Denker, J., & Solla, S. (1990). Optimal brain damage. In D. Touretzky (Ed.), *Advances in neural information processing systems 2* (p. 598-605). San Mateo, CA: Morgan Kaufmann.

Leerink, L., Giles, C., Horne, B., & Jabri, M. (1995). Learning with product units. In *Advances in neural information processing systems 7* (pp. 537–544). London: MIT Press.

Levitan, I., & Kaczmarek, L. (1997). *The Neuron: Cell and Molecular Biology* (4th ed.). New York: Oxford University Press.

Lewis, L., & Albrecht-Buehler, G. (1987). Role of cortical tension in fibroblast shape and movement. *Cell Motil Cytoskeleton*, *7*(1), 54–67.

Livingstone, M., & Hubel, D. (1988). Segregation of form, color, movement, and depth: Anatomy, physiology, and perception. *Science*, *240*, 740—749.

Lockery, S., & Sejnowski, T. (1993). Realistic network models of distributed processing in the leech. In D. Gardner (Ed.), *The neurobiology of neural networks* (pp. 107–135). London: MIT Press.

Mallot, H., & Giannakopoulos, F. (1996). Population networks : a large-scale framework for modelling cortical neural networks. *Biological Cybernetics*, *75*, 441–452.

Maren, A. (1990). Neural network structures : Form follows function. In A. Maren, C. Harston, & R. Pap (Eds.), *Handbook of neural computing applications* (pp. 45–57). London: Academic Press.

McCulloch, W. S., & Pitts, W. H. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, *5*, 115–133.

Merrill, J., & R., P. (1988). A stochastic learning algorithm for neural networks. In C. Langton, C. Taylor, J. Farmer, & . Rasmussen (Eds.), *Artificial life ii*. Wokingham,England: Addison-Wesley.

Mezard, M., & Nadal, J. (1989). Learning in feed-forward layered network: The Tiling algorithm. *Journal of Physics A*, *22*, 2191-2204.

Michalewicz, Z. (1996). *Genetic algorithms + data structures = evolutionary programs*. London: Springer.

Miller, G., Todd, P., & Hegde, S. (1993). Designing neural networks using genetic algorithms. In J. Schaffer (Ed.), *Proceedings of the third international conference on genetic algorithms* (pp. 379–384). San Mateo, CA: Morgan Kaufmann.

Mountcastle, V. (1979). An organizing principle for cerebral function: the unit module and distribution function. In F. Schmitt & F. Worden (Eds.), *The neurosciences: fourth study program*. Cambridge: MIT Press.

Murre, J. (1993). Transputers and neural networks: An analysis of implementation and constarints. *IEEE Transactions on Neural Networks, 4*, 284—292.

Nelson, M., & Bower, J. (1990). Brain maps and parallel computers. *Trends in Neuroscience, 13*, 403—408.

Nolfi, S. (1997). Using emergent modularity to develop control systems for mobile robots. *Adaptive Behaviour, 3-4*, 343-364.

Nottebohm, F. (1991). Reassessing the mechanisms and origins of vocal learning in birds. *Trends in Neuroscience, 14*, 211-213.

Posner, M., Peterson, S., Fox, P., & Raichle, M. (1998). Localisation of cognitive operations in the human brain. *Science, 240*, 1627—1631.

Principe, J., Euliano, N., & Lefebvre, W. (2000). *Neural and adaptive systems: fundamentals through simulations.* New York: John Wiley and Sons, Inc.

Prusinkiewicz, P., & Lindenmayer, A. (1990). *The algorithmic beauty of plants.* New York: Springer.

Pudipeddi, B., Abbot, A. L., & Athanas, P. M. (1998). A configurable computing approach towards real-time target tracking. In J. D. P. Rolim (Ed.), *Parallel and distributed processing, 10 ipps/spdp'98 workshops held in conjunction with the 12th international parallel processing symposium and 9th symposium on parallel and distributed processing, orlando, florida, usa, march 30 - april 3, 1998* (Vol. 1388, pp. 79–84). Springer.

Purves, D., & Lichtman, J. (1985). *Principles of neural development.* Sunderland, Massaschusetts: Sinauer Assoc. Inc.

Rakic, P. (1985). Mechanisms of neuronal migration in developing cerebellar cortex. In G. Edelman, W. Cowan, & E. Gull (Eds.), *Molecular basis of neural development.* New York: Wiley.

Rall, W., & Agmon-Snir, H. (1998). Cable theory for dendritic neurons. In C. Koch & I. Segev (Eds.), *Methods in neuronal modelling: From ions to networks*. London: The MIT Press.

Reed, R. (1993). Pruning algorithms: A survey. *IEEE Transactions on Neural Networks*, *4*, 740-744.

Reichardt, L., & Tomaselli, K. (1991). Extracellular matrix molecules and their receptors: functions in nerual development. *Annual Review of Neuroscience*, *14*, 531-570.

Reike, F., Warland, D., Steveninck, R. de Ruyter van, & Bialek, W. (1997). *Spikes: Exploring the neural code*. Cambridge: MIT Press.

Rosenblatt, F. (1958). The perceptron: a probalistic model for information storage and organisation in the brain. *Psychological Review*, *65*, 386-408.

Rosenzweig, M., Leiman, A., & Breedlove, S. M. (1999). *Biological psychology: An introduction to behavioural, cognitiveand clinical neuroscience*. Sunderland, Mass: Sinauer Associates.

Rumelhart, D., & McClelland, J. (1986). *Parallel distributed processing : explorations in the microstructure of cognition. vol.1*. London: MIT Press.

Schaffer, J., & Morishima, A. (1985). An adaptive crossover distibution mechanism for genetic algorithms. In C. Langton, C. Taylor, J. Farmer, & . Rasmussen (Eds.), *Artificial life ii*. Wokingham,England: Addison-Wesley.

Segev, I. (1992). Single neuron models: oversimple, complex and reduced. *Trends in Neuroscience*, *15*(11), 414–421.

Segev, I., & Burke, R. (1998). Compartmental models of complex neurons. In C. Koch & I. Segev (Eds.), *Methods in neuronal modelling: From ions to networks*. London: The MIT Press.

Shakespeare, W. (1601). *Hamlet*. (Act II, Scene ii)

Sharkey, A. (1997). Modularity, combining and artifical neural nets. *Connection Science*, *9*(1), 3–10.

Sietsma, I., & Dow, R. (1991). Creating artificial neural networks that generalize. *Neural Networks*, *4*, 67-79.

Snyder, A., Abdullaev, Y., Posner, M., & Raichle, M. (1995). Scalp electrical potentials reflect regional cerebral blood flow responses during processing of written words. *Proceedings of the National Academy of Sciences USA*, *92*, 1689-1693.

Softky, W., & Koch, C. (1995). Single-cell models. In M. Arbib (Ed.), *The handbook of brain theory and neural networks*. London: The MIT Press.

Song, H., Ming, G., He, Z., Lehmann, M., McKerracher, L., Tessier-Lavigne, M., & Poo, M. (1998). Conversion of neuronal growth cone responses from repulsion to attraction by cyclic nucleotides. *Science*, *281*, 1515–1518.

Stevens, C. (1993). Two principles of brain organisation: A challenge for artifical neural networks. In D. Gardner (Ed.), *The neurobiology of neural networks* (p. 13-20). London: MIT Press.

Tessier-Lavigne, M., & Placzek, M. (1991). Target attraction: are developing axons guided by chemotropism? *Trends in Neuroscience*, *14*, 303-310.

Todd, P. (1988). Evolutionary methods for connectionist architectures. In C. Langton, C. Taylor, J. Farmer, & . Rasmussen (Eds.), *Artificial life ii*. Wokingham,England: Addison-Wesley.

Uhr, L. (1973). Decider-1: A system that chooses among different types of acts. In *Proc. of the 3rd ijcai* (p. 396-401). Stanford, MA.

Vaario, J., Ogata, N., & Shimohara, K. (1996). Synthesis of environment directed and genetic growth. In *Artificial life v.* May 14-16, 1996, Nara, Japan.

Vaario, J., & Ohsuga, S. (1992). An emergent construction of adaptive neural architectures. *Heuristics - The Journal of Knowledge Engineering*, *5*(2), 1–12. (WWW : http://mi-2.mech.kobe-u.ac.jp/ jari/Publications/vaario_heuristics.ps)

Vaario, J., & Ohsuga, S. (1994). On growing intelligence. In G. Dorffner (Ed.), *Neural networks and a new AI.* London: Chapman & Hall. ((Publisher seems to be changed.))

Vaario, J., Onitsuka, A., & Shimohara, K. (1997). Formation of neural structures. In *Ecal97.*

Vaario, J., & Shimohara, K. (1995). On fromation of structures. *Lecture Notes in Artificial Intelligence, 929,* 421-435.

Van Essen, D., Anderson, C., & Felleman, D. (1992). Information processing in the primate visual cortex: An integrated systems approach. *Science, 255,* 419—423.

Van Rooij, A., Jain, L., & Johnson, R. (1998). *Neural network training using genetic algorithms.* Singapore: World Scientific.

Yaeger, L. (1996). *Computational genetics, physiology, metabolism, neural systems, learning, vision and behaviour.* World Wide Web Document. Available : http://www.apple.com/ yaeger.

Zeki, S., & Shipp, S. (1988). The functional logic of cortical connections. *Nature, 335,* 311—317.

Zipser, D. (1989). A subgrouping strategy that reduces complexity and speeds up learning in recurrent networks. *Neural Computation, 1,* 551-557.