The Functionality of Spatial and Time Domain Artificial Neural Models

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

Niccolo Francesco Capanni

School of Engineering
The Robert Gordon University
Aberdeen, Scotland, August 2006

Declaration

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

Niccolo Francesco Capanni

2006

Abstract

This thesis investigates the functionality of the units used in connectionist Artificial Intelligence systems. Artificial Neural Networks form the foundation of the research and their units, Artificial Neurons, are first compared with alternative models. This initial work is mainly in the spatial-domain and introduces a new neural model, termed a Taylor Series neuron. This is designed to be flexible enough to assume most mathematical functions. The unit is based on Power Series theory and a specifically implemented Taylor Series neuron is demonstrated. These neurons are of particularly usefulness in evolutionary networks as they allow the complexity to increase without adding units. Training is achieved via various traditional and derived methods based on the Delta Rule, Backpropagation, Genetic Algorithms and associated evolutionary techniques. This new neural unit has been presented as a controllable and more highly functional alternative to previous models.

The work on the Taylor Series neuron moved into time-domain behaviour and through the investigation of neural oscillators led to an examination of single-celled intelligence from which the later work developed.

Connectionist approaches to Artificial Intelligence are almost always based on Artificial Neural Networks. However, another route towards Parallel Distributed Processing was introduced. This was inspired by the intelligence displayed by single-celled creatures called Protoctists (Protists). A new system based on networks of interacting proteins was introduced. These networks were tested in pattern-recognition and control tasks in the time-domain and proved more flexible than most neuron models. They were trained using a Genetic Algorithm and a derived Backpropagation Algorithm. Termed “Artificial BioChemical Networks” (ABN) they have been presented as an alternative approach to connectionist systems.

Acknowledgements

I would like to thank my family and those friends whose assistance over the years has allowed me to reach this level of academia. My parents continued faith in education has encouraged me to attempt to improve my own knowledge.

There is a debt to my supervisor Dr Chris MacLeod which has accrued over the years and will require considerable effort to repay. I am also grateful for the assistance provided by my second supervisor Mr Grant Maxwell who has been an excellent source of advice and stability during this time.

Various additional people have greatly assisted in proof reading and expert advice, their corrections and alternative views have enriched my efforts. These are Miss Claire Jamieson, Dr Stuart Watt, and Prof. Adam McBride (of Strathclyde University).

I would like to thank the other researchers who have permitted me to include details of their work in this text; Dr Christopher MacLeod, Dr David McMinn, Dr Sethuraman Muthuraman, and Mrs Ann B Reddipogu.

A consideration should also be shown to my examiners; Dr Tony Miller of The Robert Gordon University, Professor C Tim Spracklen of the University of Aberdeen and Professor Philippe De Wilde of Heriot-Watt University.

Finally I am grateful to those staff of the Schools of Computing and Engineering who have provided encouragement and support.

N F Capanni 31:5:06

Table of Contents

Chapter 3 - Universality and Generalisation in the Spatial Domain

Chapter 4 - Power Series

Chapter 5 - Power Series Neuron

Chapter 6 - Artificial BioChemical Networks

Chapter 7 - Artificial BioChemical Networks - Design and Function

Chapter 8 - Artificial BioChemical Networks - Experiments and Results

Chapter 9 - Future Research

Chapter 1
Introduction to the Thesis

1.1 Introduction to the Chapter

This chapter sets out the problems addressed by the project and explains their presentation in this thesis. Firstly, the aim and objectives of the research are outlined and discussed, next the original ideas that were discovered using the project are detailed. Finally, the remaining chapters are listed and summarised.

1.2 The Nature of the Problem

This project contributes research into the building blocks of Artificial Neural Networks (ANNs) - the Artificial Neurons (ANs) or Logic Units (LUs).

Currently, most research into the applications of Artificial Neural Networks falls into three areas; pattern recognition, control and signal analysis. As most development of neural networks is therefore focused on improvements in these abilities, it has resulted in the use of one of several types of standard models.

The three main standard models are the McCulloch-Pitts (MP) used in Multi-Layer Perceptrons (MLPs), the Radial Basis unit in networks of the same name and the Spiking neurons of computational neuroscience. There are several lesser known models such as Sigma-PI and Adeline units.

However such a focused, application led, approach is not suitable for all neural networks. Consider, for example, evolutionary networks being used in robot control. In this case, the networks need to be able to evolve to deal with different functions – for example, visual processing or actuator control.

If a single type of unit is to perform these tasks, it must be flexible enough to evolve into a form suitable for all of them. This is what is meant by the term “Universal Unit” in the

context of this project. The object of the research presented here is to investigate the functionality necessary to achieve this.

In other words, the project aims to investigate a neural model suitable for use in artificial neural systems, where information may come from many data domains (both inputs and outputs). Such systems include (but are not limited to) evolutionary networks and applications such as robotics.

1.3 Universality and Generalisation of Artificial Neuron Function

The starting point for this research was a survey of artificial biological nervous systems. When considering this, it should be noted that the nervous systems of animals are modular – that is they are made up of several smaller networks, operating together as an ensemble. Therefore, any investigation of functionality must bear this in mind.

Biological neurons exist in a variety of shapes, sizes and functions. They have also evolved specialisation, on an operation level, depending on their role in the part of the nervous system in which they reside. Disentangling such complexity is difficult and therefore a more systematic, theoretical and mathematical approach was adopted in the project. To this end, the research was split up into two sections.

The first investigation was into units that were designed to operate in a static abstract data space. This visualisation approach was pioneered by Minsky and Papert [1969] who represented the output from a McCulloch-Pitts based "Perceptron" type unit as a straight line (sometimes known as a "Linear Separator") in such a data space. In the research presented here, this and subsequent work is contrasted and extended.

The second investigation was into units that code information in the time domain (that is, by means of a varying waveform). An example of this is the biological neuron itself, which encodes its level of stimulation as the period of a pulsing waveform. Such units are important because system outputs (for example, the actuators of a legged robot) often need a time varying waveform to drive them (because the sequence of events in the system is important).

A new theoretical framework was constructed around such Time Domain Units in order to establish their limitations in a similar way to the static abstract data space of the non-time varying neurons. Additionally, this investigation led to a new connectionist model, based on the dynamics of biochemistry rather than neurons.

Finally, once these investigation were over, the units developed were integrated into a simple evolutionary modular system (based on a legged robot) in order to test their effectiveness.

1.4 Aims and Objectives

The overall aim of the research in this thesis is to investigate new "Integrated Neural Models" that fulfils the evolvable functionality requirements discovered through previous research in Artificial Neural Networks.

The primary aim is to investigate the School's ideas of a "generalised neuron" based on the Taylor Series (TS) [MacLeod et al., 2001] and to use this model to build a more controllable neuron using Evolutionary Techniques. The system will be tested against standard models to find how its generalisation abilities compared. Then, the unit functionality will be expanded to incorporate time domain data. The goal of these stages is to produce a neuron that can act as a building block for the next stage of the project.

The neural model investigated in the previous stages is to be integrated into routing and learning algorithms to produce a working system. However, as explained below, this aim altered during the course of the research and lead to the development of a new approach to connectionism termed Artificial BioChemical Networks (ABNs).

To accomplish these aims, the following objectives were set out at the beginning of the project.

To review the literature on the subject of generalised Artificial Neural Networks.

A literature search into generalised Artificial Neural Networks will be undertaken. This constitutes a portion of the background research of the project. Initially concentration is placed on alternative neuron models and architectures. The field will then broaden to include methods from associated fields in Artificial Intelligence. Both mathematical and biological approaches are examined. The literature search will continue for the duration of the project, and is included in each chapter as appropriate.

To review the biological relationship of the work, paying particular attention to the cellular, embryological and evolutionary aspects.

Textbooks, documentaries, papers and web sites on cell biology, genetics, zoology, nervous systems and evolution will be examined as directed. These will placed in context with the appropriate research material obtained from the literature search. Such material will concentrate on biologically inspired Artificial Intelligence implementations. This background material is covered in Chapters 2, 3 and 6.

To develop an appropriate generalised neural model, based on the above, which can assume any function (in combination with a genetic algorithm). It is anticipated that this will rely on the polynomial models in which background research has been completed.

The purpose of this section is to produce an Artificial Neuron that will be flexible enough to solve problems in the static mapped space that a single MP neuron cannot solve. This will be accomplished by the problem being examined from a mathematical viewpoint and implementing a solution based on the Taylor Polynomial. The neuron is reported as a "Universal Neuron" in this thesis, with the requirements for it being identified in Chapter 2 and its capabilities being discussed in Chapter 3. Research into such a neuron is presented in Chapters 4 and 5.

To extend the function of the above to time-domain behaviour.

The Artificial Neuron produced will be extended to the time domain so that it can produce time varying behaviour such as waveforms. In previous projects, McMinn [2002] and Muthuraman [2005], have shown that such behaviour was essential for controlling systems like robot actuators. This also has a biological basis as such oscillators are known to exist in all nervous systems. These investigation and results are presented in Chapters 5,6,7 and 8.

To compare these results with published and standard data.

The results will be compared with standard ANN and data. A review is included in Chapter 5.

To integrate these models into a complete neural system.

The neuron models and the placement algorithms will be integrated into a single neural system, as described in Chapter 8. This system developed under investigation to become of a biochemical rather than neural basis.

To apply this system to a standard problem such as the evolutionary walking robots which exist within the School.

The integrated system will be used to recognise artificial visual stimuli, produce control signals as a response to these and convert the control signals into a walking gait based on the simulated robot models used by the other research projects in the group. This work is described in Chapter 8.

To compare these results with previously published material.

These results were compared with other published results from within the group and the approaches of external researchers, as described in the relevant sections.

1.4.1 Alteration to Objectives During the Project

As with all PhD projects, results from the early periods informed the direction of the later research. In this case, the unexpected richness of the neuron models investigated inspired the author to concentrate on these aspects of the project and to scale down the planned research into learning algorithms. It was also decided that the routing and placement algorithms was sufficiently complex to merit their own project and this was completed by Muthuraman [2005].

To develop an adaptable learning algorithm, possibly based on the "neuron in a box" concept, which can add a learning influence to the above.

As mentioned above, a review of the research of the project's first eighteen months suggested that more effort should be expended on research into neural functionality. To

this end it was decided to undertake only preliminary work on learning. This research is outlined in Chapter 9.

To develop a placement and routing algorithm for use with the system described above. Again, as a result of early findings, as noted above, it was decided to make the initial research into routing and placement a PhD project in its own right. Muthuraman [2005] undertook this. The current author used the appropriate results of this work, developed it and incorporated it into the current thesis. This is contributes to the work of Chapter 8.

1.5 Novel Aspects of this Research

These are some of the aspects of this research that contribute to the originality of the thesis.

- A highly functional advance to the neuron model based on the Taylor Series approach, Chapters 4 and 5.
- A comprehensive theoretical and experimental consideration of the mapping abilities of neurons in the spatial-domain, Chapter 5.
- A new approach to connectionism based on the biochemistry of single celled organisms. This approach yielded insights into new time varying units and network paradigms. This work is presented in Chapters 6, 7 and 8.
- The integration of the models produced into modular connectionist networks. This is described in Chapter 8.
- A consideration and investigation of neural functionality in the context of robotic systems, presented in Chapters 7 and 8.
- A basis for further research into learning, modular networks and time-domain connectionism presented as part of the further work section in Chapter 9.

1.6 Thesis Structure

An overview of the remaining chapters is given below.

Chapter 2 - Review of Previous Work
The work undertaken by previous researchers within the group is described and the development and context of the current work is explained.

Chapter 3 - Universality and Generalisation in the Spatial Domain
In this chapter, the concept of universality is explained at the unit and network level.

Chapter 4 - Power Series
The mathematical basis of a new neural model is presented and compared to historically similar and alternative approaches.

Chapter 5 - Power Series Neuron
This chapter demonstrates the implementation of the models developed in the previous chapter.

Chapter 6 - Artificial BioChemical Networks
The chapter examines the environmental intelligence as expressed in single celled organisms.

Chapter 7 - Artificial BioChemical Networks - Design and Function
The chapter extends the research to time varying systems and suggests a new model based on biochemical pathways. This Connectionist model is implemented, compared with other models and its limitations explored. The new unit is integrated into a modular network.

Chapter 8 - Artificial BioChemical Networks - Experiments and Results
Both new approaches are combined to produce an artificial node that is universal in both the spatial and time domain, as defined in the previous chapters.

Chapter 9 - Further Work

Suggestions are made for further work. These include improvements and extensions to the work described in this thesis, as well as its combination with the other work from the research group.

Chapter 10 - Conclusions

The main objectives of this research are revisited and critically appraised and the original contribution of the work is discussed.

Published papers, extra results and reports produced during this research are included in the appendices.

Chapter 2
Review of Previous Work

2.1 Introduction to the Chapter

The Artificial Neural Networks group is based in the School of Engineering at the Robert Gordon University. It was formed by MacLeod and Maxwell in 1994. At the time of writing (October 2005), in the main research area alone, the group had published 16 papers externally, 15 MSc, 1 MPhil and 3 PhD theses as well as contributing towards BSc and BEng honours projects, and various press and magazine articles.

Since its establishment, the group's main interests have been in Evolutionary Artificial Neural Networks (EANNs). The ultimate purpose of the research is to achieve a viable process by which real artificial intelligence can be instigated. Advancement occurs in steps, not leaps and the group is working towards significantly improving Artificial Neural Networks' real world functionality as a means toward the greater purpose. Possibly due to the strong engineering element in the group, the research has mainly used legged robots as test beds.

Membership of the group has varied with staff and student progression. A core composition of four full-time members of staff with a full academic workload and three research students is indicative of the general size. The early work of the group was broadly supported by Eident Ltd who contributed towards the first two PhDs, those of MacLeod and McMinn. The later research has operated without external support.

This chapter shows the logical development of research from MacLeod [1999] through McMinn [2002] and Reddipogu [2002] to Muthuraman [2005], in conjunction with this thesis.

2.2 Single String Evolutionary Techniques

The first thesis from the group by MacLeod [1999] sets out the initial concepts that have been explored through later projects. These concepts focus on fundamental problems in ANNs that have restricted their abilities. One of these restrictions is that artificial networks are frequently highly designed. This limits their functionality and they are often highly or fully-connected, small and highly task specific. The discoveries of subsequent researchers support the observations in the paper "*Evolution and Devolved Action*" [MacLeod et al., 2002] and this author's basic assertion that:

> "*The greater the dependence on outside design, the more specific, inflexible or restricted the network functionality will be.*"

MacLeod's major contribution was his Single String Evolutionary Technique, termed "Incremental Evolution". This concentrated on the optimisation of ANN topologies through their synthesis by Embryological and Evolutionary Algorithms (EAs).

The importance of modularity was also discussed and a proposal for an artificial nervous system for an animal-like robot called an Animat [Wilson, 1991] was made in the thesis as a test bed for the development of modular networks. The significance of the failure of current ANNs to address time series modelling was also observed.

Macleod's second contribution was a clear list of problems or limitations of current ANNs. The investigation of these problems, their solutions and that of subsequently uncovered challenges, is the basis for all other research in the group.

2.2.1 Embryological Algorithm and Incremental Evolution

The terms "Embryological Algorithm" and "Incremental Evolution" are to an extent interchangeable. The latter term is that which Macleod used and was also referred to as "Incremental Growth". However, different researchers use alternative terminology and a proposal on this is given later in this chapter.

As the author's research group applies it, Incremental Evolution operates by increasing the functionality of an individual through the addition of component parts. Layers or modules of components are added onto a simpler but functional model and evolve together with its ability to sense and respond to its environment. It is akin to foetal development, where the embryo increases in complexity through a recognisable series of stages which mimic those of its evolutionary ancestry.

The embryological development of a fish, chick, pig, rabbit and human at different stages is shown in figure 2.1. Note the stage by stage similarities between different animals that grow into very different individuals. The similarity goes deeper than physical resemblances. The biochemistry and physiology of the organism may also have similarities.

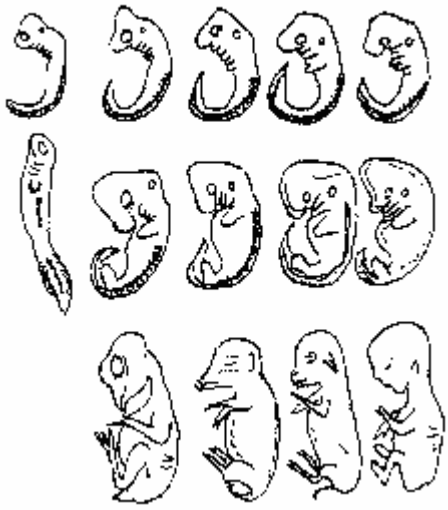


Figure 2.1 - After diagram in: 'The penguin book of the Natural World", edited Martin et al., [1976],

The parity between embryology and evolutionary development is the inspiration behind Incremental Growth. Existing EAs do not add new components to the structure of previous individuals in an incremental manner. Instead, they create a new population of more complex but completely re-wired individuals. This is a major difference between this and previous methods and has important ramifications with respect to cost and functionality. The exact nature of these are discussed later.

The implication of embryological development is that a newly evolved individual is not completely re-wired after the previous stage, rather that the new additions are layered around the older ones. This is particularly apparent in the nervous system as shown by MacLean's [1990] Model of the Brain, figure 2.2.

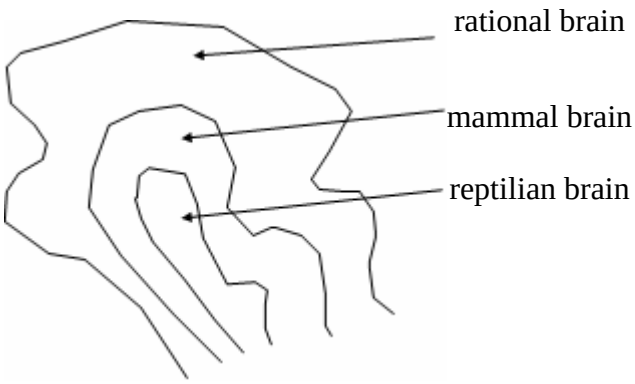


Figure 2.2 – MacLean's theory of brain organisation

In summary, MacLeod's Incremental Evolution allows an ANN to grow from a simple to a complex form, until it is able to perform its intended function.

2.2.2 Growth Strategies

Macleod presented six ways in which an Artificial Neural Network could increase in functionality, to become capable of achieving a solution to a set problem. These "Growth Strategies" are:

1. Change the number of neurons.
   Increase or decrease the number of neurons in a layer.
2. Change the connectivity.
   The number of active weights of the network may be altered.
3. Asymmetry.
   More connectivity may be provided in parts of the network.

4. Horizontal connections.

   Synchronous networks may introduce connections between neurons in the same layer.

5. Skipping layers.

   A connection may omit the immediate subsequent layer and connect to one deeper into the network.

6. Feedback.

   Feedback may be allowed to any previous layer.

MacLeod applied these growth strategies successfully to demonstrate solutions to pattern recognition problems. In doing this, four limitations were observed.

1. The whole network is retrained after each alteration to its topology.

   MacLeod discusses this – and its significance is presented later in this chapter.

2. The network architecture is highly structured and simple.
3. The algorithm was only applied to simple tasks and would be more useful if applied to other applications as well as pattern recognition.

   MacLeod proposed further development with "A framework for evolution of an Animat Nervous System" [MacLeod et al., 1998].

4. Only McCulloch-Pitts neurons were implemented.

The research that followed on from MacLeod's work strove to overcome these limitations. McMinn and Reddipogu developed the training with respect to topology alterations and expanded the complexity and flexibility of the architecture in significantly different ways. McMinn went on to advance the complexity of the algorithm and develop new neuron models. Muthuraman presented a complete solution to retraining after topology alterations and to the limitation of structure, both of which can be combined with work by this author. McMinn's advances on applications were superseded by modular implementation by Muthuraman and this author. Finally, Muthuraman produced more flexible, elegant models and showed the importance of this.

2.2.3 Incremental Training

There is a quandary with the training of many ANNs which is applicable to several standard types of network and those based on them, including Multi Layer Perceptrons and their associated feed-forward networks as well as recurrent networks. Once such an ANN has been trained, it cannot alter its abilities, e.g. recognise additional patterns, without undergoing retraining with the new data set. This erases all the previous knowledge. Additionally, if the network is to be retrained after it has increased in complexity, then there is an increase in training cost. Such was the difficulty with the retraining requirements that Grossberg [1976] introduced Adaptive Resonance Theory (ART) as an alternative method. ART is a useful approach but is very limited in how it operates, mainly by increasing network size when a new memory is required. Just how expensive these retraining requirements actually are was observed by Muthuraman and is presented later in this chapter.

The solution proposed by MacLeod is to train the initial ANN, then to allow it to undergo Incremental Growth. Further training is only applied to the newly added parts of the network. Proving the viability of this became the major objective of Muthuraman's research.

2.2.3.1 A Proposal on Terminology

Terminology is introduced as research uncovers innovations. Often these are shared across different fields, occur in different contexts or are applied to concurrent discoveries, so that to the reader they may have different meanings. As it is applied in the context of this thesis, Macleod's "Incremental Growth" refers to the addition of modules to an ANN or comparative units to the sensory, control or output structures of a robot. Macleod's "Incremental Training" refers to the training of the newly added modules while leaving previously trained modules unaltered. If both these Incremental Strategies are applied as parts of the same algorithm then that algorithm is called "Incremental Evolution".

2.2.4 A Framework for Evolution of an Animat Nervous System

In his thesis, MacLeod explored the importance of the unconstrained environment, non-specific problems and previous work on modular networks. He proposed work on the animal-like robots called Animats, from MacLeod [1999] as a test-bed for Modular Artificial Neural Networks (MANNs) which could be exposed to an unconstrained environment. This proposal formed the basis for the research of McMinn and Reddipogu. MacLeod's discussions on MANN synthesis and functionality was the foundation of the subsequent modular work of McMinn, Reddipogu and Muthuraman.

2.2.5 The Plasticity – Stability Dilemma

It should be mentioned that with the limited understanding of the operation of memory at a cellular and sub-cellular level, in Biological Neural Networks (BNNs), any training algorithm must overcome a lack of biological inspiration. These solutions rely on increasing the number of units that compose the system and BNNs do not seem to operate in this way.

The previously mentioned difficulties of standard Artificial Neural Network models to retain their capabilities during the acquisition of new ones presents a dilemma. Most ANNs initialise in an untrained state. They are considered trained when they have reached a level of usefulness expressed by an arbitrarily low value of an error function. To achieve this stability (the ability of a network to retain trained patterns) they sacrifice their plasticity (the ability of a network to learn new patterns). This is expressed as the Plasticity-Stability dilemma [Wasserman et al., 1989].

Muthuraman's work on modularity may present methods by which this dilemma may be overcome without specifically designing a training algorithm to counter it.

2.2.5.1 A Note on Modularity

The development in the previous sections relies on modular networks. These are not a new concept in Artificial Neural Networks. Their importance is widely understood and they are a popular research subject, this is well reported by Azam [2000]. However, just like fully

connected global neural networks, modular networks are often application specific and on examination their synthesis is designed and they are inflexible , as shown by McMinn [2002].

From his work on modularity and functionality, MacLeod identifies the three growth strategies of size, shape and configuration. These are investigated by Muthuraman [2005] who adapts and develops these into his "Principles of Modularity".

2.2.6 Time Series Modelling

The inability of many current ANNs to operate with time varying data is observed by MacLeod. He examines attempts to solve the time-series modelling problem and suggests how this may be investigated. Further, when the unconstrained environment expands to include time-dependent data as well as spatial data then the magnitude of the non-specific problem is greatly increased. There have been many previous attempts to address this problem but they generally rely on complex models such as spiking neurons [Maas and Bishop, 1999].

Biological Neural Networks have evolved in a time-dependent environment and so the inclusion of time-series data in their processing ability has been natural. Most current Artificial Neural Networks regard data as fixed and spatially observable. Even time series data is often sampled or mapped into a spatial domain before it is presented to the network [Bishop 1995].

An investigation into the time domain response of neurons became a major component of this thesis to allow the possibility of real time autonomous robotics.

2.3 Animat Nervous Systems

The Animat Nervous System (ANS) model suggested by MacLeod was proposed as an initial concept. It is hierarchical and modular. It is therefore an incomplete solution to non-specific problems as it does not operate in an unconstrained environment. It is, however, a significant step towards this. A fully evolvable modular system was not the initial intention of the project but was proposed and developed during work on the Animat.

The Animat model separates out component modules into hierarchical parts which each have their own systems to control. The interaction of these modules combines the solutions to separate problems and allows the Animat to function.

McMinn performed the work on the lower layers of this model. These produced the reflexes and cyclic patterns the Animat required for mobility. Reddipogu worked on the upper layers. These represent the sensory input and processing, which corresponded, in this case, to visual stimulation.

2.4 Using Evolutionary Artificial Neural Networks to Design Hierarchal Animat Nervous System - Lower Layers

McMinn's model is shown in figure 2.3. Multiple modules can exist in certain layers; these are marked with an asterisk. The hierarchical structure is evident.

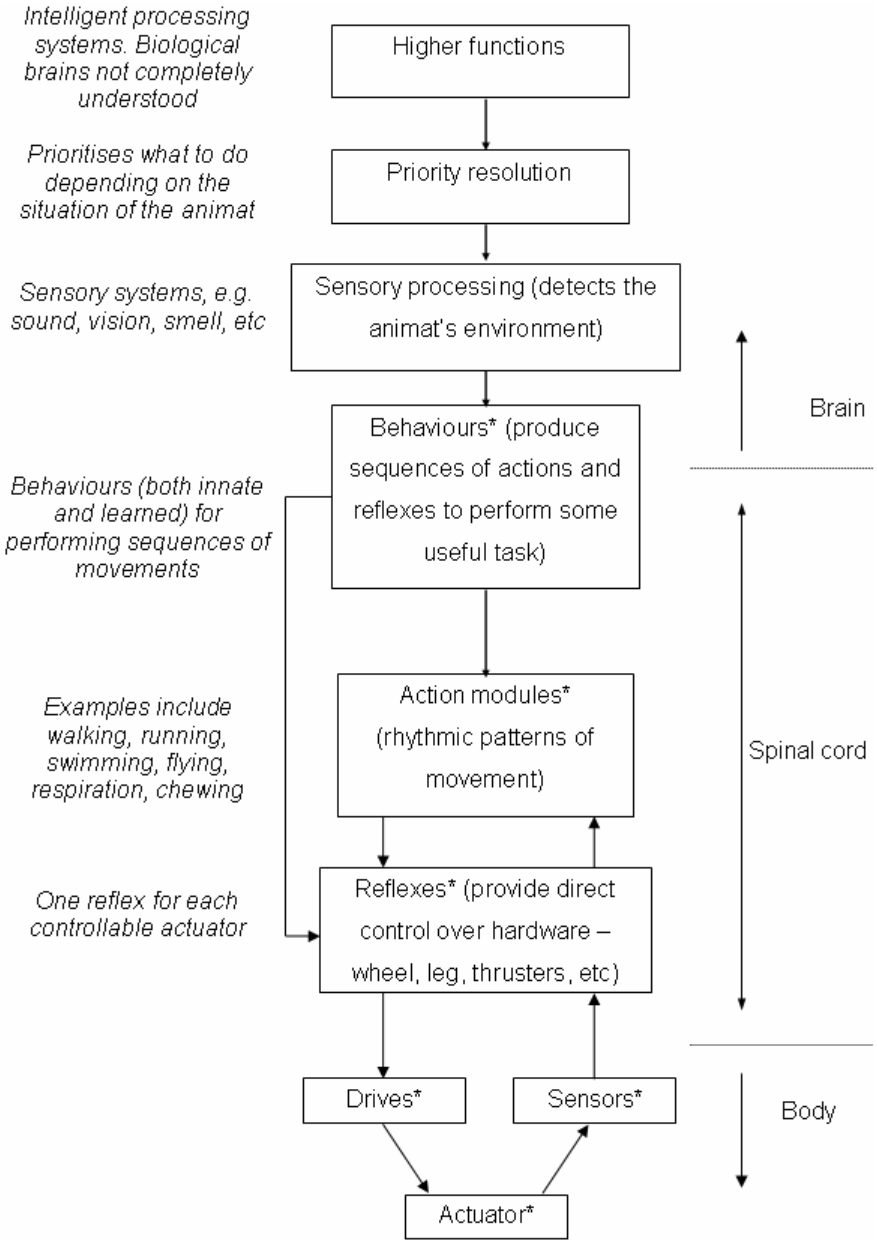


Figure 2.3 - McMinn's Artificial Nervous System

Reproduced by permission of McMinn [2002]

A detailed account of the operation of this structure will not be given here, as it is not directly relevant to the current work and was superseded by the modular work of Muthuraman. It is sufficient to report that McMinn used the model successfully. He implemented EANNs which incorporated Central Pattern Generators (CPGs) as the action layer and artificial reflexes in the reflex layer [McMinn, 2002]. The CPGs effected appropriate walking gaits for the Animat. McMinn successfully evolved both biped and quadruped gaits. The reflexes controlled the position of an actuator in a simulation of a DC electric motor.

To develop the reflexes, the neuron model used was the McCulloch-Pitts with a sigmoid (logistic) transfer function. Three main classical Evolutionary Algorithms: Evolutionary Programming, Evolutionary Strategy and Genetic Algorithms (GAs) were applied to the synthesis of simple feed-forward and recurrent ANNs. These provided good solutions.

McMinn constructed a new neural model to generate the specific timings required for CPGs. The McCulloch-Pitts neurons initially used did not have time-domain behaviour and so McMinn built in rigid time parameters. The intra-modular topology for the CPGs was evolved; the topology of the inter-module connections was designed.

The conclusion of McMinn's work was the combination of the reflexes with the CPGs. The reflexes require a continuous input (from their McCulloch-Pitts heritage) while the CPGs produce a pulsed time-domain output. McMinn therefore added a Leaky Integrator (LI) as an interface between the modules.

McMinn also included an alternative investigation using the CPGs, operating as oscillators. Biological neural oscillators [Lansner et al., 1998] are known to exist, so this was an appropriate investigation. The oscillating output was produced in response to a specific input. Pattern generators received this oscillating signal and produced the appropriate quadruped gaits of gallop, trot, pronk, and walk. McMinn concluded that by making the CPGs structures more modular the evolutionary process was simplified. This investigation provided useful material for Muthuraman's research.

The limitations of McMinn's Animat are that the individual modules do not grow but are fixed in size. These are placed within the ANN and are trained individually with independent fitness functions. Additionally, the unit functionality is fixed and the different

types of neurons are designed for specific tasks. This is a top down approach to creating an artificial system, and requires designer knowledge of the modules required. The model is an extremely useful model for robots, Remotely Operated Vehicles (ROVs), Autonomously Operated Vehicles (AOVs) etc., which have a specific design and functionality and operate in a known environment. For further explanation of this section see [McMinn 2002]

2.5 Evolution of Functions within the Animat Nervous System - Upper Layers

Reddipogu's work has strongly biological influences. After considering several biological vision systems, she researched the visual system of the toad, due to the structural similarity between toad and human Central Nervous Systems (CNS) and that it was one of the few vertebrate systems thoroughly researched by biologists, [Reddipogu, 2002].

When humans see an object such as a glass, they are able to identify it quickly. However, glasses come in many different shapes, sizes and colours. Humans can still identify the classification of the object even if they have never seen that exact type before. Neural networks have difficulty in doing this. If presented with a different object from the same domain there is no guarantee that the network will "work it out". Even if the object is known, it can be presented in a different orientation so that it is no longer recognised. Other researchers such as Reid [1989] have worked on the problems of distinguishing objects when presented in untrained size or translational positions. There has been much less work done on distinguishing between objects which are trained sequentially but presented simultaneously, such as a network being presented with two examples of a glass at the same time. Add to this the requirement for differentiation, size and translation capabilities and this is quite a challenging problem.

A novel visual system, based on the differentiation between prey and predator, was constructed. This is of fundamental importance to the unconstrained environment as all previous (simple type) ANNs cannot differentiate between two individually trained stimuli if they are presented with both simultaneously [Reddipogu, 2002].

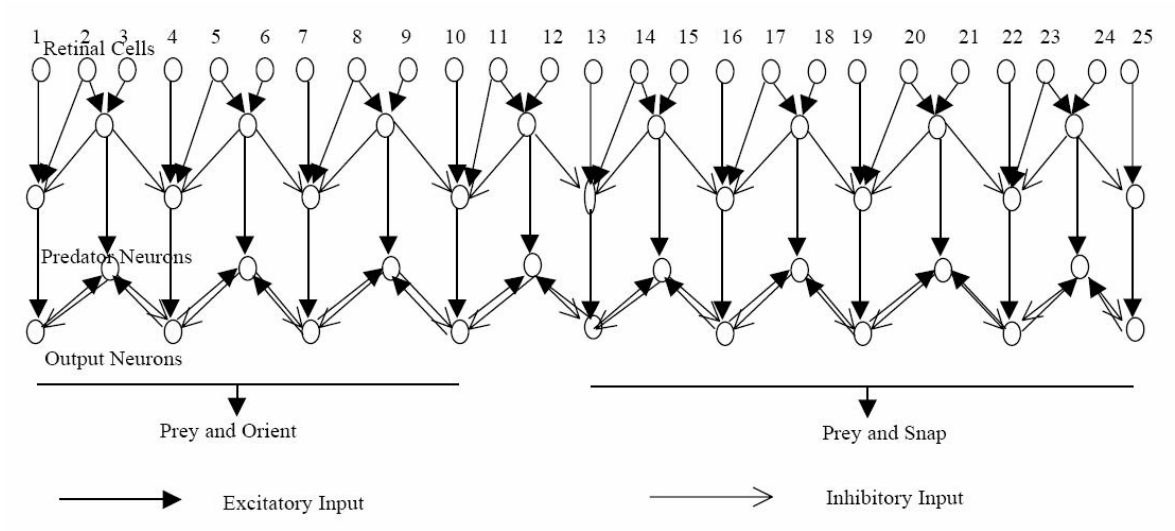


Figure 2.4 - Reddipogu's Artificial Vision System,
reproduced by permission of Reddipogu [2002]

The above diagram, figure 2.4, shows the implemented modular assembly.

The system is based on a modified biological neural circuit proposed by Ewert [1987]. The components are McCulloch-Pitts neurons with sigmoid (logistic) outputs. Reddipogu made use of an Evolutionary Algorithm employing Reinforcement Learning (EARL) to train the system.

A robotic visual system was developed from the network's ability to recognise combinations of patterns trained separately. It was suspected that the modularity of the system gave rise to these abilities. Useful lessons on modular placement were learned from this and implemented by Muthuraman [2005]. For further explanation of this section see [Reddipogu, 2002]

2.5.1 Summary of Animat Investigation

McMinn and Reddipogu produced interesting results in their investigation of the effects of evolutionary modularity on the functionality of Artificial Neural Networks. There are two other main areas noted by MacLeod that were, at the time, still to be incorporated.

Firstly, their systems had fixed modules that are placed in a hierarchical order based on functionality. The systems do not grow and the separate modules are evolved individually.

This means the Incremental Growth introduced by MacLeod was not incorporated into the model. This is attended to in Muthuraman's work.

Secondly, the unit functionality is limited and requires design based on what the parent modular functionality is. The use of different types of Artificial Neuron was sufficient for the purpose of this research but the authors were aware of and commented on the limitations. The significance of this was reinforced and clarified by Muthuraman. The solution to this problem is the subject of this thesis.

2.6 The Evolution of Modular Artificial Neural Networks

The modular networks used by McMinn and Reddipogu still required a fixed hierarchical design that represented a severe limitation in the unconstrained environment. The solution to this was essentially to combine the work of MacLeod, McMinn and Reddipogu. This was done by Muthuraman whose work was to focus on the evolution of the modular aspects of the system. The evolution should be both unconstrained and open-ended. It must therefore operate using an incremental adaptation to its environment. Such an approach replaces the constraints required by design with interaction of environmental parameters that are equivalent to evolution by natural selection in nature.

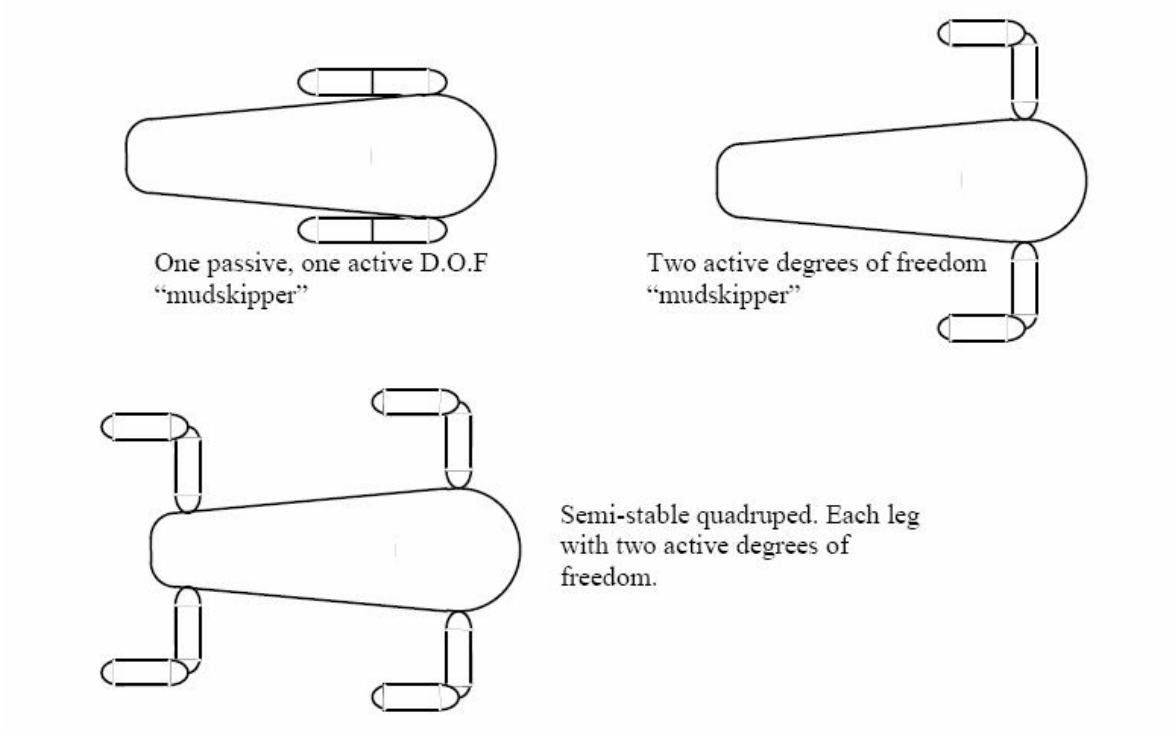


Figure 2.5 - Muthuraman's Robotic Body Plan Evolution,
reproduced by permission of Muthuraman [2005]

The system, invented by Muthuraman, began with a mechanically simple robot in a simple environment (actually as simple as possible). The robot used was akin to an artificial mud-skipper with simple single-jointed limbs. The environment is as complex as the robot's ability to sense it, so only factors that it can react to can be part of the fitness function. The environment and the robot's body plan become gradually more sophisticated. This incremental change is expressed in the robot being able to sense different, or more detailed,

environmental factors and react with more degrees of freedom. Its visual field also grows and accommodates more patterns.

To allow the robot to take advantage of greater inputs and outputs, previously evolved neural network modules are retained but not retrained. New modules are added and adapted to the whole structure. They undergo training and evolutionary change until the fitness function is satisfied. The process is an implementation of what MacLeod termed "Incremental Evolution".

The influence on this work is to make the focus and demonstration more robotic, which ties in with the previous work by the group. The robot's complexity can be varied as required. It can have input sensors, pattern generation and control networks, rather like the human body itself.

2.6.1 Principles of Modularity

There is a singular contribution from Muthuraman that is of fundamental importance to the incremental evolution of modular networks. Muthuraman successfully produced an algorithm that allows the MANNs to evolve in an unconstrained manner. Where previous work had made use of some intra-modular evolution, the algorithm allows all inter and intra-modular design to be evolved. This algorithm can be called the "Principles of Modularity". For a full explanation, see Muthuraman [2005].

The principles of modularity contain four components. The first is of primary importance to the aim of this thesis, and the others are also relevant.

1. Functionality

   The importance of the neuron's functionality was observed and shown; restricted functionality creates an over complexity of the modules in compensation. If the unit functionality is not enough, then there can be functions which cannot be evolved (even with larger networks). No solution to this was provided by Muthuraman as it is the basis of work by this author.

2. Wiring

   Modules must be permitted to break connections between neurons and modules. Even small residual weights on unwanted connections, that appear mathematically to have no influence, do not permit the system to operate correctly.

3. Placement

   The physical location of new modules within the system is essential to the success of the system. Once this is known, it can be incorporated as part of the EA.

4. Size

   There are a certain minimum number of neurons which each module must have in order to evolve correctly. Over-connection of neurons causes the information to be lost in the background noise of the network.

There is a complicated interaction between these principles. Module size and connections are strongly linked to unit functionality. The growth of the modules must be balanced with their training.

2.6.2 Complexity of Training

The increase in computational cost of retraining networks that increase in size to fulfil a functional requirement was known to MacLeod. Just how important this is and how debilitating to large systems was commented on by Muthuraman. It is demonstrated in the following example.

If a module begins as a fully connected structure of 10 neurons, each having only one connection to all other neurons and including a feedback to itself, then there are $10^2 = 100$ weights that require training.

If a new module adds another 10 fully connected neurons, then the whole system must be retrained with $20^2 = 400$ weights. So, to reach this stage the module has undergone, sequentially, training on 100 then 400 weights. This means 500 weights have undergone training.

If a small Artificial Neural Network is eventually evolved to a size equivalent to 100 modules, totalling 10^3 neurons, it will have 10^3 x 10^3 = 10^6, one million, weights that require simultaneous training.

The total number of connections trained to reach this stage, is the sum of a sequence of squares. (10,20,30,40,…1000).

The sum of n^2 where n is the set of integers from 1 to *x* is given as follows;

$$\sum_{n=1}^{n=x} n^2 = \frac{1}{3}x^3 + \frac{1}{2}x^2 + \frac{1}{6}x \qquad \text{equation 2.1}$$

In this case, *x* represents the effective number of modules; so, by the time the system is composed of 100 modules, it will have undergone training on the sum of n^2 for *x* = 100 modules. At 100 connections a module this means 33,835,000 total weights have been trained.

If the system complies with Incremental Growth, each new module is probably not fully connected to the previous modules. The entire system still requires to be retrained after each addition. The cost is the sum of the sequence (100 + 200 + 300 + 400 +… + 10000). This can be written as 100 x (1 + 2 + 3 + 4 + … + 100). This is expressed as the following

$$C \cdot \sum_{n=1}^{n=x} n = \frac{n}{2}\left(n_1 + n_x\right) \quad \text{equation 2.2}$$

This results in a total 505,000 trained connections by the same stage, 67 times less.

A fruit fly has around 10^3 times as many neurons as in this artificial example, about a million, with vastly more connections. A human has around 10^7, “ten million” times as many, with an average of $2x10^3$ to $5x10^3$ connections per neuron [Edelman, 1987]. If the brain was rewired at every evolutionary junction, every one of these connections would have to be re-evolved and re-trained. There is simply no possibility of a viable creature being produced in such a staggering search space.

If instead the training follows the principles of Incremental Evolution, then only the new module requires to be trained. Following the same example, with the same 100 connections in the initial module, once these are trained, they are left alone. On the addition of each new module, only the new connections are trained.

The total number of trained connections on the addition of the second module is 100 + 100 = 200.

If we expand this example to 100 modules, totalling 10^3 neurons, there are a resultant 10^4 connections. As only each new module is trained, each connection is only trained once, therefore the Total Incremental Training cost gives a total training cost of 100 x 100 = 10^4 trained connections.

In this example Incremental Evolution has trained 1000 connections, while Incremental Growth has 505,000 connections and Sequential Growth has trained 33,835,000 connections. The evolutionary advantage is 0.19801% and 0.00295% respectively of the required training. In each of the non-iterative evolutionary methods a progressively larger network has to be trained. In Artificial Neural Networks smaller networks train in fewer epochs.

The costs (in terms of training requirement of connections) for each new stage of training can be expressed as follows;

Sequential Growth	(AxB)
Incremental Growth	(A+B)
Incremental Evolution	(B)

where A equals the complexity of the previous stage and B is the complexity of the newly added stage.

2.6.3 Functionality and Modularity in Artificial Neural Networks

As previously stated, the importance of functionality at a neuron level was already known to the group even at the stage of MacLeod's initial work. McMinn and Reddipogu had implemented systems that were successful but relied on design at neuron level and again unit functionality caused problems in their implementations. Muthuraman designed his own neurons for specific tasks, and went on to establish the necessity for a highly adaptable neuron that had been missing from the previous work. There was now a need for a "Universal Neuron" that could take any position in any of the modules and evolve or train to fit the desired functionality.

2.7 Conclusions Drawn from Previous Work

Early research targeted the growth of simple networks to solve uncomplicated functions. This relied on the existing simple neuron structure and evolutionary techniques. The limitations of these was observed, particularly with regard to intensive design by the user, iterative training, modularity, time-domain performance and functionality. At an early stage, research became focused on robotic development as a method for simulating an unconstrained and challenging environment in which to develop advanced networks.

The ability of Artificial Neural Networks to perform a single well defined task but be poor at solving non-specific problems or multiple tasks, led to the group's work on communities of cooperative neural networks. These developed into the fixed hierarchical modules of the Animat nervous system.

An effort to overcome the restrictions of specific fixed operation and a designed hierarchy gave rise to the combination of growth and modularity algorithms. The resultant Principles of Modularity enabled the Evolutionary Algorithm for Modular Growth to be both open ended and environmentally unconstrained.

During this research the limitations of the existing neurons became apparent at almost every stage. These limitations resulted in specific neuron designs, dependent on the network function. Pursuing these issues of functionality would have been a distraction

from the research at hand - but there are two obvious limitations to the approach. Firstly, there were no time-domain capabilities for the simpler neurons and no evidence that a cluster could easily develop them. Secondly, increasing the number of neurons through clustering into functional groups increases the complexity of the search space as previously discussed, and makes a solution exponentially more difficult to find.

Work by other researchers has made clear that increasing the complexity of the system cannot on its own provide a solution to the non-specific problem, [Potter et al., 1995], [Thompson, 1996]. This is because the system does not laterally increase its abilities, and perform different tasks, simply by increasing its size in units or connections. Increases in the system's size are restricted in terms of universality and generalisation as discussed in this thesis.

This may seem to conflict with the accepted doctrine that an Artificial Neural Network is a universal mapping function from problem domain to solutions space [Hornik, 1989], [Barron, 1993] and [Sima and Orponen, 2003]. However, time-series problems are outside the domain of the simple neurons based on the McCulloch-Pitts model.

All of this points to the requirement for some type of "Universal Neuron" as defined earlier in this chapter. What is meant by "universal" will be discussed in the next chapter.

Chapter 3
Universality and Generalisation in the Spatial Domain

3.1 Introduction to the Chapter

The terms “Universal” and “Generalisation” are central to this project and occur frequently in Artificial Neural Network literature. This chapter explains what they mean in the context of this thesis and the research of the group.

Definition of Universality :

> *The ability of an Artificial Neural Network to approximate any functional mapping from its input (data) space to its output (solution) space.*

Definition of Generalisation :

> *The ability of an Artificial Neural Network to correctly classify new data which belongs to the same system as the training data with which it has been taught.*

3.2 Universality

A universal approximator is one that can perform an arbitrary mapping from one multi-dimensional data space to another. This is generally regarded as a mapping from an input space to a output space. A universal optimiser is one that performs well on a large set of optimisation problems within the same mapping from input space to output space. [Duch and Jankowski, 1997], [Briggs, 2005]. An example is shown in figure 3.1, where different representations of characters are recognised as one of 4 letters.

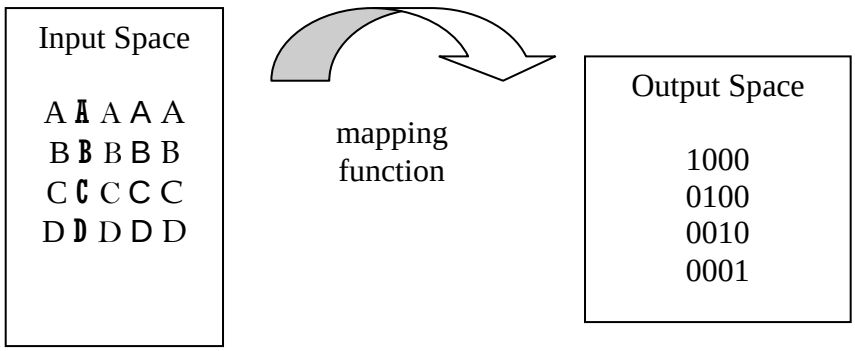


Figure 3.1 – Text recognition mapping function

One of the factors that brought most neural network research to a halt in the 1970s was the work by Minsky and Papert [1969]. Amongst other observations, they showed two major weaknesses with Perceptron universality.

Firstly, as shown in figure 3.2, a single neuron of the McCulloch-Pitts type, which accounted for almost all artificial neural units of the day, could not provide a solution to the XOR problem. The XOR problem is a two dimensional example of the parity-check problem and therefore easy to visualise, (see figure 3.3). It can be expanded into any number of dimensions and is known as the "d-bit parity problem".

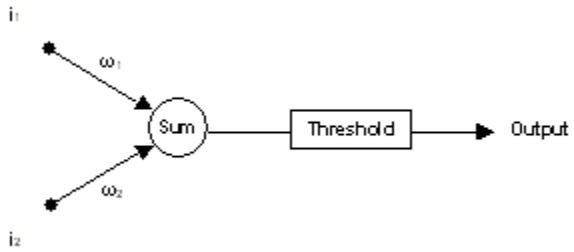
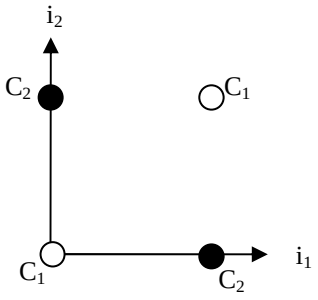


Figure 3.2 – McCulloch-Pitts Neuron

i_1	i_2	Desired Output	Class
0	0	0	1
0	1	1	2
1	0	1	2
1	1	0	1



A single linear decision boundary cannot classify the input vectors i_1 [(0,0),(1,1)] and i_2 [(0,1),(1,0)] correctly.

Figure 3.3 – Exclusive-or (XOR) problem

Secondly, they demonstrated that a simple two-layer perceptron was not capable of providing a useful function approximation outside a narrow class.

In hindsight, their work did ANN research a great service, because, through visualising the problems, they allowed a greater understanding of network limitations. This focused attention on solving these problems and a great deal of the subsequent work on universality is based on their observations or in providing solutions to them. The second weakness provided a solution to the first using Backpropagation training [Rumelhart, Hinton and Williams, 1986], [Werbos, 1974], [Parker, 1985] as shown in figure 3.4.

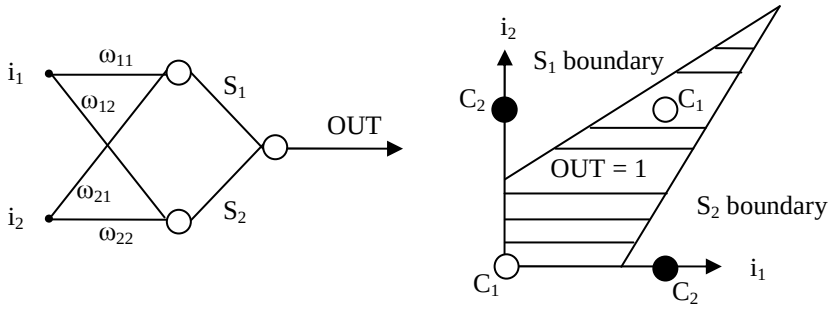


Figure 3.4 – solution to the parity-check problem

The solution, of course lies in the use of networks rather than individual neurons as shown in the diagram and it can be extended to any d-bit parity problem so that multi-layer networks are universal approximators [Hornik, 1989]. This extension has a mathematical basis as shown by Kolmogorov [1957], who disproved Hilbert's famous 13th conjecture. Flaws were pointed out and fixed by subsequent authors including Lorentz [1966] who extended Kolmogorov's work to show, in theory, that an approximation can be obtained for any multi-variate function by a compositional network of univariate functions. The specific function determines the accuracy of the approximation. Despite this, there are many limitations and the specific size of the model cannot be determined from the function itself. As Elder and Brown [1992] observed, "inducing such a model from sample data remains a great challenge".

Applications to neural networks have shown that Kolmogorov's theorem could be generalised for Multi-Layer Perceptrons (MLPs) or any multi-layer feed-forward neural

network, and so these could be considered to be universal approximators [De Figueiredo, 1980].

Hecht-Nielson noted that Kolmogorov's superposition can be interpreted as a three-layer neural network and any continuous function defined in (d) dimension unit hypercube could be implemented exactly by a three layer network of (2d+1) hidden units and with a suitable transfer function [Hecht-Nielsen, 1987].

A survey of universality [Tikk, Kóczy, and Gedeon, 2001] has shown that since then it has been proved that different types of neural network possessed the universal approximation property [Blum and Li, 1991], [Hornik, Stinchcombe and White, 1989], [Kurkov´a, 1992].

As noted above, the specific transfer function is of critical importance to a network's universality [Duch and Jankowski, 1997]. Extending the functionality of the unit can reduce the dimensional requirement of the network and it has been shown that a single hidden layer neural network can be a universal approximator [Hornik, Stinchcombe and White, 1989].

Single-layer neural networks with sigmoidal functions have been demonstrated as universal approximators. They can approximate an arbitrary continuous function, on a compact domain, with arbitrary precision, given sufficient number of neurons [Cybenko, 1989].

This can be extended to a continuous function, that show a single unit of the correct type can itself be a universal approximator [Capanni et al., 2003].

Polynomial networks (see next chapter) have been shown to have both universal approximation abilities and good generalisation [Nikolaev, 2003]. Nikolaev notes that problems in generalisation can, partially, be attributed to fixed network structure [Chang and Cheung, 1992] and that this cannot be countered by restricting the polynomial complexity or by their learning algorithms and therefore under-fitting or over-fitting occurs [Heywood and Noakes, 1996].

3.3 Generalisation

In many neural network applications generalisation is best explained by a pattern recognition example. Once a network has been trained to recognise specific patterns from a training set, it may be presented with patterns that were not members of that set. If the network has learned the *underlying* structure of the problem domain, then it should be able to correctly classify these new patterns. Such a network is said to have good generalisation. If the network cannot generalise, then it is simply performing a one-to-one mapping from the input space to the solution space as in figure 3.5a. This could be achieved far more simply with a lookup table or template match [Gurney, 1997a], [Bishop, 1995a].

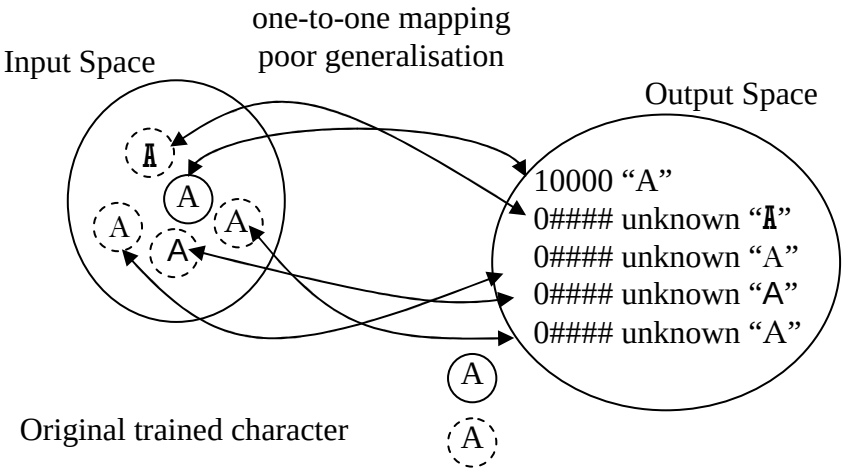


Figure 3.5a – A one-to-one mapping

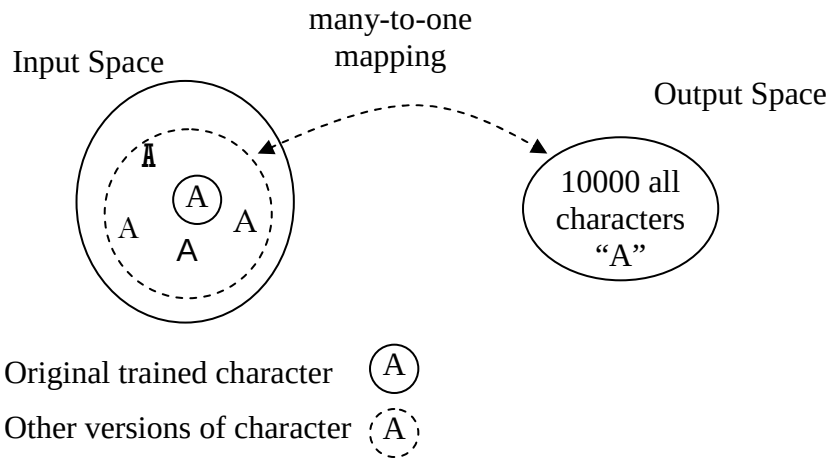


Figure 3.5b – A many-to-one mapping

Figure 3.5b shows the many-to-one mapping actually required. The new patterns or cases that the network operates on must belong to the same system as the training set [Elder and Brown, 1992]. Failure to generalise can be attributed to many different factors such as:

- The training set not being a true representation of the problem, perhaps because too few examples are used or they cluster in areas and do not contain all the factors that separate the cases.
- The network having too few neurons or weights, so that it may not be able to differentiate between all the factors and will under-generalise. This will be observable in training, as the network will have difficulty achieving (or fail to reach) an acceptable target error.
- The network having too many neurons or weights. In this case, it may fit the training set too well and suffer from what is termed over-training or over-fitting.
- If the search space is too big, the initial parameters may result in the network reaching a sub-optimal minimum before it can find a good solution or the global minimum.

3.3.1 Over-Fitting and Over-Training

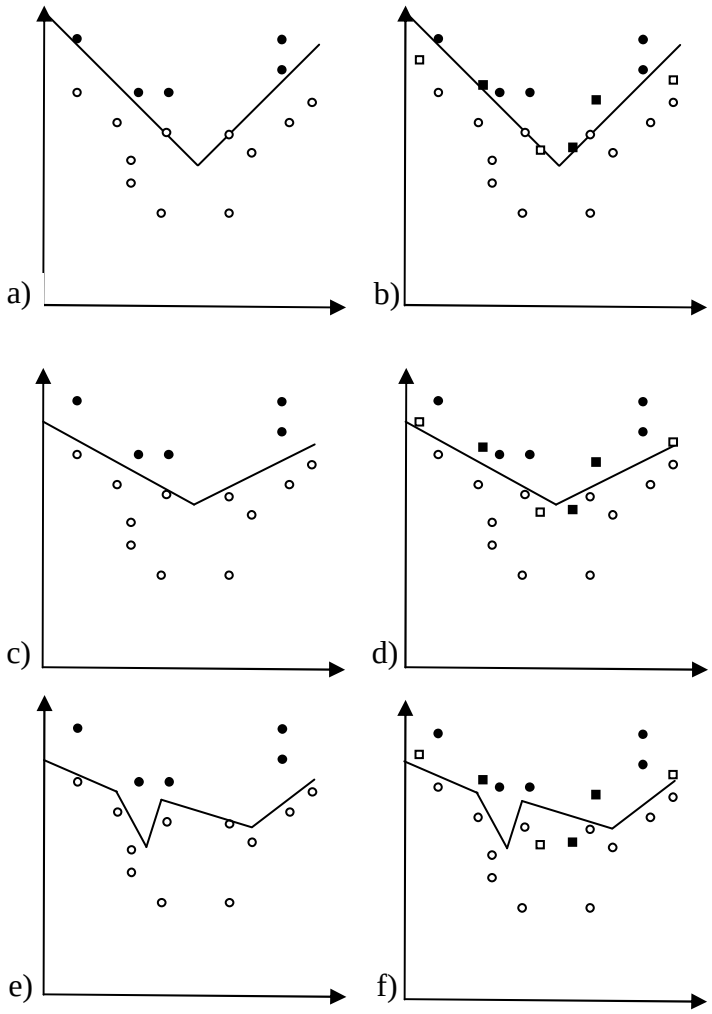


Figure 3.6 – Generalisation and over-training – modified from Gurney [1997b]

Consider figure 3.6. There are two classes distinguished by open and closed symbols. The training set is shown as circles and the untrained patterns are shown as squares.

In the first diagram-pair (a & b), the two lines represent the linear separation of a two hidden neuron MLP. Initially, it appears that there is poor training as two open circles lie directly on the linear separators. This will result in some residual error in training. The network is then tested with the untrained data, represented by the squares, and correctly classifies them. The network has therefore generalised well with the unseen data, capturing its essential characteristics in pattern space.

If the low error in training is not accepted and a lower error is sought by either continuing to train (altering the hyperplanes), as in the second pair (c & d) or adding additional separators, as in the third pair, figure (e & f) then a zero error can be found. However, once the unseen data is added neither of these techniques manage to correctly classify the new patterns. These networks have been over-trained and possess poor generalisation. They have over-fitted the decision surface to accommodate all the noise and specifics of the training data without learning the underlying trends [Gurney, 1997c], [Bishop, 1995b]. Networks that are too large for the problem domain are susceptible to learning without good generalisation [Chen, 1991] and can result in a multi-dimensional Lagrange interpolation [Steffensen, 1950] of the training data. This is expressed in very poor recognition ability when presented with new patterns. However, the extent of the performance is determined by the nuances of the pattern space.

Overtraining can be countered with testing called “cross-validation” [Gurney, 1997d] or “holdout-validation” [Pednault, 2004]. In these, the error during training is tested on a validation-set of untrained patterns that come from the same problem as the training-set. As shown in figure 3.7, the error tends to follow that of the training-set but remains slightly higher. As training continues, the training-set error will continue to decrease but at some point the validation-set error will begin to rise. This is the point when the network is starting to over-fit and is losing its ability to generalise.

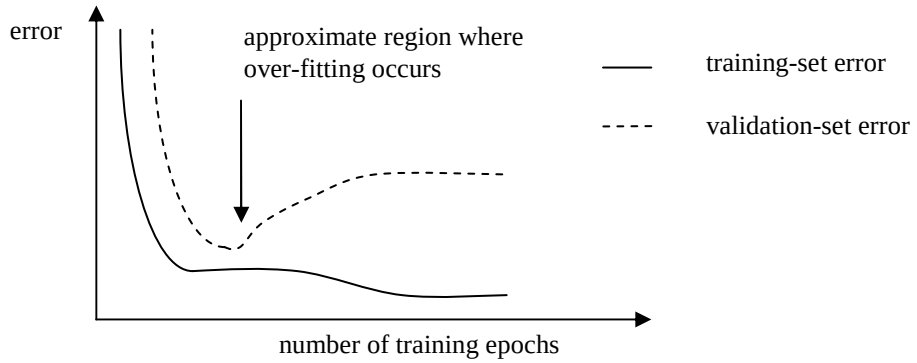


Figure 3.7 – Training and validation error – modified from [Gurney, 1997e]

3.4 Universality and Generalisation Trade-Off

From the previous section, it can be seen that a balance must be found between generalisation and universality. ANNs should be adaptable and therefore universal in their environment and this can be done by increasing their size. However, we also know that as the size increases, the network risks losing its generalisation abilities. So the solution to this is to combine the lessons learned about training and growth with an increased functionality of the basic unit, as in Chapter 4, that allows the use of fewer units in the ANN construction. However, not just any unit can be used; it should be an improvement over linear seperability but must have some constraints or the results may be polynomial over-fitting, (see Appendix D). The investigation of such functionality is the purpose of the next chapter.

Chapter 4
Power Series

4.1 Introduction to the Chapter

This chapter outlines a neural model, which has been designed to be flexible enough to assume most mathematical processes. It is particularly useful in evolutionary networks as it allows the network complexity to increase without adding neurons. The theory is presented in this chapter, this forms the base for the development of both time-series and non time-dependent applications as the next chapter shows. This work was originally published by Capanni [2003], see Appendix A.

4.2 Evolution by Devolved Action

The requirements that lead to the research covered in this chapter were introduced in the paper "*Evolution and Devolved Action*" by MacLeod et al., [2002]. This paper identified many of the problems with Artificial Intelligence (AI) development, among which was unit functionality.

4.2.1 Unit Functionality

All neural biological systems have similar neurons. However all do not have exactly the same unit functionality, even within one organism [Levitan and Kaczmarek, 2001]. Neurons have been categorised into general types, based on physical appearance, location in the organism and perceived function but the subtle differences go deeper than these broad categorisations and the operations of the more exotic types are not fully understood.

Also, as described in the last chapter, there are many different types of artificial unit, such as, Perceptrons, Radial basis units, Sigma-Pi units, etc. "*What is needed is an evolutionary system which can evolve any reasonable neural function*" [MacLeod et al., 2002].

This is the basis of the Power Series research (and through its expansion from the static to the time-domain, it also leads logically onto the Artificial BioChemical Networks presented later in this thesis).

4.3 Power Series

Power expansions belong to a grouping of infinite sequences and series where the n^{th} term is a function $u_n(x)$. The general power series is;

$$\sum a_n x^n = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n + \ldots \qquad \text{equation 4.1}$$

where the numbers a_n (n = 0,1,2,…) are constants independent of each other and of x. They are used extensively in mathematical physics, allowing descriptions of various phenomena including signals such as current and voltage, [Thomas and Finney, 1996a].

In a sequence $u_n(x) = cx^n$, or $u_n(x) = c(x-a)^n$, where a and c are non-zero constants, then the sequence converges to zero if $|x| < 1$ or $|(x-a)| < 1$, and converges to c if $x = 1$ or $(x-a) = 1$ and otherwise it diverges.

If it converges, the sum to infinity of a formal power series can be expressed as;

$$\sum_{n=0}^{\infty} a_n x^n = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n \qquad \text{equation 4.2}$$

These power series can be truncated to give polynomial approximations to standard elementary functions such as e^x and $\sin x$. The range over which these approximations are accurate is determined by the order of the polynomial used (and proximity to the radius of convergence).

It is a specific type of polynomial expansion derived from power series that is of interest in this thesis, namely the Taylor Polynomials.

4.4 Taylor and Maclaurin Series

While not every function may be represented by a power series, every[1] function f that is defined in a neighbourhood of $x = 0$ and has finite derivatives f', f'', … ,f^n at 0 generates polynomials $p_0(x)$, $p_1(x)$, … , $p_n(x)$ that approximate $f(x)$ successively more accurately for values of x near 0, [Thomas and Finney, 1996b].

For any non-negative integer k, the polynomial $p_k(x)$ can be taken to be the terms up to and including x^k in the power series shown in equation 4.2 to give;

$$p_k(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \ldots + a_kx^k \qquad \text{equation 4.3}$$

The coefficients $a_0, \ldots, a_k$ are determined as the derivatives of $f(x)$ at $x = 0$, (the point (0, $f(0)$)). Thus, the polynomials $p_0(x)$, $p_1(x)$, … , $p_k(x)$ expressed as in equation 4.3 pass through (0, $f(0)$). This is shown in equations 4.4(i to iv);

$$p_0(x) = a_0 \qquad \text{where} \quad a_0 = f(0) \qquad \text{equation 4.4i}$$

$$p_1(x) = a_0 + a_1x \qquad \text{where} \quad a_1 = f'(0) \qquad \text{equation 4.4ii}$$

$$p_2(x) = a_0 + a_1x + a_2x^2 \qquad \text{where} \quad a_2 = \frac{f''(0)}{2!} \qquad \text{equation 4.4iii}$$

$$\vdots$$

$$p_k(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \ldots + a_kx^k$$

$$\text{where} \quad a_k = \frac{f^k(0)}{k!} \qquad \text{equation 4.4iv}$$

Replacing the coefficients allows the expression to be re-written as shown;

$$p_k(x) = f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \ldots + \frac{f^k(0)}{k!}x^k \qquad \text{equation 4.5}$$

[1] Actually, there are some exceptions - certain pathologic functions such as the function defined piecewise as $f(x) = e^{-1/x^2}$ if $x \neq 0$ and $f(0) = 0$. All the derivatives of $f(x)$ are zero at $x = 0$. Therefore the Taylor series of

The specific Taylor series shown in equation 4.5 corresponds to expansion about $x = 0$ and is called the Maclaurin series, [Thomas and Finney, 1996c]. If an approximation is required near another point a, the powers can be re-written as $(x\text{-}a)$ which results in the following Taylor polynomial and series;

$$p_k(x) = a_0 + a_1(x-a) + a_2(x-a)^2 + a_3(x-a)^3 + \ldots + a_k(x-a)^k \qquad \text{equation 4.6}$$

$$p_k(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \ldots + \frac{f^k(a)}{k!}(x-a)^k$$

equation 4.7

This results in the approximation $p_k(x)$ about $x = a$ for any non-pathologic function $f(x)$, that has finite derivatives of all orders at a. As with the previous power series approximations, the Taylor approximation can be improved by increasing the order.

4.5 Relevance of Taylor Series

The paper "*Evolution by Devolved Action*" [MacLeod et al., 2002] set out a biological basis for exploring unit functionality. This was followed up by the current author Capanni [2003] where a new neural model was presented, based on the idea that a neural unit should be flexible enough to fulfil any differentiable mathematical function required of it. The model is a logical extension of the Perceptron and the first advances mentioned by Capanni in this paper were later developed to what is presented in this and the next chapter.

Any lower unit number universality of a Taylor Series network compared to MLP comes at a price, as polynomial over-fitting can develop, Appendix D. Taylor series networks are vulnerable to the Plasticity-Stability dilemma, explained below, through what Bishop [1995] calls the Bias-Variance trade-off.

$f(x)$ is zero even though the function $f(x)$ is not zero. So it is assumed that f is well approximated by its Taylor polynomials.

The Plasticity – Stability dilemma is as follows:

Plasticity:	The ability of a network to learn new patterns.
Stability:	The ability to retain previously trained patterns.
Dilemma:	A fixed topology network cannot learn new patterns without affecting the memory of old ones.

4.6 Linear vs. Non-Linear Separability

The previous chapter introduced linear separability and explained how this simple concept was the basis for Artificial Neural Network functionality. The advantages and limitations of this with respect to universality, generalisation and the famous parity-bit problem were explored. In figure 4.1 it is shown visually how a single, second order TS neuron can exactly map a non-linear boundary, whereas a McCulloch-Pitts Multi-Layer Perceptron requires two layers and several neurons to approximate the same boundary (it cannot match it exactly). A potential disadvantage of which, that will be explored later, is the danger of exact matching, resulting in over-fitting and loss of generalisation.

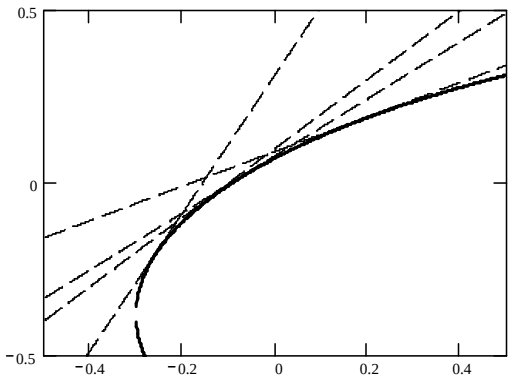


Figure 4.1 - Comparison of McCulloch-Pitts MLP to single 2nd order TS neuron

A non-linear decision boundary can only ever by approximated by a set of linear separators. While for generalisation purposes approximation may be desirable, a non-linear separator can either exactly match the boundary or, if the training algorithm includes an approximation element, provide such a level of approximation with a single unit of sufficient power order.

The universality of the TS neuron can be seen in that it reduces to a MP neuron as shown in equations 4.8c and 4.9a and explained below.

4.7 Model Solution

The majority of ANNs currently use a neuron developed from the original McCulloch-Pitts model, as shown in figure 4.2. Other types use a Euclidean difference formula $S = (x - w)$. Those that do not may still utilise the initial *(input x weight)* summation stage. The output from this stage usually undergoes a transformation using a threshold or squashing function. This function normalises the output, common examples being binary {0,1} or logistic sigmoid (0,1). Without the use of the normalising function, the activity of this neuron is given by;

$$S = \sum_{i=1}^{n} x_i w_i \qquad \text{equation 4.8a}$$

The variable S denotes the sum of each input x_i multiplied by the strength of its connection termed the weight w_i and is known as the "activation" of the neuron. In the absence of a normalising function the output of the neuron O is equal to S as shown in equation 4.8b.

$$O = \sum_{i=1}^{n} x_i w_i \qquad \text{equation 4.8b}$$

For a neuron of two inputs this can be given as the easily visualised equation 4.8c of a straight line with respect to the variables x_1, x_2.

$$O = x_1 w_1 + x_2 w_2 \qquad \text{equation 4.8c}$$

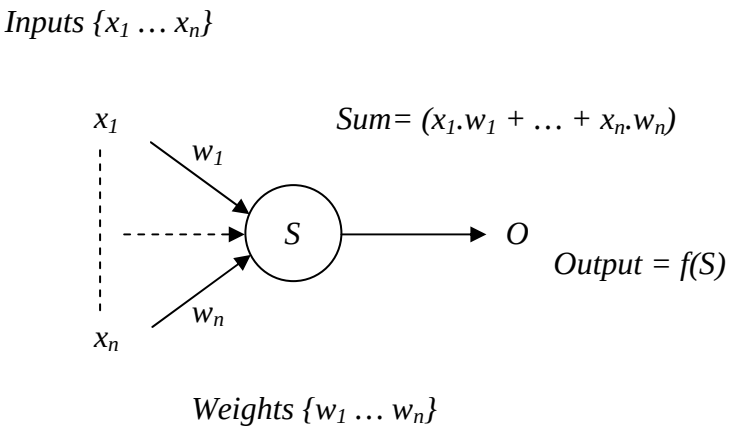


Figure 4.2 - McCulloch-Pitts neuron of *n* inputs

As discussed previously in this chapter continuous non-pathologic function can be modelled using an infinite power series as shown in equation 4.2, specifically a Taylor series as shown in equation 4.5. This can be implemented as a neuron using the output function shown in equations 4.9 and figure 4.3.

A TS neuron of two inputs with a second order expansion can be expressed as in equation 4.9a. The variables x_1, x_2 are independent of each other in real terms and in the mathematical sense. Therefore x_1, x_2 could be replaced by x, y.

$$O = x_1^1 w_{1,1} + x_2^1 w_{2,1} + x_1^2 w_{1,2} + x_2^2 w_{2,2} \qquad \text{equation 4.9a}$$

For a TS neuron of two inputs, an order p expansion can be expressed as in equation 4.9b.

$$O = x_1^1 w_{1,1} + x_2^1 w_{2,1} + x_1^2 w_{1,2} + x_2^2 w_{2,2} + \ldots + x_1^p w_{1,p} + x_2^p w_{2,p}$$

equation 4.9b

In its full expression, a TS neuron of n inputs expanded to an order of m can be expressed as equation 4.9c.

$$f(x_1, \ldots, x_n) = O = \sum_{p=1}^{m} \sum_{i=1}^{n} x_i^p w_{i,p} \qquad \text{equation 4.9c}$$

In all of equations 4.9, the derivatives of the time-series are replaced by the weights in a like for like manner, so that;

$$w_{i,p} = \frac{\partial f^{\,p}}{\partial x_i^{\,p}}(0) \cdot \frac{1}{p!} \qquad \text{for } p = k \text{ and } i \text{ is dependent on the input,}$$

replacing $\frac{f^{\,k}(0)}{k!}$ for $f(x)$ as the function is now multi-variate in $f(x_1,\ldots,x_n)$.

The term $f(0)$ can be replaced with the bias θ_i as in standard MP neuron operation to give equation 4.10a.

$$O = \sum_{p=1}^{m}\left(\sum_{i=1}^{n}\left(x_i^{\,p} w_{i,p} + \theta_i\right)\right) \qquad \text{equation 4.10a}$$

In an MP neuron, θ is a property of the neuron, not the connection. However, $f(0)$ is a constant and so the sum of all the θ_i can given as θ, a constant term without input, to give equation 4.10b.

$$O = \theta + \sum_{p=1}^{m}\sum_{i=1}^{n} x_i^{\,p} w_{i,p} \qquad \text{equation 4.10b}$$

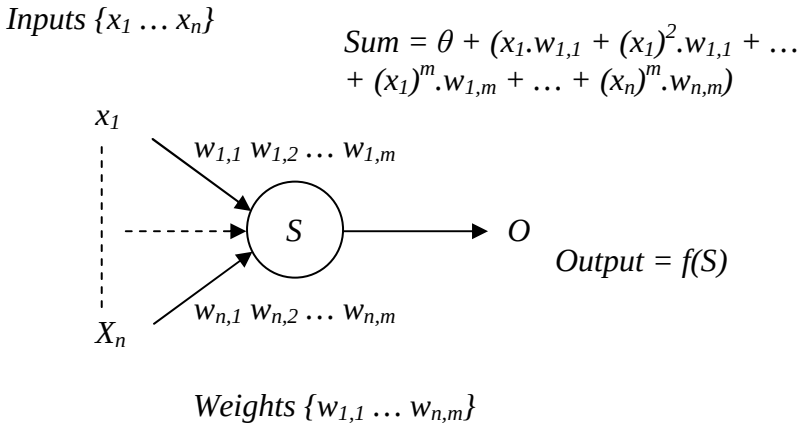


Figure 4.3 – Taylor series neuron of n inputs and order m

The factorial term $p!$ is theoretically absorbed by the weight term and as the values for x_i should be constrained within the range [-1,1] or [0,1] then the powers of x_i will not become

uncontrollably large. However, some account of this must be taken when generating the initial parameters for the neurons by dividing the initial range for each power by the appropriate factorial or the factorial must be left in place to give equation 4.10c. If this is not done then the higher powers will have a disproportionate significance than a Taylor series would indicate and may result in high sensitivity to small changes in weights or inputs that have a major effect on boundary conditions. As higher orders of power are taken the weights may become so small that their effect is negligible (especially if the implementation has a limited decimal accuracy). In the case of the weights, a well structured training program may compensate for this but a significant point is that the effect of noise would be magnified by these disproportionate values that could have a significant effect on the generalisation abilities of the network.

$$O = \theta + \sum_{p=1}^{m} \sum_{i=1}^{n} x_i^p \frac{w_{i,p}}{p!} \qquad \text{equation 4.10c}$$

Comparison of equations 4.8c and 4.9a shows that if the coefficients of the second order terms of x_1 and x_2 reduce to zero then the TS neuron reverts back to a McCulloch-Pitts performance. This remains true for any number of terms as a McCulloch-Pitts represents the first order of a Taylor Series, or Power Series, neuron.

Note that each connection has a separate weight for each order of series used. It is not practically possible to implement an infinite series without contributing to Bellman's [1961] "curse of dimensionality". Therefore it is necessary to restrict the order of the Taylor series, possibly to as much as a second or third order series. However it has already been demonstrated that a second order series is many times more capable of approximating a non-linear separator as it can follow the contour rather than exploiting tangents. Within the following sections the specifics of restricting the orders are examined.

As each input undergoes a separate Taylor series expansion the operation of the neurons does not represent a true multi-variable Taylor series. This is intentional as if it was allowed to do so then the "curse of dimensionality" would apply and the implementation would produce a variation of the Polynomial Neural Network (PNN) [Ivakhnenko, 1968]. This would have eight variables for a 2nd order 2 input neuron, and as such is frequently

restricted to two inputs such as in the Group Method of Data Handling (GMDH), [Ivakhnenko, 1971].

Chapter 5
Taylor Series Neurons and Networks

5.1 Introduction to Chapter

This chapter demonstrates the implementation of the Taylor series neural model which was presented in the previous chapter. The implementation is compared to equivalent Single and Multi-Layer Perceptron and the results are shown. Significant operations are explored, with specific attention being paid to universality and generalisation. Other major polynomial type networks are also discussed. Finally, the relevance of time-domain operation is introduced and the application of power series networks to this is illustrated with an explanation of how this leads to the Artificial BioChemical Networks presented in the next chapter. Additional results and expanded figures, denoted F.#, are included in Appendix F.

5.2 Background to Chapter

As discussed in Chapter 4, the Taylor Series neuron differs from the McCulloch-Pitts neuron in its connections and weights.

Both neuron types can utilise a range of output functions, common ones being the piecewise linear, threshold (also called heaviside step) function, the logistic sigmoid and the hyperbolic tangent functions. The combination of the summation function and the output function is called the transfer function.

5.2.1 Output Functions

To assist with visualisations the threshold and logistic sigmoid piecewise output functions are illustrated below. The linear and hyperbolic tangent output functions are included in Appendix F.

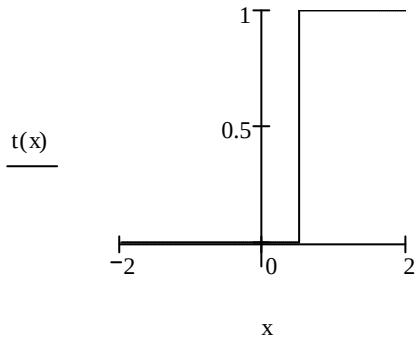


Figure 5.1 – Threshold function *t*(x)

$$t(x) = \begin{cases} 1 & \text{if } t \geq 0.5 \\ 0 & \text{if } t < 0.5 \end{cases}$$

equation 5.1

A threshold function, sometimes called a “heaviside or step function”, usually operates with output {min, max} values and a decision or threshold value. If the sum reaches the threshold values, the output is set to max; otherwise, it is set to min.

The specific function shown here has an output set of {0,1} which are commonly used values. The function may use any pair of values however, these and the set {-1,1} are the most frequently implemented.

The step function produces the binary decision about a threshold point, often denoted by the Greek lower-case theta θ. In the example shown in figure 5.1 and equation 5.1, the theta value is 0.5.

A neuron using such a function is often called a Threshold Logic Unit (TLU). Theta is usually set at the same value for all neurons in the network, often the mid-point between the output max and min. Such a network can be denoted as TLU{θ, min, max} with TLU{0.5,0,1} and TLU{0,-1,1} being typical values. In the perceptron training algorithm, θ is a trainable parameter.

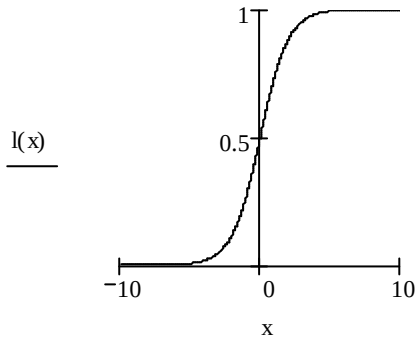


Figure 5.2 – Logistic sigmoid function l(x)

$$l(x) = \frac{1}{1+e^{-(x)}} \qquad \text{equation 5.2}$$

The sigmoid function is probably the most commonly used output function. The example shown in figure 5.2 and equation 5.2 is the logistic sigmoid. This is a useful extension of the step function that provides a continuous solution and overcomes the limitations of the previous binary functions. It is symmetric through the range [0,1] about its output value of 0.5 and has a slope controllable by a parameter rho ρ and an intersect by the use of bias θ, which is inherited from the threshold function. The effect of these is shown in figure 5.3 at the end of this section.

5.3 Design and Implementation

In this section, the capabilities of the individual Taylor Series neuron are explored and compared to the McCulloch-Pitts neuron in the same environment. It is intended that an understanding of the operation of single neurons will assist in the understanding of the operation of a network of neurons.

5.3.1 Taylor Series Neuron Output Functions

Two inputs (x_1,x_2) are used so that the operation can be easily visualised. For the Taylor series neuron, implementation proceeds in increasing order of powers while it is feasible to analyse and represent the unit like this.

The output O and sum S values of the MP and the TS neurons are represented by equations 5.3 and equations 5.4, which are derived from the equations 4.8(a,b,c) and equations 4.10(a,b,c) respectively. This assumes no transfer function is used and represents a linear output function with no amplification.

$$O = S = \theta + \sum_{i=1}^{2} x_i w_i \qquad \text{equation 5.3a}$$

$$O = S = \theta + x_1 w_1 + x_2 w_2 \qquad \text{equation 5.3b}$$

$$O = S = \theta + \sum_{p=1}^{m} \sum_{i=1}^{2} x_i^p \frac{w_{i,p}}{p!} \qquad \text{equation 5.4a}$$

$$O = S = \theta + \sum_{p=1}^{m} \left(x_1^p \frac{w_{1,p}}{p!} + x_2^p \frac{w_{2,p}}{p!} \right) \qquad \text{equation 5.4b}$$

Specifically for a 2nd order and a 3rd order expansion, equation 5.4b can be expressed as 5.4c and 5.4d respectively.

$$O = \theta + x_1 w_{1,1} + x_2 w_{2,1} + \left[x_1^2 \frac{w_{1,2}}{2} + x_2^2 \frac{w_{2,2}}{2} \right] \qquad \text{equation 5.4c}$$

$$O = \theta + x_1 w_{1,1} + x_2 w_{2,1} + x_1^2 \frac{w_{1,2}}{2} + x_2^2 \frac{w_{2,2}}{2} + \left[x_1^3 \frac{w_{1,3}}{6} + x_2^3 \frac{w_{2,3}}{6} \right] \qquad \text{equation 5.4d}$$

The terms in the square brackets represent the change from the previous order of power. Notice that for equation 5.4c, this indicates the change from a 1st order neuron; comparing this to equation 5.3b, shows that a 1st order TS neuron is a MP neuron. This means that if a linear separator is required in a TS network then weights $\{w_{i,p}\}$ will train to 0 for all powers $p > 1$.

A logistic sigmoid output function, as in equation 5.5a, is applied to give the output values of each as shown in equation 5.6 for a MP neurons and equation 5.7 for a TS neuron. This function squashes the output to [0,1]. The maximum and minimum values are theoretically reached when the sum value reaches ±∞. Practically, this occurs due to computational rounding to prevent exponential overflow.

$$Output = \frac{1}{1+e^{-(Sum)}} \qquad \text{equation 5.5a}$$

$$Output = \frac{1}{1+e^{-(Sum)/\rho}} \qquad \text{equation 5.5b}$$

$$O = f(S) = \frac{1}{1+e^{-\left(\theta+\sum_{i=1}^{2} x_i w_i\right)}} \qquad \text{equation 5.6a}$$

$$O = \frac{1}{1+e^{-(\theta+x_1 w_1+x_2 w_2)}} \qquad \text{equation 5.6b}$$

$$O = f(S) = \frac{1}{1+e^{-\left(\theta+\sum_{p=1}^{m}\sum_{i=1}^{2} x_i^p \frac{w_{i,p}}{p!}\right)}} \qquad \text{equation 5.7a}$$

$$O = \frac{1}{1+e^{-\left(\theta+\sum_{p=1}^{m}\left(x_1^p \frac{w_{1,p}}{p!}+x_2^p \frac{w_{2,p}}{p!}\right)\right)}} \qquad \text{equation 5.7b}$$

The equations for the logistic hyperbolic tangent output function are given in Appendix F. These are no more complicated than those of the sigmoid function as it operates on repeated terms.

As the logistic sigmoid function is the main output function used in this thesis, an explanation is given. A logistic function has a range of [0,1], while a threshold function forces a choice between the extreme values of {0,1}. This is not as unrelated as it appears; it is attributable to the previously mentioned term sometimes used with the logistic function, the slope, denoted by the Greek lower case rho ρ and affecting the sensitivity of the function - the range of the sum over which the output produces its extreme values. This is shown in equation 5.5b and figure 5.3. So the threshold replaces an extreme use of ρ and is used to simulate a binary decision of the logistic function.

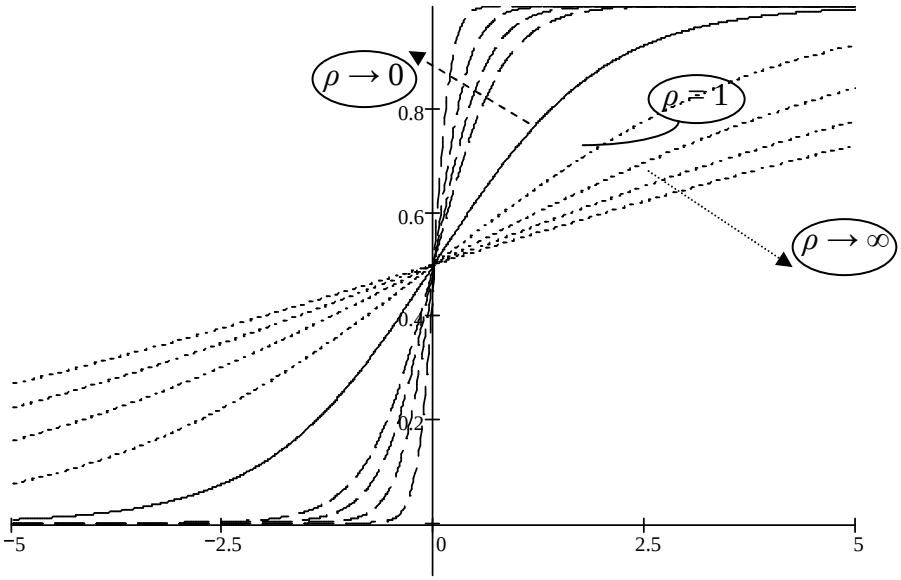


Figure 5.3 – Affect of slope over logistic sigmoid

A neural network makes a decision on the inputs it receives. To do this mathematically, a threshold function can be employed to give a binary response. However, not all decisions are binary and so a continuous output can give a decision expressed in more detail or confidence. If such an output is to be of value, it must be quantifiable. Therefore a squashing function is used to constrain the outputs to a known range so each specific output can be qualified and the values do not tend to such large numbers that the network saturates and becomes untrainable.

In calculating the operations of the neurons, a classical matrix notation or an object model can be used.

5.4 Testing : Single Neuron Functionality

This section visualises the various separator functions in three dimensions. The x-axis and y-axis represent the inputs (x_1,x_2) while the z-axis shows the output value.

5.4.1 McCulloch-Pitts Functions

The McCulloch-Pitts neuron is first examined as a benchmark. This is shown with two different separators; threshold and logistic sigmoid. The piecewise linear and hyperbolic tangent separators are shown in Appendix F.

The sum value of the neuron is calculated as in equation 5.3b. In the specific example shown in equation 5.8, the value of 0.5 is assigned to w_1, w_2 and θ. This is chosen to scale and shape the separator to the relevant axes so that its operation can be clearly shown.

$$Sum = 0.5 \cdot x_1 + 0.5 \cdot x_2 + 0.5 \qquad \text{equation 5.8}$$

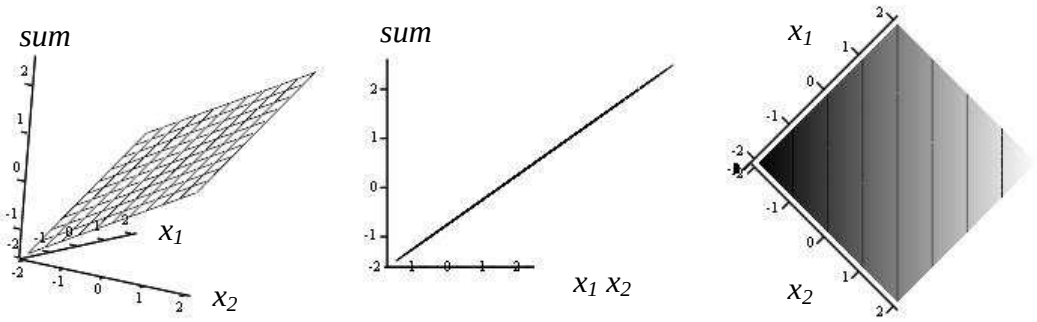


Figure 5.4(a,b,c) – The Sum value expressed as a function of inputs

In figure 5.4a the resultant sum is visualised as a flat plane which can take any angle between the output-axis and the input-plane (made from the input axes x_1,x_2). It can also transect the input-plane in any straight line. The contours in the z-axis, including where it meets the input-plane, represent any proportionality of decision. These contours are all straight lines as shown in figure 5.4b by rotating the figure to view directly through the input-plane. This plane extends to a hyper-plane in more than two input dimensions.

Viewing the gradient from directly into the input-plane shows the increase in output-axis value symmetrically and uniformly with respect to the input-plane.

When a threshold, as in equation 5.1, is applied to the Sum values of equation 5.8 and figure 5.4, the step can be seen. This is similar to the piecewise linear separator without the incremental region. The McCulloch-Pitts neuron originally used this type of separator and it remains popular.

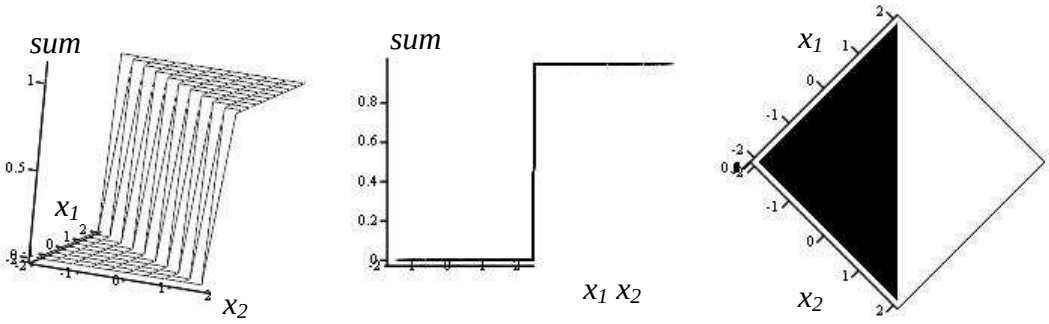


Figure 5.5(a,b,c) – Threshold output functions

Figures 5.5 show the clear binary separation that is typical of the threshold function. As figure 5.5b shows this is still a linear operation, with only one contour in the output, z-axis. The result is obvious when viewed in terms of z-axis values of figure 5.5c.

The sigmoid output functions squash the output into a domain. In contrast to the discontinuities of the piecewise linear and threshold functions. This means that any measurable change in the sum has a distinct effect on the output although changes are not linearly equitable.

In the next figures, the logistic sigmoid separator of equation 5.5 is applied to the values of equation 5.8 and figure 5.4. The resultant figures 5.6 are obviously more complicated than the previous separators. The output range is now squashed to (0,1) and the function is continuous and differentiable over its entire range.

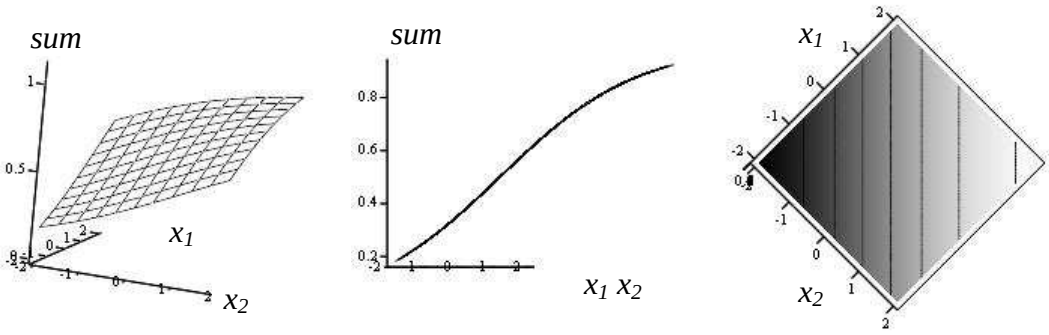


Figure 5.6(a,b,c) – Logistic sigmoid output functions

The sigmoid plane in figure 5.6a can be seen to be an extension of the two dimensional sigmoid curve into three dimensions. This is verified by taking a cross section of the plane in figure 5.6b. The effect on the decision surface is shown in figure 5.6c, where the decision surface can be seen to be completely incremental in terms of output. However, the gradient is non-uniform, which allows the amplification/squashing of data from the input-plane. As before, the expansion into more dimensions results in hyperplanes operating equivalently.

The sensitivity of the function can be tuned to the relevant area of the input plane by the use of a slope parameter. The lineage of the function can be also observed through the effect of $\rho \in \{0.1,1,10\}$, although ρ can take any value.

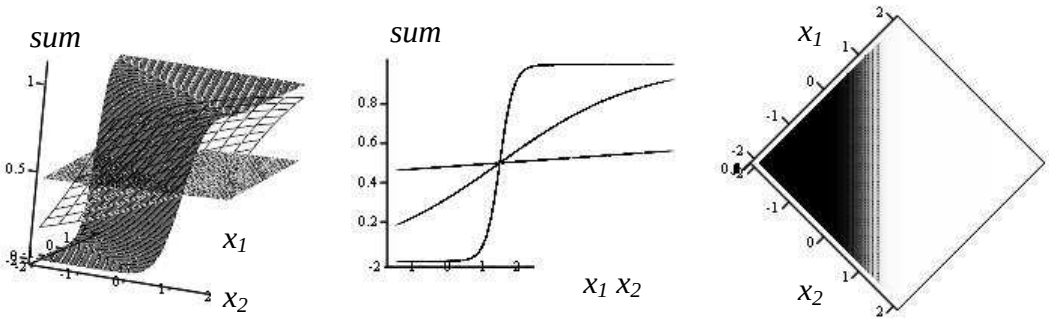


Figure 5.7(a,b,c) – Logistic sigmoid output functions with ρ

As $\rho \rightarrow 0$ the function approaches its equivalent step function, approximating a vertical plane through the input-plane mid point, and becomes increasingly sensitive to changes in the input-plane. As $\rho \rightarrow \infty$, the function stretches out becoming less sensitive to changes in the input-plane and approximates a horizontal plane through the Output-axis mid point.

Figure 5.7a and figure 5.7b show all three values of ρ, the original value of 1 is retained from the previous example is shown by the middle sigmoid. The z-axis decision is shown in figure 5.7c for (ρ = 0.1), a comparison to figure 5.5c and figure 5.5c shows that a decreasing value of ρ causes the output function to approximate the threshold output function.

5.4.2 Taylor Series Functions

The Taylor Series neuron is examined with the same two output functions as the McCulloch-Pitts neuron. These are; threshold and logistic sigmoid. The piecewise linear and hyperbolic tangent are shown in Appendix F. The separators will be shown on a 2nd order and 3rd order Taylor Series neuron. Parameter values will remain the same as before, except when this places the observable region outside any decision boundary. Then specific examples will be shown to demonstrate the full flexibility of the Taylor Series neuron.

The sum values of the neurons are calculated using equation 5.4c for 2nd order and equation 5.4d for 3rd order. These are compared directly to the previous calculation of equation 5.3b, 1st order or MP.

In the specific example shown - a 2nd order Taylor Series neuron in equation 5.9 and a 3rd order Taylor Series neuron in equation 5.10 - the terms in square brackets represent the additional parameters required for the increase to the current order of power. Equation 5.8 and figures 5.4 which represent the McCulloch-Pitts neuron or a 1st order Taylor Series neuron, are repeated for comparison purposes.

The value of 0.5 is assigned to $w_{1,1}$, $w_{2,1}$, $w_{1,2}$, $w_{2,2}$, $w_{1,3}$, $w_{2,3}$ and θ except when stated otherwise. This is done to scale and shape the output function to the relevant axes so that its operation can be clearly shown.

$$Sum = 0.5 \cdot x_1 + 0.5 \cdot x_2 + 0.5 \qquad \text{equation 5.8}$$

$$Sum = 0.5 + 0.5 \cdot x_1 + 0.5 \cdot x_2 + \left[\frac{0.5 \cdot x_1^2 + 0.5 \cdot x_2^2}{2} \right] \qquad \text{equation 5.9}$$

$$Sum = 0.5 + 0.5 \cdot x_1 + 0.5 \cdot x_2 + \frac{0.5 \cdot x_1^2 + 0.5 \cdot x_2^2}{2} + \left[\frac{0.5 \cdot x_1^3 + 0.5 \cdot x_2^3}{6} \right]$$

equation 5.10

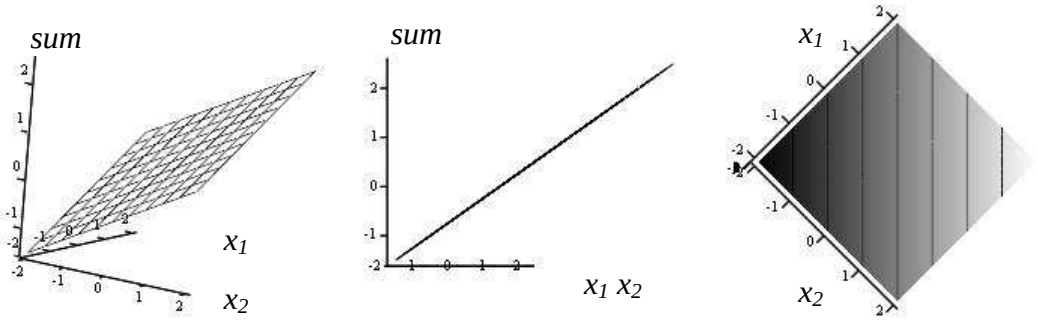


Figure 5.4(a,b,c) – The Sum value expressed as a function of inputs

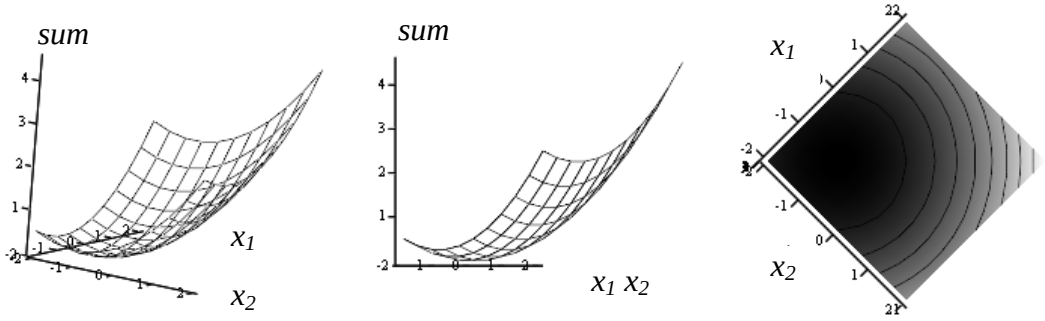


Figure 5.8(a,b,c) – The Sum value of 2nd order Taylor Series neuron

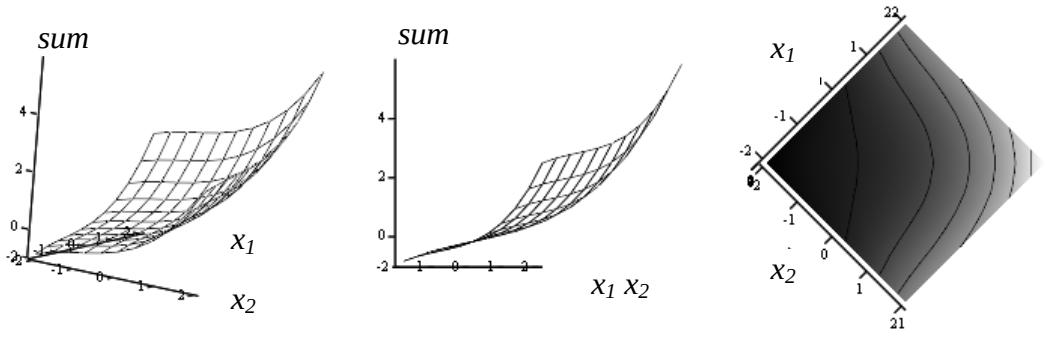


Figure 5.9(a,b,c) – The Sum value of 3rd order Taylor Series neuron

As the Taylor Series neuron increases from a 1st to 3rd order it may immediately be observed, through figures 5.8(a,b) and 5.9(a,b) that there is now a non-linear summation function and this takes the form of a contoured-plane in the output-domain.

There does not initially appear to be much variation between the 2nd and 3rd order. However, if figure 5.8c and figure 5.9c are examined, it may be observed that an adaptability in dimensionality has occurred with the increase to 3rd order. The contours in figure 5.8c show that a 2nd order power is all that is required for a non-linear separator; however, observing the contours shows that there is an apparent symmetrical but non-uniform (in the input-plane) relationship to the output-domain. So far, all examples have used identical inputs in the dimensions of the input-domain and this gives rise to the symmetry being equal in all input-plane dimensions. This is shown more clearly in figures 5.10. The introduction of the 3rd order terms allows an extra variation. The variation in the input-plane is non-linear; however, it is also non-symmetrical in input-plane dimensions.

It can be surmised that extending the order to 4th and above will extend the variation between the input-dimensions and the output-plane. What practical use this may have will depend on the complexity of the input-domain. Any increase in order beyond the requirements of the input-dimensions will simply increase the number of parameters that the network has, beyond those required. This will result in reduced training performance in terms of number of epochs and over-fitting. This is illustrated later in the experimental section.

$$Sum = 0.5 + 0.05 \cdot x_1 + 0.05 \cdot x_2 + \left[\frac{1.0 \cdot x_1^2 + 1.0 \cdot x_2^2}{2} \right] \qquad \text{equation 5.11}$$

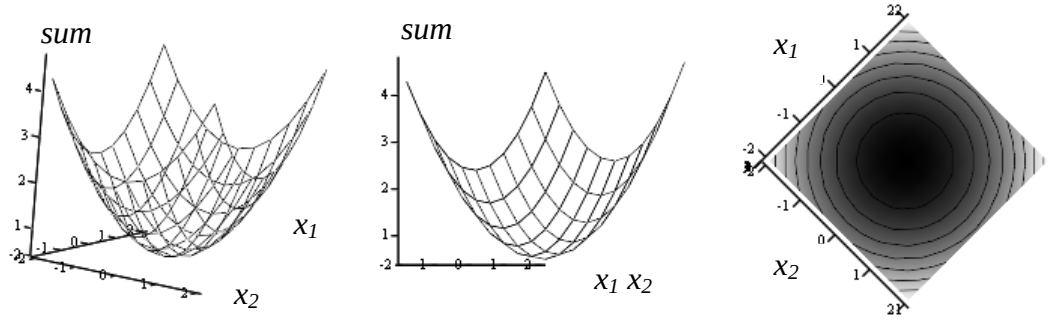


Figure 5.10(a,b,c) – Sum value of 2nd order Taylor Series neuron focusing on decision region

The non-linear symmetry of the input-plane (x_1,x_2) in the output-domain is clearly shown in figures 5.10. This is a marked improvement over the linear-plane of the 1st order as it is also capable of representing this. It is achieved by increasing the co-efficient of the 2nd order terms; this corresponds to an increase in magnitude of weights as shown in equation 5.11.

The simple extension of the 1st to 2nd order allows output functions similar to the Sigma-Pi neurons described by Rumelhart et al., [1986] but with a gradient expression and without the restrictions on network topology and non-linearity. While the extension to 3rd order gives the additional capabilities of non-symmetry in the 2nd dimension, this is achieved in the same way as for the 2nd order, with the emphasis on 3rd order terms, shown in figures 5.11 and equation 5.12.

$$Sum = 0.5 + 0.05 \cdot x_1 + 0.05 \cdot x_2 + \frac{0.05 \cdot x_1^2 + 0.05 \cdot x_2^2}{2} + \left[\frac{1.5 \cdot x_1^3 + 1.5 \cdot x_2^3}{6} \right]$$

equation 5.12

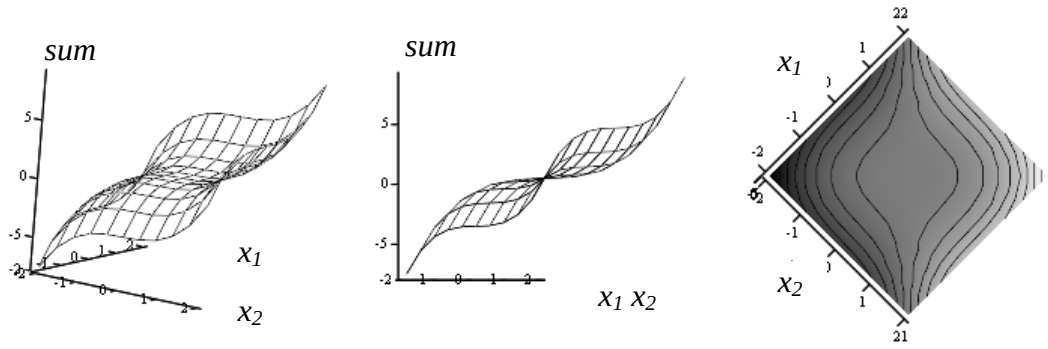


Figure 5.11(a,b,c) – Sum value of 3rd order Taylor Series neuron focusing on decision region

The effect of each order of power can be shown through independently varying the coefficient, or weight, affecting it. This is first shown in all orders with positive values of the 1st, 2nd, and 3rd order Taylor Series neurons. Negative values simply invert the decision surface of the output-axis.

Skewing the inputs in the 1st order neuron (McCulloch-Pitts) results in a tilt of the flat plane towards the input-axis with the lower coefficient, shown in equation F.8 and through comparison of figures 5.4 and figures F.7. This effect of skewing 1st order inputs is inherited by the Taylor-Series neurons when the pairs of higher orders coefficient remain equal. The 2nd order neuron is shown in equation F.9 and by comparison of figures 5.10 and figures F.8. For the 3rd order neuron equation F.9 and a comparison of figures 5.11 and figures F.12 apply.

Higher order terms can be examined by skewing individual orders, or combinations of higher orders, while fixing the 1st order terms and remaining higher orders. The various combinations of this are examined in Appendix F for 2nd and 3rd order neurons.

Skewing the 2nd order coefficients on their own causes the decision surface to stretch along the input-axis with the lower coefficient. This is best visualised by comparing figure 5.10c and figure F.9c although it is observable in the other relevant figures.

The effects of skewing the 1st or 2nd order terms individually on their own affect different aspects of the decision surface. This implies an independence of operation. If both the 1st and 2nd order terms are altered from equation 5.11 to give equation F.9 and equation F.10 and applied simultaneously, they give equation F.11. A comparison of figures F.8, figures F.9 and figures F.10 shows the independence of the actions of the 1st and 2nd orders. Figures F.10 shows a direct combination of both effects.

An effect of the results of this independence between the different orders mean that the decision surface can be altered independently towards what is required. This is without an interaction between the orders becoming reliant on parameter interaction and therefore much more difficult to control than independent parameters.

Examining the effects of different input domain values with added 3rd order terms results in figures F.12 to figures F.24. These show both the independence of the power terms and the effect of each order on the separator. These are all created by modifying the terms of equation 5.12 and can be compared to the symmetrical input-domain in figures 5.11.

The above figures, and those in Appendix F, demonstrate the flexibility of the Taylor Series neuron with respect to its inputs. In each case the variation in the coefficients is seen to be independent, allowing an element of control of the neuron while being able to exploit all the variation of the output-domain.

In the previous sections, the polarity of the coefficients has always been positive. The input values have taken either positive and negative values. What follows are examples of the coefficients taking different polarity, positive/negative values, for the same values of inputs.

The combinations of possible values are enormous, so the following are examples of interesting occurrences to show the flexibility of the error surface. If all values take the opposite polarity then the error surface is inverse in the output-domain; therefore, any combinations of values that produce valleys in the output-domain will produce peaks if the polarities of all values are inverted.

McCulloch-Pitts neuron.

$$Sum = 0.5 \cdot x_1 - 0.5 \cdot x_2 + 0.5 \qquad \text{equation 5.13}$$

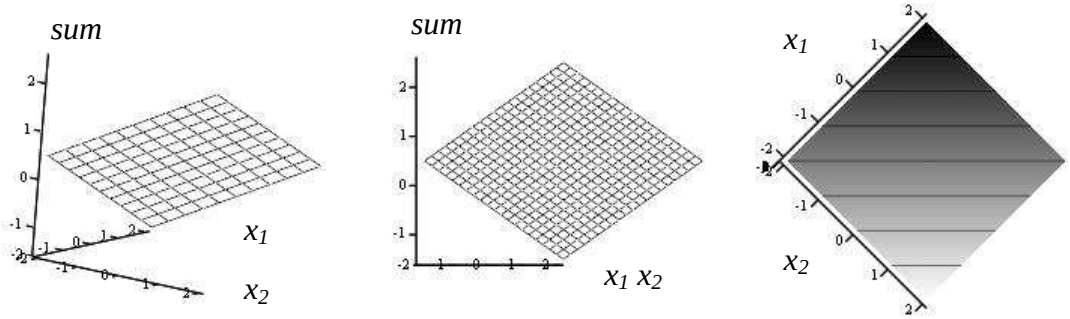


Figure 5.12(a,b,c) – The Sum value expressed as a function of inputs of McCulloch-Pitts neuron with opposing polarity of coefficients

The sum values shown in equation 5.8 are altered to a negative coefficient for the second input, to give equation 5.13. A resultant angular change occurs in the orientation of the output-plane. This is manifested in the decision-surface as a rotation of 90° around the

decision-axis as shown in figure 5.14c. This alteration does not express any change to the ability of the McCulloch-Pitts neuron to be a linear separator. If the other coefficient is given a negative polarity, then the rotation occurs in the opposite direction. If both are applied a 180° rotation occurs.

Taylor Series neuron - 2nd order.

$$Sum = 0.5 + 0.05 \cdot x_1 + 0.05 \cdot x_2 + \left[\frac{1.0 \cdot x_1^2 - 1.0 \cdot x_2^2}{2} \right] \qquad \text{equation 5.14}$$

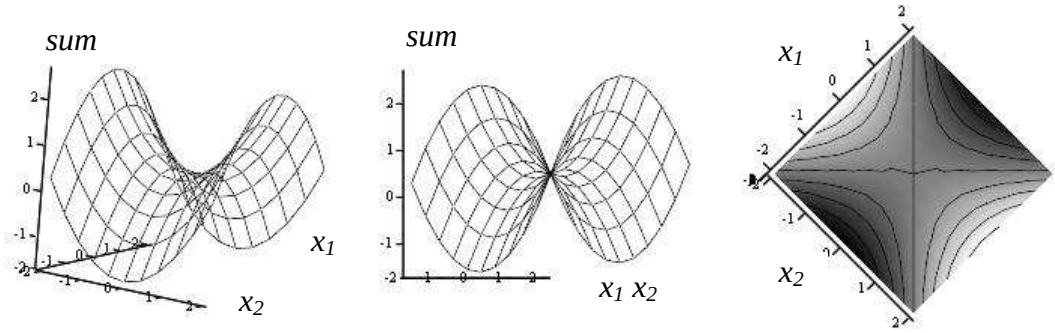


Figure 5.13(a,b,c) – Sum value of 2nd order Taylor Series neuron focusing on the decision region with opposing polarity of 2nd order coefficients

The 2nd order Taylor Series neuron can partially invert its decision-surface by altering equations 5.11, 5.12 and F.9 to F.25. These regions are only symmetrical as the opposing coefficients of equation 5.14 have equal magnitude; the region can stretch and tilt just as in the previous section. Once separators are applied later in this chapter, the importance of this specific example is explored.

As before, when different orders of coefficients were skewed with respect to the input-axis, the polarity of these coefficients can also be altered. The same rules apply, as shown in figure 5.12. A 1st order swap in polarity causes a rotational effect in the McCulloch-Pitts neurons. If applied to the 1st orders of a 2nd order or higher Taylor Series neuron, then the decision-surface undergoes the same rotation. Likewise, the effects of other orders are independent when applied to neurons of higher orders.

Taylor Series neuron - 3rd order.

$$Sum = 0.5 + 0.05 \cdot x_1 + 0.05 \cdot x_2 + \frac{0.05 \cdot x_1^2 + 0.05 \cdot x_2^2}{2} + \left[\frac{1.5 \cdot x_1^3 - 1.5 \cdot x_2^3}{6} \right]$$

equation 5.15

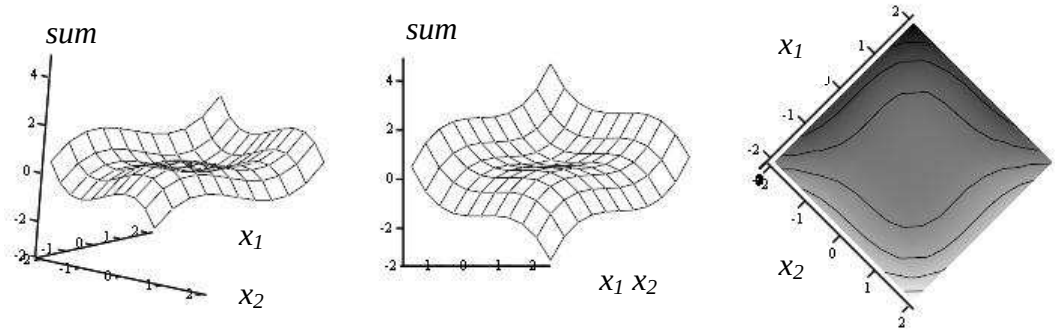


Figure 5.14(a,b,c) – Sum value of 3rd order Taylor Series neuron focusing on the decision region with opposing polarity of 3rd order coefficients

The 3rd order Taylor Series neuron allows a decision boundary with more gradient changes, which permits some interesting behaviour. This does not inhibit it from mimicking the capabilities of the 2nd order neuron - it can utilise any of these and apply its own. Equation 5.15 is modified from equation 5.12 in the same way as before. The figures 5.14 can be compared to figures 5.11 to demonstrate the decision-surface changes as can figures 5.15 and F.25 to F.27.

Taylor Series neuron - mixed orders.

$$Sum = 0.5 + 0.5 \cdot x_1 - 0.5 \cdot x_2 + \frac{0.5 \cdot x_1^2 - 0.05 \cdot x_2^2}{2} + \left[\frac{-1.0 \cdot x_1^3 + 1.0 \cdot x_2^3}{6} \right]$$

equation 5.16

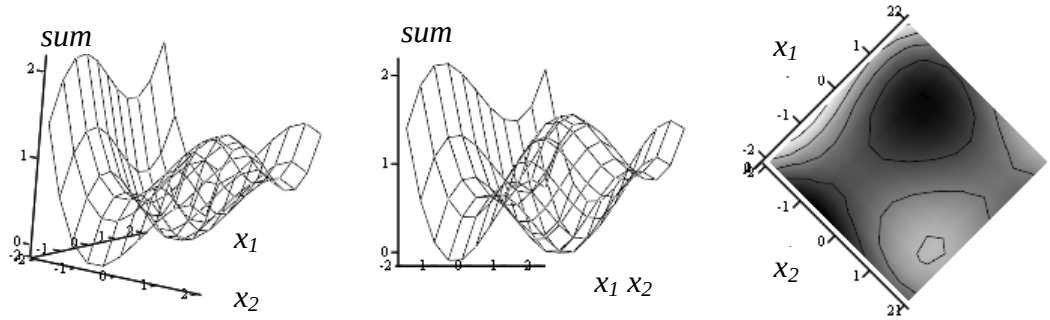


Figure 5.15(a,b,c) – Sum value of 3rd order Taylor Series neuron focusing on the decision region with opposing polarity of various coefficients

Despite the intricacies of the output-domain, it is when the output functions are applied that the values become obvious. This is attended to next in this chapter.

Taylor Series neuron – output functions.

The output functions are applied to the 2nd order Taylor Series neuron as expressed in equation 5.11 and shown in figures 5.10. The following figures 5.16, 5.17 and F.28, F.29 are all performed on the same 2nd order Taylor Series neuron which has parameters that focus the figures on the decision region.

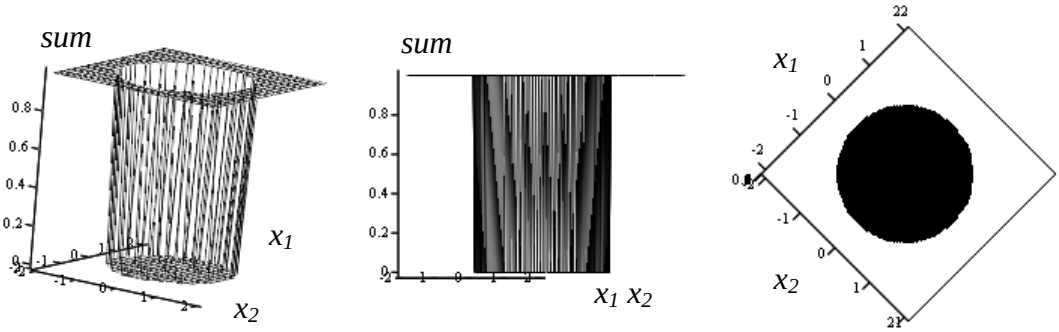


Figure 5.16(a,b,c) – Threshold output functions

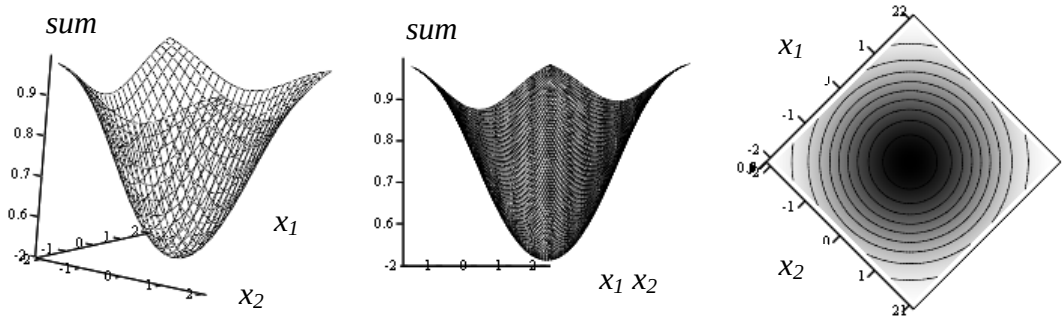


Figure 5.17(a,b,c) – Logistic sigmoid output functions

The same output functions are now applied to the 3rd order Taylor Series neuron as expressed in equation 5.12 and shown in figures 5.11. The following figures 5.18, 5.19 and F.30, F.31 are all performed on the same 3rd order Taylor Series neuron which has parameters that focus the figures on the decision region.

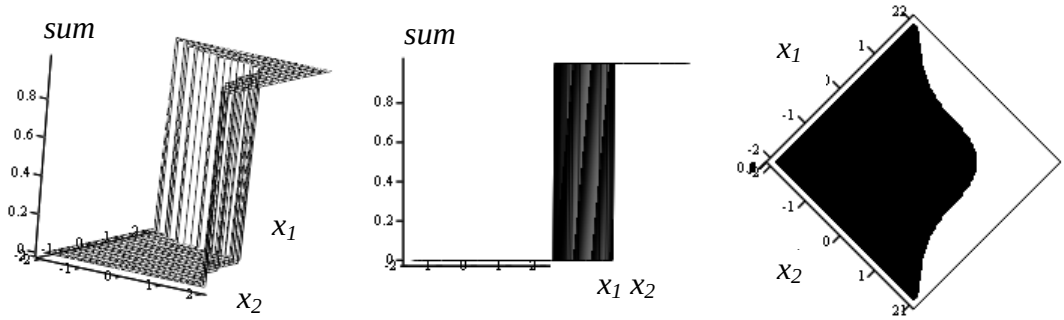


Figure 5.18(a,b,c) – Threshold output functions

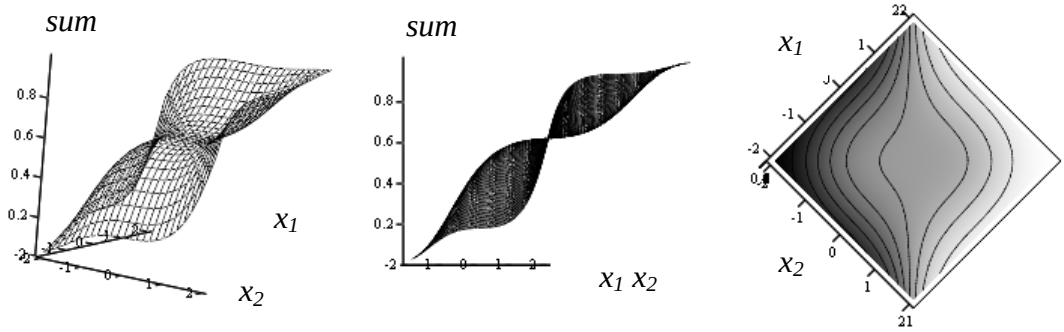


Figure 5.19(a,b,c) – Logistic sigmoid output functions

From examining figures 5.16 to figures 5.19 and F.28 to F.31, it appears that there are major differences between the 2nd and 3rd order Taylor Series neuron. This shows how the addition of different power terms can dramatically change the operation of the neuron. The 2nd order output functions have similarities which are in part due to the equal values used for the coefficients of the input-axis, that give rise to a symmetrical decision-region in both axes, essentially a circle in the decision-surface. What is obvious from comparing the figures from each 2nd order and 3rd order set, is that the 3rd order neurons appears to retain some symmetry, as would be expected given the parameters used; however, it is not the same in every dimension. This is caused by odd powers operating on negative input values. Increasing the order of power has a resultant increase in the degrees of freedom the neuron can operate with.

In a comparison against the McCulloch-Pitts figures 5.5 to figures 5.7 and F.3 to F.6, it may be seen that the Taylor Series neuron can act as a non-linear separator no matter which output function is used. Equally importantly, they can enclose or isolate a complete region of the decision surface and, if there is a continuous function applied to this, it can return a continuous output, as in the sigmoid functions.

Taylor Series neuron - non-linear and isolating behaviours.

The capabilities that are permitted by the non-linear and isolating behaviours are of significant importance to ANNs. A set of examples of what these can do is now presented.

The following shows a 2nd order Taylor Series neuron as in equation 5.17 and figures F.26. The decision-surfaces shown are for the threshold in figures 5.20 and the logistic sigmoid in figures 5.21. These should be compared with figures 5.16 and 5.17 respectively.

$$Sum = 0.5 - 0.95 \cdot x_1 + 1.15 \cdot x_2 + \left[\frac{0.5 \cdot x_1^2 + 3.0 \cdot x_2^2}{2} \right] \qquad \text{equation 5.17}$$

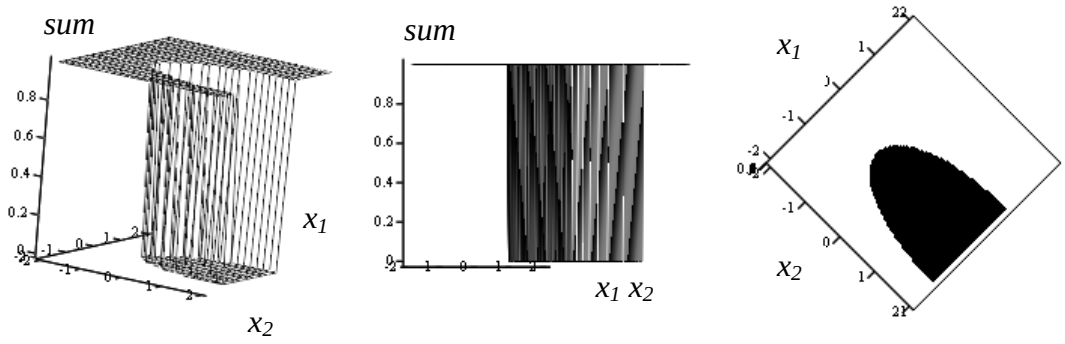


Figure 5.20(a,b,c) – Threshold output functions

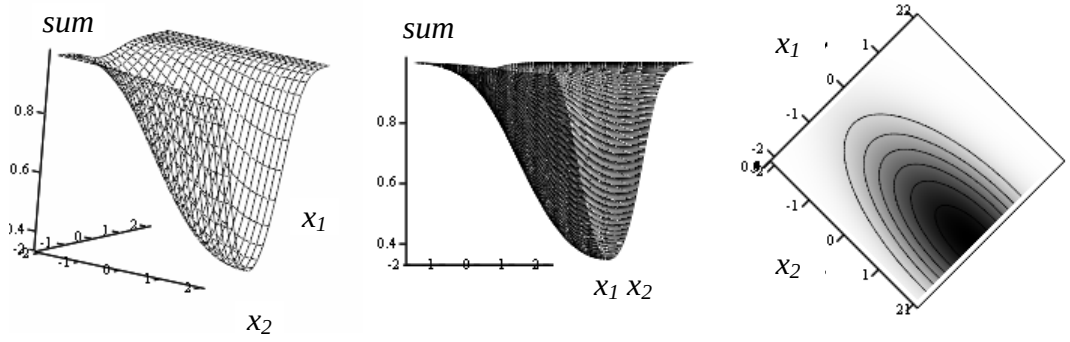


Figure 5.21(a,b,c) – Logistic sigmoid output functions

If the 2nd order Taylor Series neuron has parameters that are allowed to become non-symmetrical, its output functions can form an oval for threshold functions, as shown in figures 5.20. It may also form a series of concentric ovals, for continuous or sigmoid functions, as shown in figures 5.21. A piecewise linear function would show an oval plateau bounded by a series of concentric oval contours and finishing with a second outer oval plateau. These ovals can be larger than the decision region and so bisect it in curves and curved hyper-planes. This is something a McCulloch-Pitts neuron is not capable of and that a network of such neurons can only approximate.

If another case is examined, that of the 2nd order neuron shown in equation 5.14 and figures 5.13, then an even more important capability can be demonstrated.

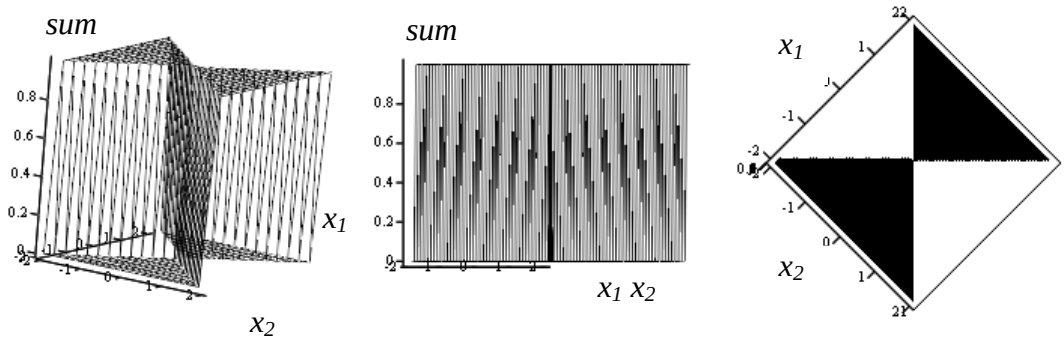


Figure 5.22(a,b,c) – Threshold output functions

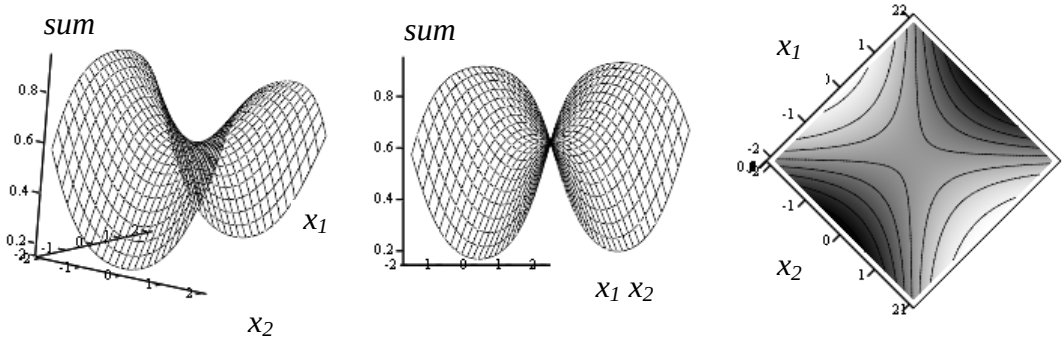


Figure 5.23(a,b,c) – Logistic sigmoid output functions

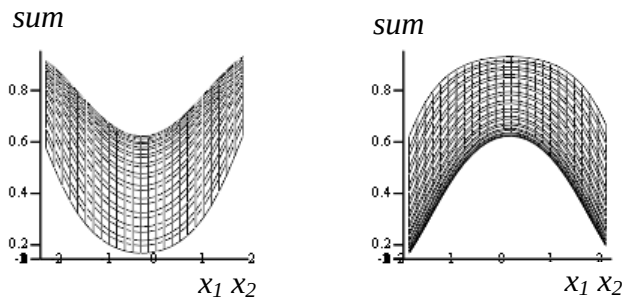


Figure 5.23(d,e) – Rotations of logistic sigmoid output functions

It can be seen in figures 5.22 that a threshold function can separate non-continuous regions; this shows that a 2nd order Taylor Series neuron can provide a solution to the XOR (parity-bit) problem presented by Minsky and Papert [1969], which was a significant factor in the downturn of research in ANNs until backpropagation became widely known in the 1980s as discussed in Chapter 3. The sigmoid functions produce a saddle or butterfly shape which can be viewed in the various rotations of figures 5.23. For a MLP to perform this solution to the XOR problem at least three McCulloch-Pitts neurons are required. As

the MLP can only approximate a curve, it is a further order of dimension out in an attempt to approximate a curved plane.

A 3rd order Taylor Series neuron can operate with even more flexibility than the 2nd order. The examples shown are based on the 3rd order neuron of equation 5.16 and figures 5.15. The decision-surfaces shown are for the threshold in figures 5.24 and the logistic sigmoid in figures 5.25.

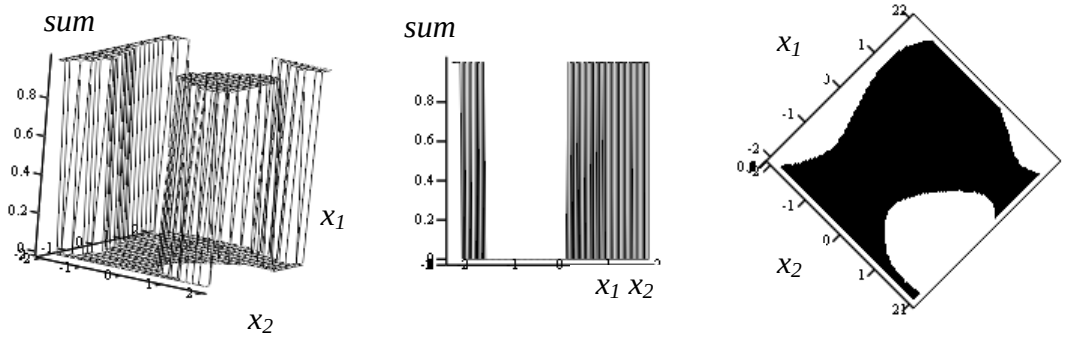


Figure 5.24(a,b,c) – Threshold output functions

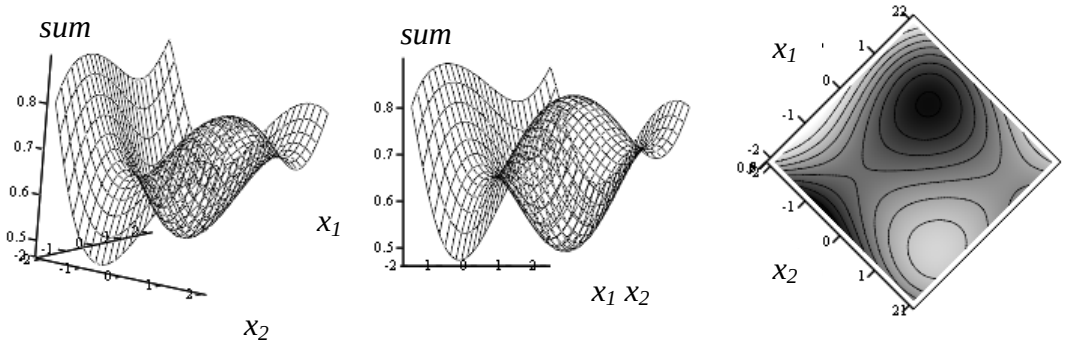


Figure 5.25(a,b,c) – Logistic sigmoid output functions

The use of 3rd order terms allow the Taylor Series neuron to isolate a non-symmetrical region and to divide up the remaining problem-domain. This is shown clearly in figures 5.24 and can be observed in the contours of figures 5.25. Behaviour like this is far beyond the flexibility of McCulloch-Pitts neurons. However, the more flexible the behaviour, the harder it is to control and therefore attention must be paid in the training method to this - and subduing terms such as the factorial divisor are advisable. If the orders of power are increased, then both the flexibility and difficulty in control will increase also.

5.4.3 Universality and Robustness Trade-Off

The more universal a neuron becomes, the more functional it becomes. It is then capable of performing mappings beyond those of other neurons. This appears to be a straightforward benefit. The drawback is that as this results in a single neuron performing complex functions, the system becomes dependant on individual neurons. A critical failure may then occur if one neuron is damaged (as opposed to the gradual degradation of performance which occurs in networks with many simpler neurons).

An overdependence on single components therefore creates a lack of redundancy which can lead to a delicately balanced system where good noise tolerance is un-achievable. This results in a trade off between universality and robustness.

This point does not reduce the requirement for a neuron to be universal, or at least more universal than the models currently implemented. However it may be that such neuron should be able to be implemented in instances that allows them to perform at different levels of functionality.

5.4.4 Summary – Single Neuron Functionality

In a comparison of the Taylor Series neuron against the McCulloch-Pitts neuron, it can be seen that the Taylor Series neuron is not bound by the linear separator properties of the McCulloch-Pitts neuron - although it can, if necessary, adopt them. Clearly, the Taylor Series neuron can adopt curved separators and through this solve problems like the parity bit problem. The flexibility of the Taylor Series neuron must be controlled with a carefully constructed training algorithm, as the addition of new orders of powers significantly changes the error surface by introducing new degrees of freedom. However, all orders of power operate independently and therefore, if there is an error-minimum-seeking training method employed, these sudden changes are likely to be avoidable.

The Curse of Dimensionality [Bellman, 1961] associated with higher-order units that prevents their use with many input parameters or orders of power is controllable and limited in the Taylor Series neuron. The complexity increases in a Taylor Series neuron is

in order of sums as new orders are added, rather than products. This is due to the lack of interaction between the inputs, which is not the case in other Polynomial units.

The operation of Taylor Series neurons as part of a practical network is the subject of the next section.

5.5 Taylor Series ANNs vs. McCulloch-Pitts ANNs

In the previous section, the flexibility of Taylor Series neurons was explored. Their abilities were examined and contrasted with those of McCulloch-Pitts neurons. This section now considers networks of such neurons and how they perform against each other.

The networks used in these experiments are presented in two topologies. Each of these is populated with MP neurons for benchmark testing and then TS neurons for comparison testing. The topologies are a single-layer network and a two-layer network. When populated with McCulloch-Pitts neurons these are referred to as a Single-Layer Perceptron (SLP) and a Multi-Layer Perceptron. When populated with Taylor Series neurons they are termed as a Single-Layer Taylor Series network (SLT) and a Multi-Layer Taylor Series network (MLT). All of these networks have an additional layer of input nodes.

The networks implement a standard logistic sigmoid function as shown in equation 5.5.

The performance characteristics of the networks during and after training are compared. These characteristics include training time, memory capacity, and pattern recognition ability.

5.5.1 First Comparison

This is a comparison of a SLP and a SLT using the standard Delta Rule training for the McCulloch-Pitts neurons and a derived Delta Rule for the Taylor Series neurons. There is no variation in network topology as it is determined by the problem parameters; both networks have 35 input nodes (one per pattern data unit) and 26 output neurons (one per input pattern). The patterns presented are shown in the next section.

The purposes of this test are to examine training time of the networks and to assess their noise tolerance capabilities.

5.5.2 Second Comparison

The second comparison is of a MLP and a MLT using a modified Genetic Algorithm as the training mechanism. A compact parameter problem is presented to compare network training time on multiple layers, and test MLP and MLT topology requirements. The purposes of this test are to examine training time of the multi-layer networks and to determine the minimum network size.

5.5.3 Third Comparison

The third comparison uses the same MLP and MLT as the previous test but uses a larger training set to ensure that the networks are capable of expanding their problem domains, and to quantify any effect of this on network size.

5.6 Comparison Parameters

Three performance parameters are compared. These are: training time to achieve a target error, minimum network size to achieve the target error (memory capacity) and noise tolerance.

5.6.1 Training Time

Training times are quantified in terms of epochs. Consideration is also given to the computational overheads, as different networks may have different lengths of epoch. In addition to total training time, it is important to observe the error profile during training as this can reflect on both possible improvements to the training algorithm and on the performance of the trained network.

In general, a shorter training time is advantageous. However, most practical networks are fully trained before their operational phase so in most cases only very significantly longer training times, (to the extent that the network is impractical) are of importance.

5.6.2 Minimum Network Size

The ability of one network to show the same memory capabilities as another with a reduced network size is important. The smaller network demonstrates superior universality - the ability to map from input to solution space.

5.6.3 Noise Tolerance

The network's resistance to noise is a test of its ability to generalise. A network with a higher noise tolerance is more capable of correctly classifying new or damaged inputs from the data set.

5.7 Design and Implementation

The networks were tested with two different data sets. A [3x3] grid, related to robot vision, was used for the MLP vs. MLT comparison. This allowed a simple network to be set up for a direct comparison of the neuron types. A [7x5] pattern set, see figures 5.26, was first used to test the noise performance of the neurons in the SLP and SLT. This was also used on the MLP and MLT to confirm that the performance achieved using the smaller grid was scalable.

5.7.1 Data Set

The first data set tests the network's ability to recognise the 26 capital letters of the western alphabet, figures 5.26a. This is a standard data set, used in the University research group and many others. Each image is a [7x5] matrix (35 inputs). There are two sets, a binary {0,1} and a continuous range [0,1]. These values were chosen as they represent the range of the output function used, the logistic sigmoid . The second data set tests its ability to recognise patterns on a [3x3] matrix, (9 inputs), see figures 5.26b.

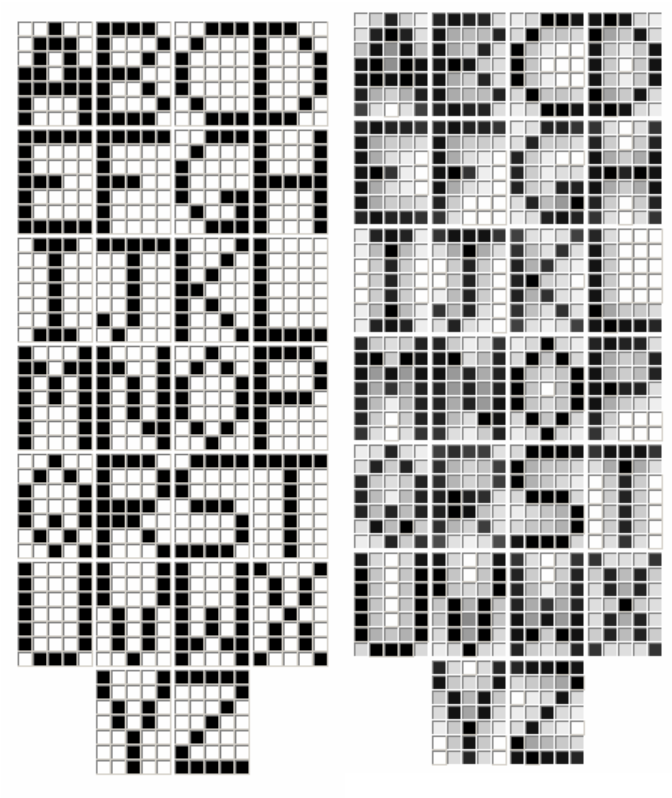


Figure 5.26a(i,ii) – Network training sets – 5x7 grids

Figure 5.26b – Network training set – 3x3 grid

5.7.2 Single-Layer Network Topologies

The network topologies are shown in a general format; the number of input nodes is dictated by the number of parameters in the training patterns, and the number of output neurons is dictated by the number of patterns the network is required to classify.

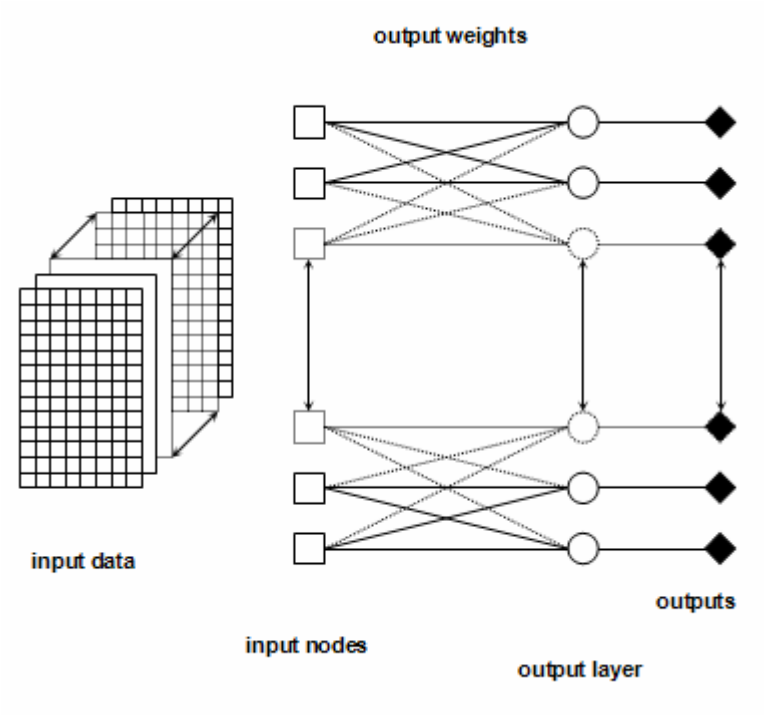


Figure 5.27 – SLP – Single-Layer Perceptron

The single-layer network implementing McCulloch-Pitts neurons shown in figure 5.27 has a topology determined by the problem parameters as mentioned above.

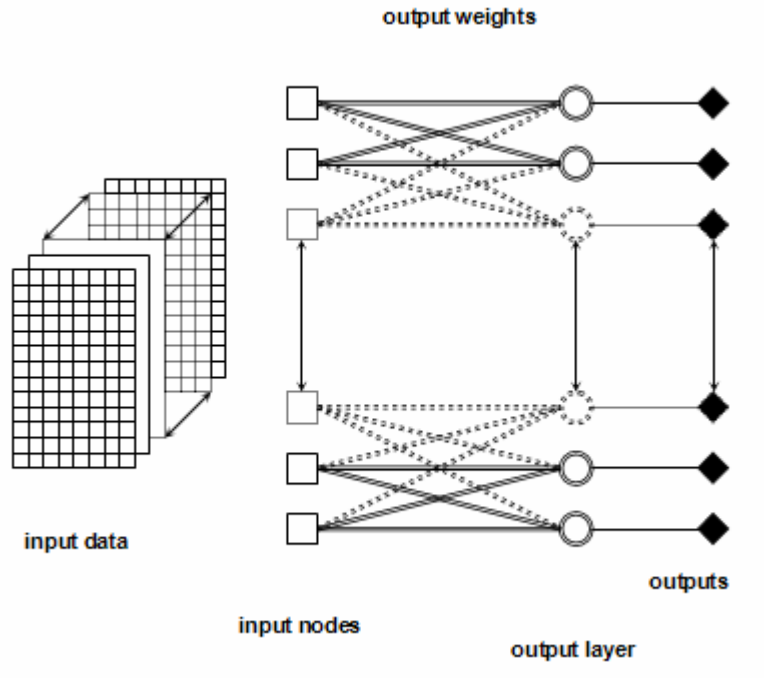


Figure 5.28 – SLT – Single-Layer Taylor Series network

The single layer implementation using Taylor Series neurons, (figure 5.28), has the same topology as the SLP. The difference is in the connections between the output neurons and the input nodes. There are the same number of these however each one may have multiple weights - one for each order of power that the neuron is implementing. The Taylor Series neurons and weights are denoted with double borders.

The outputs, targets and errors are utilised by the Delta Rule by Widrow and Hoff [1960], for the SLP and a derived Delta Rule by this author, that takes the powers into account, for the SLT. It is not necessary to give a full expansion at this point. Appendix C on the Backpropagation Algorithm includes the Delta Rule to train the output layer and the derived Delta Rule for the SLT.

5.7.3 Multi-Layer Network Topologies

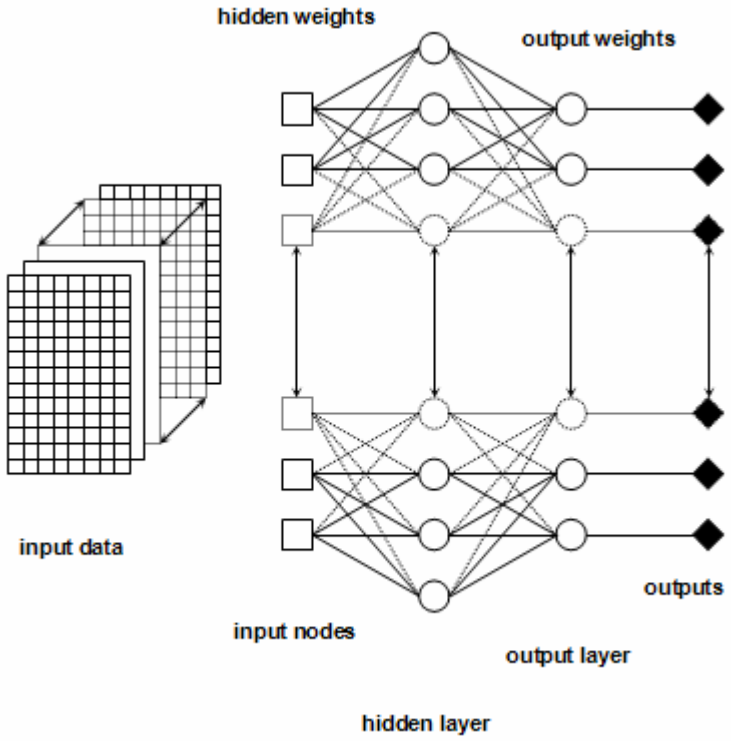


Figure 5.29 – MLP – Multi-Layer Perceptron

The multi-layer implementation of the Perceptron using McCulloch-Pitts neurons, see figure 5.29, retains the same input node and output neuron structure as in the SLP;

however it also has a hidden layer. The structure of the network shown is a standard and well tested topology.

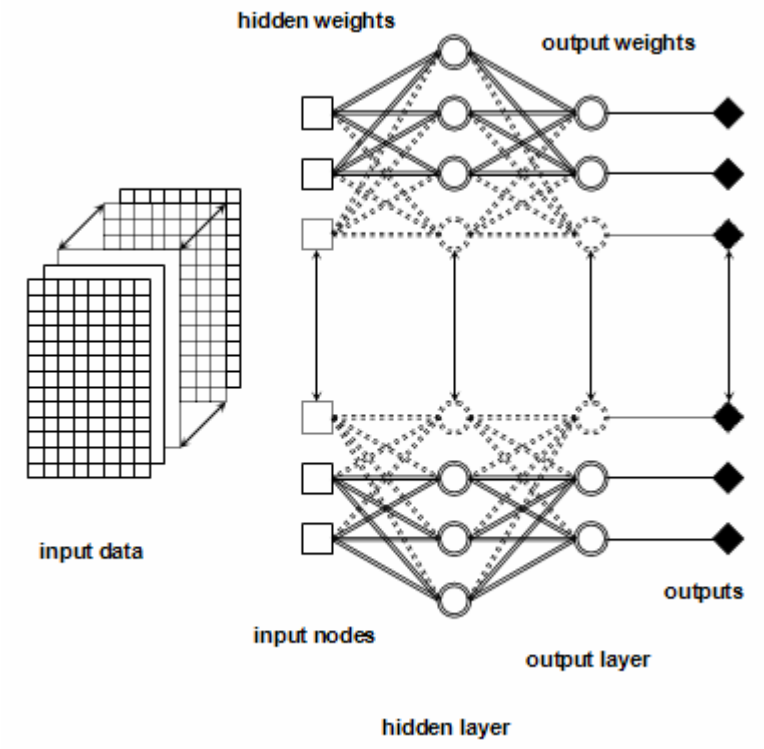


Figure 5.30 – MLT – Multi-Layer Taylor Series network

The network shown in figure 5.30 represents a multi-layer Taylor Series network. The Taylor Series neuron operation is the same as in figure 5.61 and is denoted with double borders.

In the multi-layer networks the hidden layer is placed between the inputs and outputs and denoted as circles. The number of hidden neurons is not usually determined by exact methods in neural networks but by trial and error.

Training on all multi-layer networks is via a modified Genetic Algorithm.

5.8 First Experiment : Comparing McCulloch-Pitts SLP and Taylor Series SLT – 5x7 test

This experiment tests training times and noise tolerance. The McCulloch-Pitts based network of figure 5.27 and the Taylor Series based network of figure 5.28 were trained on the 26 patterns shown in figure 5.26a. Where it assist understanding, larger versions of all multi-line graphs are supplied in Appendix F.

5.8.1 Training Time

The first set of information that was presented to the networks use the input values from figure 5.26a(i); all inputs are from the set {0,1}. The output targets belong to the set {0,1}. The target error was set at 0.125. The error used is the Least Mean Square calculation as shown in Appendix C on Backpropagation.

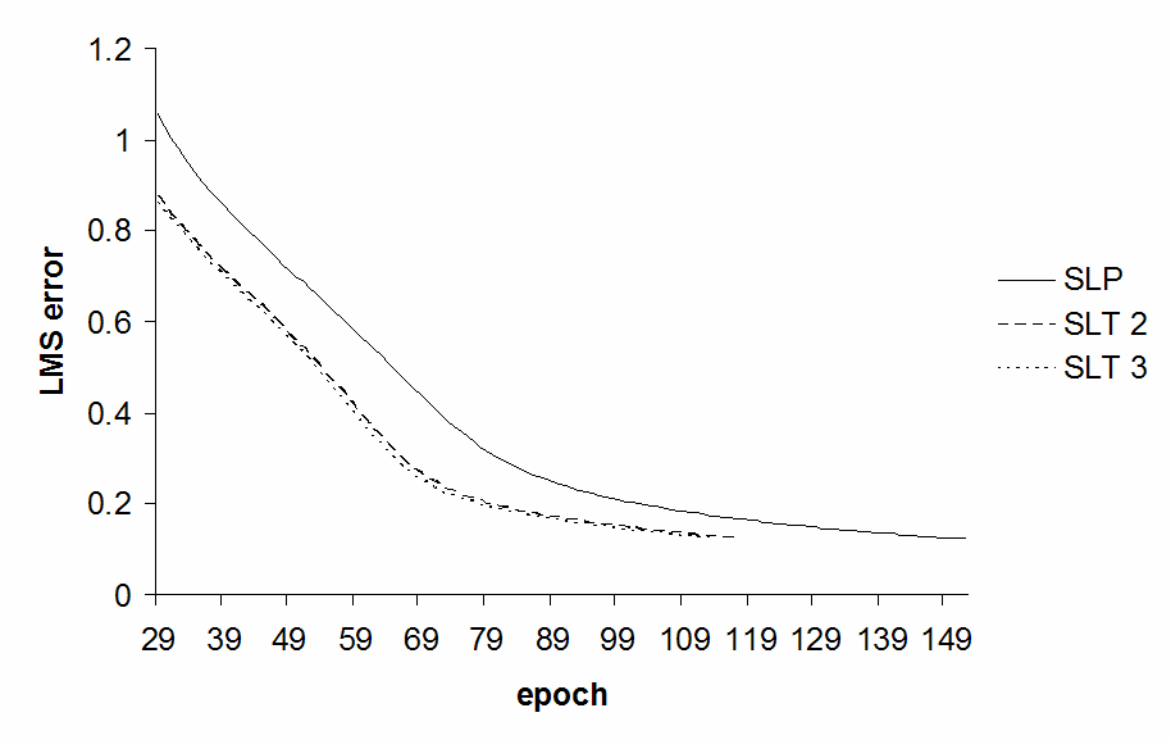


Figure 5.31a – Comparison of error vs. epoch for SLP and SLT networks

The performances of the SLP appear similar to the SLT as the order of the Taylor Series is increased. Only results from 29 epochs onwards and the first 3 orders are shown for

clarity. Higher orders were tested, but these performed little differently from the 3rd order. This is discussed later in the chapter.

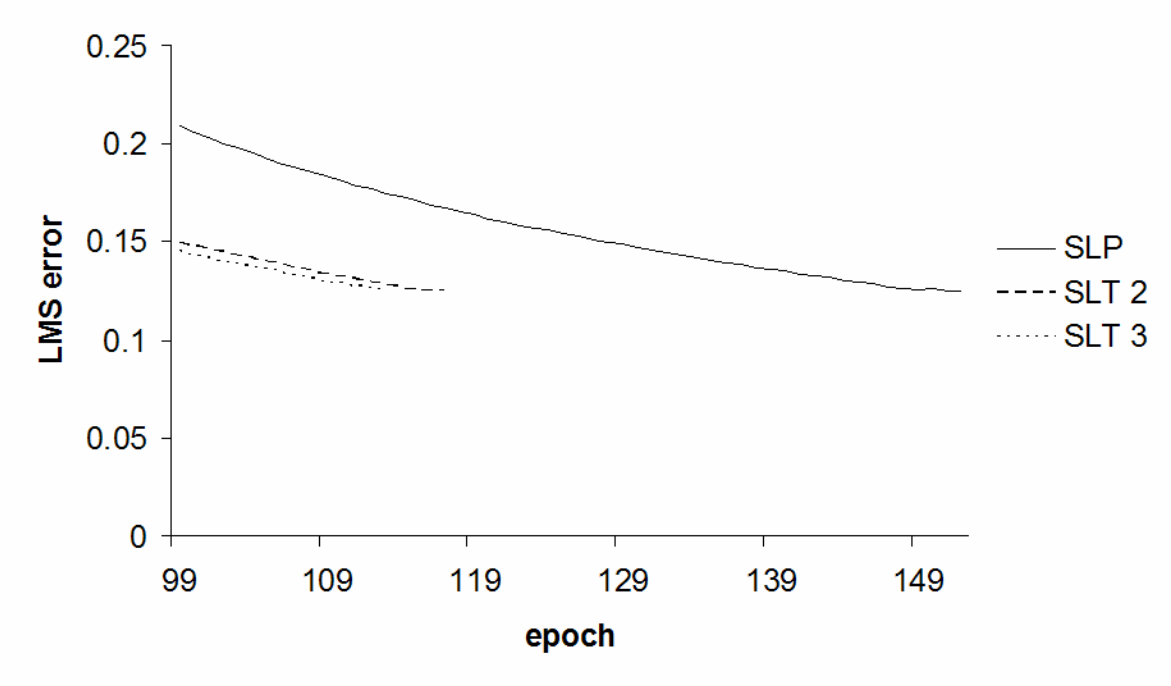


Figure 5.31b – Comparison of error vs. epoch for SLP and SLT networks
epoch ≥ 99

The training performance becomes different at low errors. Due to the initial similarity, this is not clear from figure 5.31a and so is shown by figure 5.31b. The graph focuses on the epochs from 99 onwards, where the advantages of the SLT in achieving lower errors can be seen. The time taken to reach the target error drops significantly from 150 epochs to 116 by adding a 2nd order term and then decreases slightly to 113 by adding higher terms.

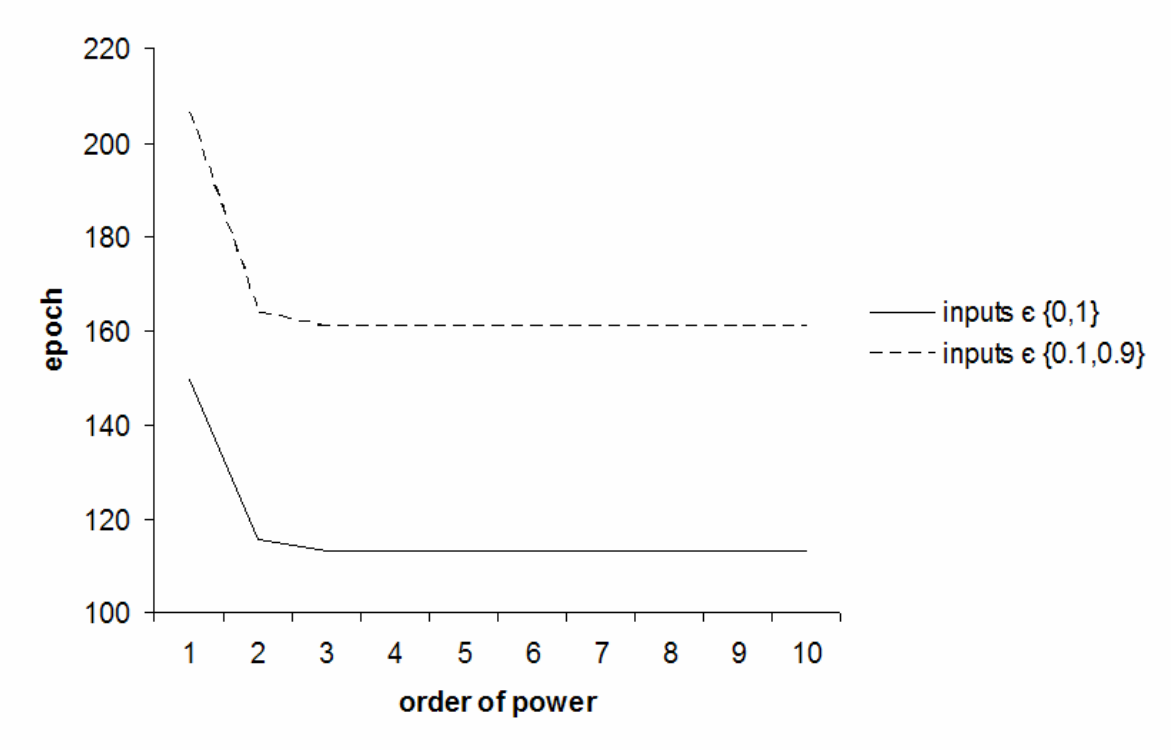


Figure 5.31c – Comparison of epoch vs. power for SLP and SLT networks
network targets ϵ {0,1}

The experiment was repeated with a constrained input set of {0.1,0.9}. This both increased the difficulty in separating the patterns, (as the previous input values of 0 have no effect on weight calculations or training), and allows the different Taylor Series (orders of power) non-linear operation. Training time increased to 207, 164 and 161 epochs respectively. The performance in the experiments is summarised in figure 5.31c. The order of power of 1 indicates the SLP, orders of 2 or greater indicate the SLT. The solid line shows the initial data set of {0,1}, the dashed line the data set of {0.1,0.9}. Both experiments were tested sequentially by increasing the SLT to the 10^{th} order and then progressively each 10^{th} order to the 100^{th} order and no further improvement was found.

A second set of experiments were then carried out. These involved setting the targets to {0.1,0.9}. As the previous targets are at the extreme values of the output function, they were achievable by allowing the network weights to tend to large magnitudes. These targets require a “finer tuning” of the weights and have correspondingly longer training times, as shown in figure 5.32.

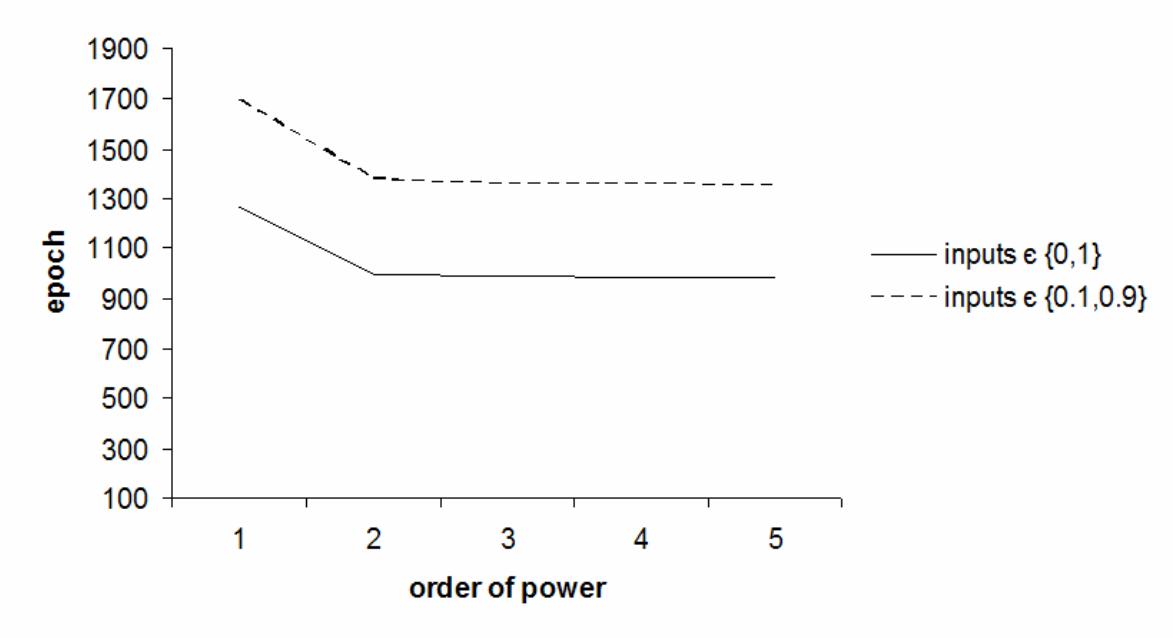


Figure 5.32 – Comparison of epoch vs. power for SLP and SLT networks
network targets ϵ {0.1,0.9}

For inputs of {0.1,0.9} and {0,1} the training time has significantly increased; however, the reduction is in line with increasing orders of power and follows the same profile as before. There is little fluctuation as the orders are increased. This may be due to initial starting values. Orders were not tested above the 5^{th} power as there seemed little to investigate.

Finally, the training pattern set of {0,1} values was replaced with the continuous values [0,1], as shown in figure 5.26a(ii). This increases the problem difficulty and produced an expected increase in training time, shown in figure 5.33. The targets were returned to {0,1}.

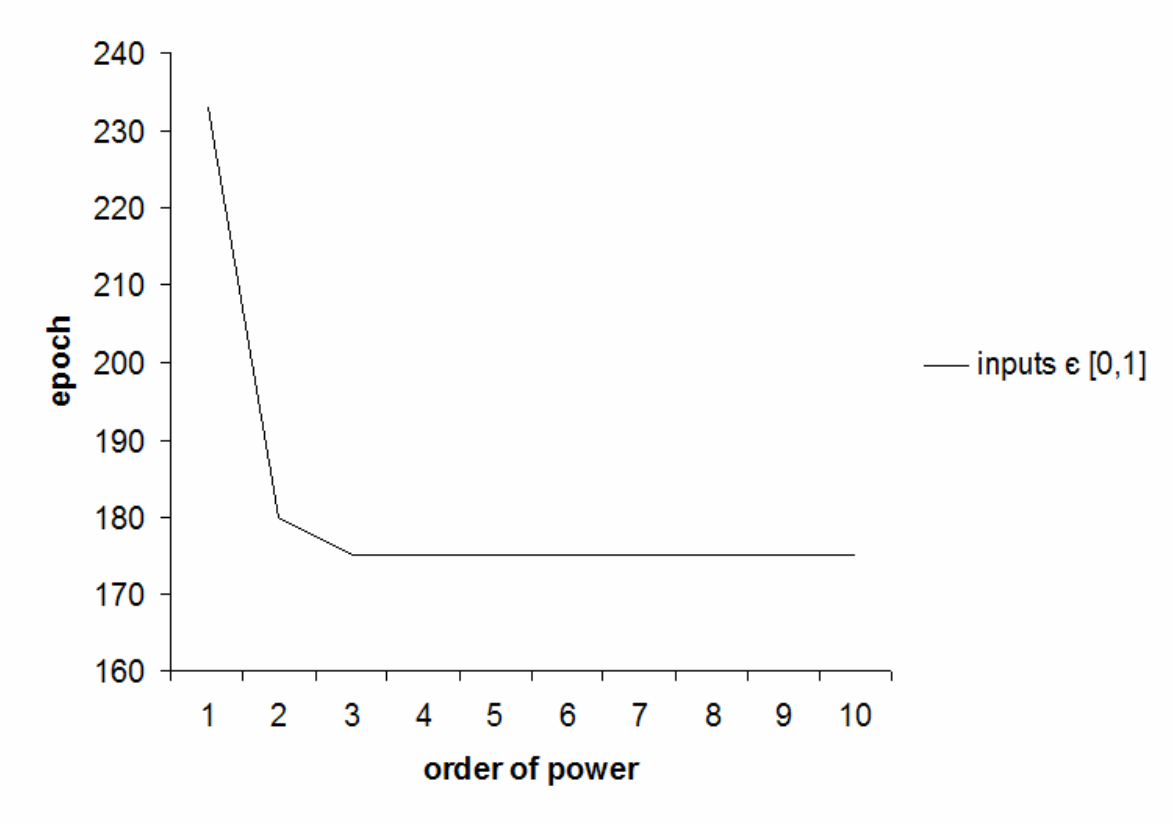


Figure 5.33 – Comparison of epoch vs. power for SLP and SLT networks
network targets ϵ {0,1}

The additional complexity of the problem increases the network training time beyond that for the input set of {0.1,0.9}, despite this set allowing some 0 value inputs. Even with the continuous inputs, the problem is still 3rd order solvable.

5.8.2 Noise Tolerance

Noise was tested at an increasing value from 0% to 24%, where 0% represents the original patterns and 24% represents the addition of a randomly generated value between 0.00 and 0.24 to every input data unit.

The networks were presented with problems which were arranged into sets as;

- data patterns figure 5.26a(i) inputs {0,1} and targets {0,1}.
- data patterns figure 5.26a(i) inputs {0.1,0.9} and targets {0,1}.

- data patterns figure 5.26a(i) inputs {0,1} and targets {0.1,0.9}.
- data patterns figure 5.26a(ii) inputs [0,1] and targets {0,1}.

Other combinations were also tested to assess if there were any unusual behaviours; however, these four tests proved sufficient for comparison.

The first pair of tests have similar results, but the alteration of the inputs from {0,1} to {0.1,0.9} reduces the noise tolerance of the SLP and the SLT for most TS orders. However, as there is a random element in the noise, anomalies do occur. The average error level for the SLT has increased as it does for the SLP. These noise effects are shown in figure 5.33a for the inputs {0,1} and figure 5.33b for the inputs {0.1,0.9}.

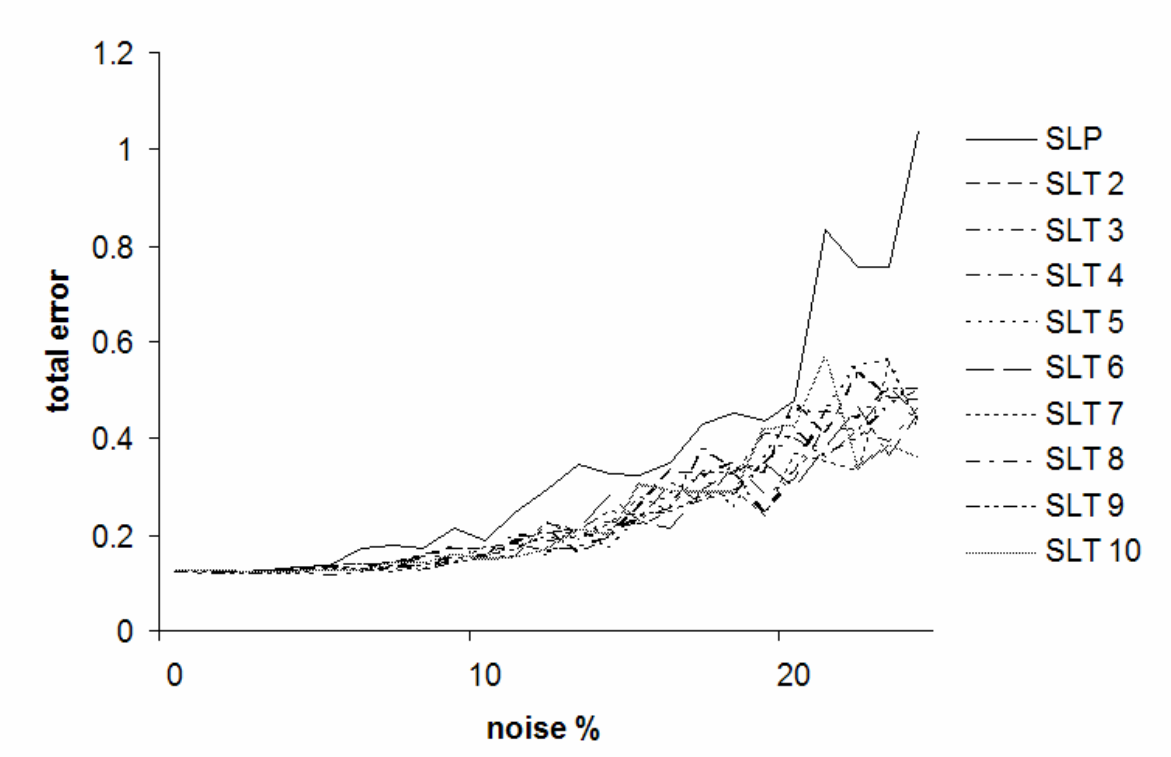


Figure 5.33a – Comparison of error vs. noise% for SLP and SLT networks network inputs ϵ {0,1}

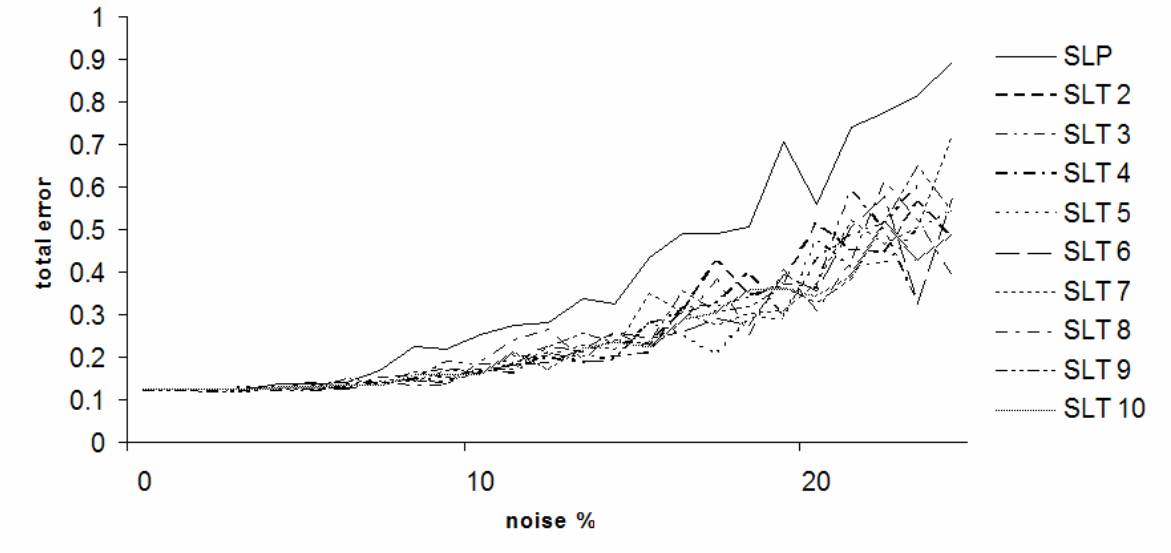


Figure 5.33b – Comparison of error vs. noise% for SLP and SLT networks network inputs ϵ {0.1,0.9}

The experiments were run multiple times to ensure that typical performances was represented. The mean value is not shown as this gives a smooth, misleading performance which is of importance later. In figures 5.33, the highest, worst performance, belongs to the SLP. The "mess" of other lines belongs to the SLT of orders 2^{nd} to 10^{th}. The individual lines of the SLT are not as important as their communal location.

It is clear that the addition of orders of power to the units improves the noise tolerance of the network. The lines represent the total error for all patterns at a particular noise level. The SLP error can be seen to be clearly greater than the SLT error. The effect of this noise is not evenly distributed and all orders of SLT and the SLP recognise roughly the same number of patterns over all noise levels and only misrecognises between 4 and 6 patterns out of 650 presentations. There is one exception; the SLT of 8^{th} order recognised all patterns. These occurrences are mainly due to the random element in the noise generation. When the data set {0.1,0.9} is used, despite the slight rise in error, pattern recognition improves significantly for the SLT, which reduces to 3 missed recognitions; however, the SLP rises to 7 missed recognitions. This may be due to the effect of 0 and 1 inputs to the TS neuron. As the TS neuron implements power terms, these values are non-applicable for 0 inputs and act as a second weight; however, they are still a linear operation for a 1.

The second test shows an unusual effect in SLPs and SLTs. The inputs are returned to {0,1} and the targets are set to {0.1,0.9}. As previously reported, the training time increases dramatically as the networks try to fine-tune the weights to these targets. In the previous reported examples the average error across all added noise for the SLP was in the region of 0.35 and 0.38 for the two data sets; it now rises to 0.43. For the SLT, the average across all powers was 0.23 and 0.25 - it now rises to 0.41. This is what would be expected for over-fitting. However, the error rise is smooth, as shown in figure 5.34.

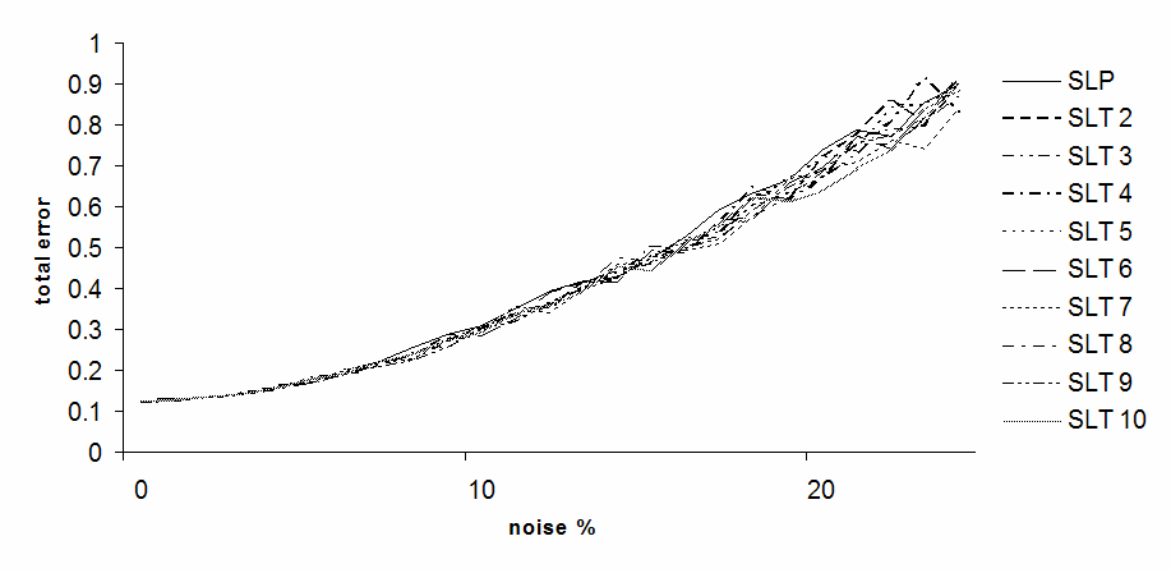


Figure 5.34 – Comparison of error vs. noise% for SLP and SLT networks
network inputs ϵ {0,1} - targets ϵ {0.1,0.9}

Comparing the smooth behaviour of this test to the erratic behaviour displayed previously does not show any apparent advantage. If the number of patterns the SLP and the SLT are able to recognise is tested, the result is that both networks recognise all patterns at all noise levels. This is excellent for avoiding the effect of noise in networks in general, however, it is so effective that it does not allow a direct comparison between the SLP and the SLT. Due to this, the targets for the final test are returned to {0,1}. It should be noted that this experimental performance contradicts much of the theory researched on polynomial over-fitting. This is attached as Appendix D.

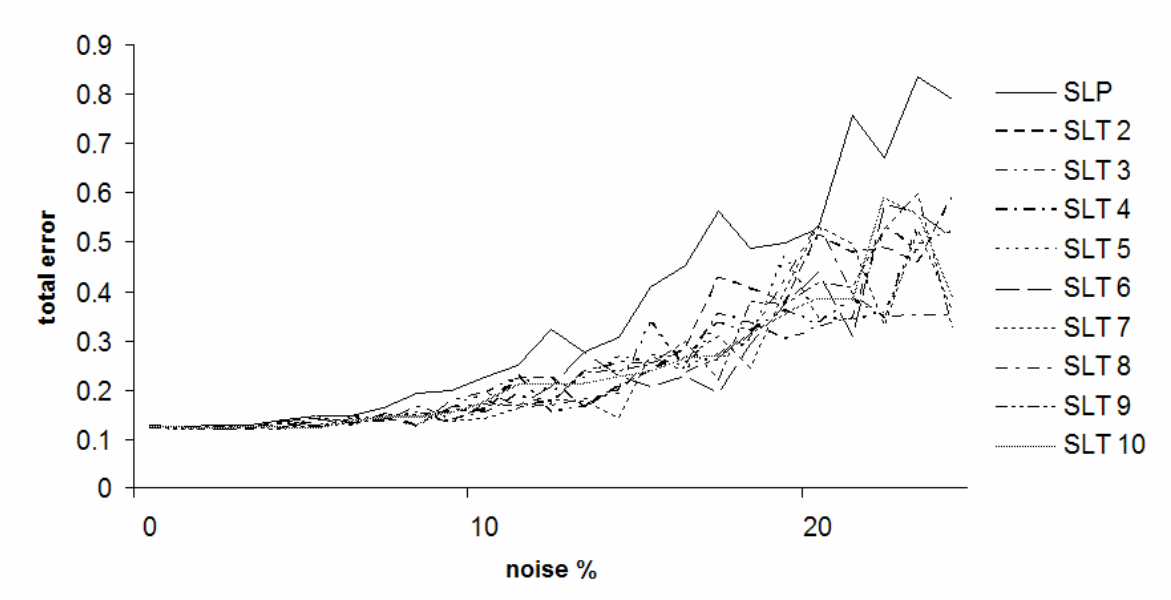


Figure 5.35 – Comparison of error vs. noise% for SLP and SLT networks
network inputs ϵ [0,1] - targets ϵ {0,1}

The noise profiles of the SLP and the SLT appear similar whether using the data set with continuous data [0,1] or the constrained data {0,1}. The SLP has an average error of 0.36, (which is between the previous performances on data sets {0.1,0.9} and {0,1}) while the SLT has an average of 0.24 (which is between its previous performances on the same data). As regards the number of patterns recognised, the SLP performs slightly poorly compared to the SLT, with 10 missed recognitions. The SLT has an average of 4 missed recognitions.

The continuous data set [0,1] was examined with targets of {0.1,0.9} and showed similar behaviour to the constrained data set {0,1}. However, there was a rise in average error values. Although it was possible to generate misrecognitions in the SLT by testing with noise, these occurred rarely and in general the SLP and SLT recognise all patterns.

5.9 Second Experiment : Comparing McCulloch-Pitts MLP and Taylor Series MLT – 3x3 test

The McCulloch-Pitts based network of figure 5.29 and the Taylor Series based network of figure 5.30 were trained on a 3x3 grid size.

The error, the number of hidden neurons required to achieve satisfactory performance, and the associated number of training epochs were assessed. This was to confirm that the performance of the TS neuron can be extended into a MLT. The number of problem parameters is reduced to simplify the comparison, hence the smaller grid size used. The only new assessment is in the number of hidden neurons required.

The training method employed is a Genetic Algorithm. Training time is measured in number of generations required to reach a target error. The size of each individual in the GA is determined by the number of parameters in the network. This means that a MLT will require a larger GA than a MLP. The parameters of the GA are shown in figure 5.36.

Parameter	*Value*
String size	One floating point number per weight
Population size	100
Crossover	Random 10 point max
Mutation rate	Uniform random 1%
Mutation	Uniform random ±5
Selection	Roulette

Figure 5.36 – Genetic Algorithm - parameters

5.9.1 Training Time

The MLP was of a fixed size. When the problem was consistently solvable by the MLP, the MLT was used and various power orders were applied to the network. The effect of this against training time is shown in figure 5.37.

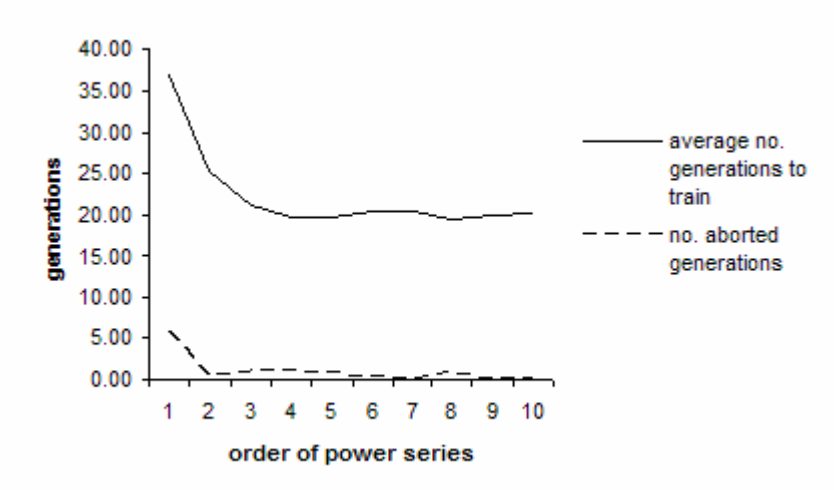


Figure 5.37 – Comparison of generations vs. power for MLP and MLT networks

This network was three layered, consisting of 9 inputs and 11 neurons configured for character recognition, as 5 hidden neurons and 3 output neurons. It can be seen that there is little point in introducing orders above the 3rd. Although the training epochs decrease, the computational power required for training increases - in the case of the 3rd order neuron, by three times. However, there is still a net improvement in training time. These results were reported by Capanni et al. [2003]

5.9.2 Size of Network

When used in a standard pattern recognition system, the use of the higher order neurons allow the system to operate with fewer units, as shown in figure 5.38.

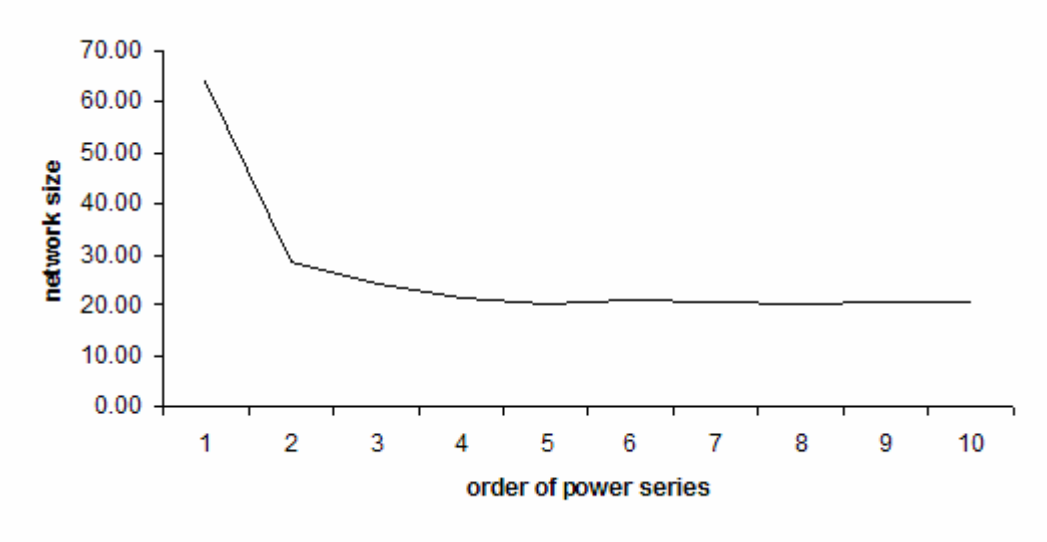


Figure 5.38 – Comparison of size vs. power for MLP and MLT networks

It can be seen in both cases that above the 5th order, performance shows little improvement. Indeed, there may be disadvantages in using too many orders, [Bishop, 1995a]. In this case, the reduction in number of neurons is offset by the increase in the multiply and accumulate instructions required for a more complex network. These results were also reported by Capanni et al. [2003].

5.10 Third Experiment : Comparing McCulloch-Pitts MLP and Taylor Series MLT – 5x7 test

The McCulloch-Pitts based network of figure 5.29 and the Taylor Series based network of figure 5.30 were trained on the 26 patterns shown in figure 5.26a.

This was to confirm that the TS neurons operates as expected as the problem domain becomes more complicated.

A MLP, shown in figure 5.29, was presented with the pattern in figure 5.26a(i). The network trained sporadically with 5 hidden layer neurons. This is a well researched network and it is known that the starting parameters can affect whether it successfully trains with this size of hidden layer. If the size is increased to 6 neurons the training becomes consistent.

From the experiments in the previous two sections, it appears that for this data set, using a TS neuron of 3rd order achieves the maximum benefit in generalisation and training time. This order of TS neuron is applied to the MLT network shown in figure 5.30 using the data of figure 5.26a(i). The MLT successfully trains on this larger data set with only 4 hidden layer neurons. This is a smaller network than was found for the MLP trained by either the GA or with Backpropagation.

5.11 Summary of Network Comparisons

An examination of Taylor Series networks has produced some interesting results. When compared against single-layer McCulloch-Pitts networks, using Delta Rule training, the performance of the two networks follows a similar path of improving error verses epoch count, as shown in figure 5.31a. Interestingly, increasing the order of the TS neuron has little or no effect in the early stages of the error profile. The conclusion is that during this time the network is improving its performance through the use of linear separators and the higher orders are unable to present an advantage. Once training slows down for the MP network it becomes progressively harder to solve the problem with linear separators, and the TS higher orders show an advantage (figures 5.31b and 5.31c). This is why the SLT,

2^{nd} order, only shows an improvement over the SLP at low errors. Following the same reasoning, the TS 3^{rd} order shows no improvement over the 2^{nd} order until the error is further reduced and the problem becomes more difficult to solve with a combination of 1^{st} and 2^{nd} order separators. The reason no improvement is shown by adding higher orders is that the network does not require them to solve the problem and in this case, the transition between 3^{rd} and 4^{th} order does not occur.

In the problem presented, increasing the order of power reduces the number of training epochs required up to 3^{rd} order, when no additional advantage is gained with further increases, (see figure 5.31c).

If the SLT is implemented sequentially, in software, then there are calculations for each increase in power and so the processing overload is greater for the higher order power even given the reduced number of epochs. In a parallel hardware implementation this would not be the case and the higher orders would have a speed advantage. It may be the case that a gradient descent learning algorithm, specifically derived for a TS network, could impart an advantage to the software implementation that reduces the training cost.

Adding TS powers improves noise tolerance, (see figure 5.33a). In the problem presented there is little gain in increasing the power beyond the 2^{nd} order. The SLT advantage in noise tolerance occurs in terms of a lower error on all noise levels and a similar pattern recognition capability to the SLP, as reported. Once the TS neurons are allowed to have inputs which provide a non-linear function, (see figure 5.33b), the effect of the noise tolerance results in better pattern recognition compared to the MP neurons SLP, as reported. This is to be expected as a smooth separator provides a better optimal fit, in data space, than the piecewise separator produced by a network of first order (perceptron) units.

Once the targets set for the networks were altered as in figure 5.34, it was observed that the non binary targets resulted in the network better fitting the problem and producing a result intrinsically more noise tolerant, without validation training. The conclusion is that this may be due to the fine tuning of weights, giving a more robust network.

When a comparison of neurons is performed using a multi-layer network, the TS neuron shows the same advantages as outlined above, in training time. Additionally, and

importantly, the TS MLT can recognise the same number of patterns with fewer neurons than the MP MLP. This shows an increase in universality of the TS neurons over the MP neurons (it may be argued that this is due to the greater number of weights associated with TS neurons). Some of this work has shown that with increasing pattern complexity, the TS neurons retain this advantage over MP neurons. This work was done using a GA and the development of a more appropriate gradient descent algorithm would allow greater examination.

In summary, the Taylor Series neuron has demonstrated better generalisation abilities than the McCulloch-Pitts neuron by consistently performing better in noise tolerance tests. Additionally, it showed an advantage in the number of training epochs required. There is the strong suggestion that these advantages can be improved through specific training algorithm development.

5.12 Time Domain Problems

So far in this thesis discussion has centred around the use of neurons in the spatial domain (for example the recognition of a stationary pattern). However, such abilities are not sufficient in many applications. Consider, for example, a neural network which has to produce an output which controls the legs of a robot. Such a neuron must have outputs which vary with time (in order, for example, to raise or lower the legs in the correct sequence). Of course, this is exactly analogous to the neurons that control leg movements in animals. The neurons so far discussed can only produce such time-varying outputs if part of a complex network with internal feedback paths. However, as is well known, biological neurons themselves produce time varying outputs and therefore operate quite differently from the artificial McCulloch-Pitts-derived models. This has spurred researchers towards modelling such neurons (often called spiky neurons) as discussed in the next chapter. Therefore, no thesis on neuron functionality could be complete without a discussion on time-varying neuron models and this is the subject of the second half of this thesis (from Chapter 6 onwards). However, before embarking on that route, first consider whether the power series neurons that have been discussed in this chapter can serve as a template for time-varying behaviour.

5.12.1 Pulse Generators

To produce time-varying signals a small oscillatory network was employed. The network used a pair of neurons arranged in a recurrent system, (see figure 5.39). The outputs from each neuron, at time *t*, become the inputs to the cross-connected neurons at time *t+1*. The TS produces an output based on incrementing the order of expansion terms, based on the signal received at time t_n. This results in a declining effect of signal over time, without the requirement for a leaky-integration summing function.

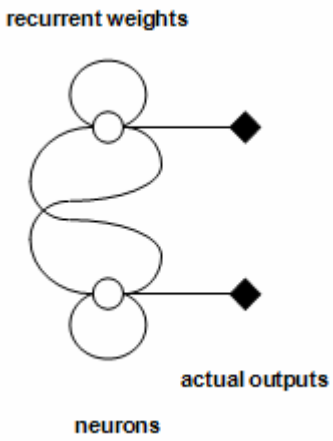


Figure 5.39 – Neural Oscillator

The network was given the task of producing outputs that mimicked specific wave-forms. The first experiments were based on producing simple exponential decay in the unit, (see figure 5.40). This was chosen as it was simple, not cyclic, useful as a basis for other functions, and as it was known that the output would initially have to rise to the peak value and provide a rough approximation of a biological action potential.

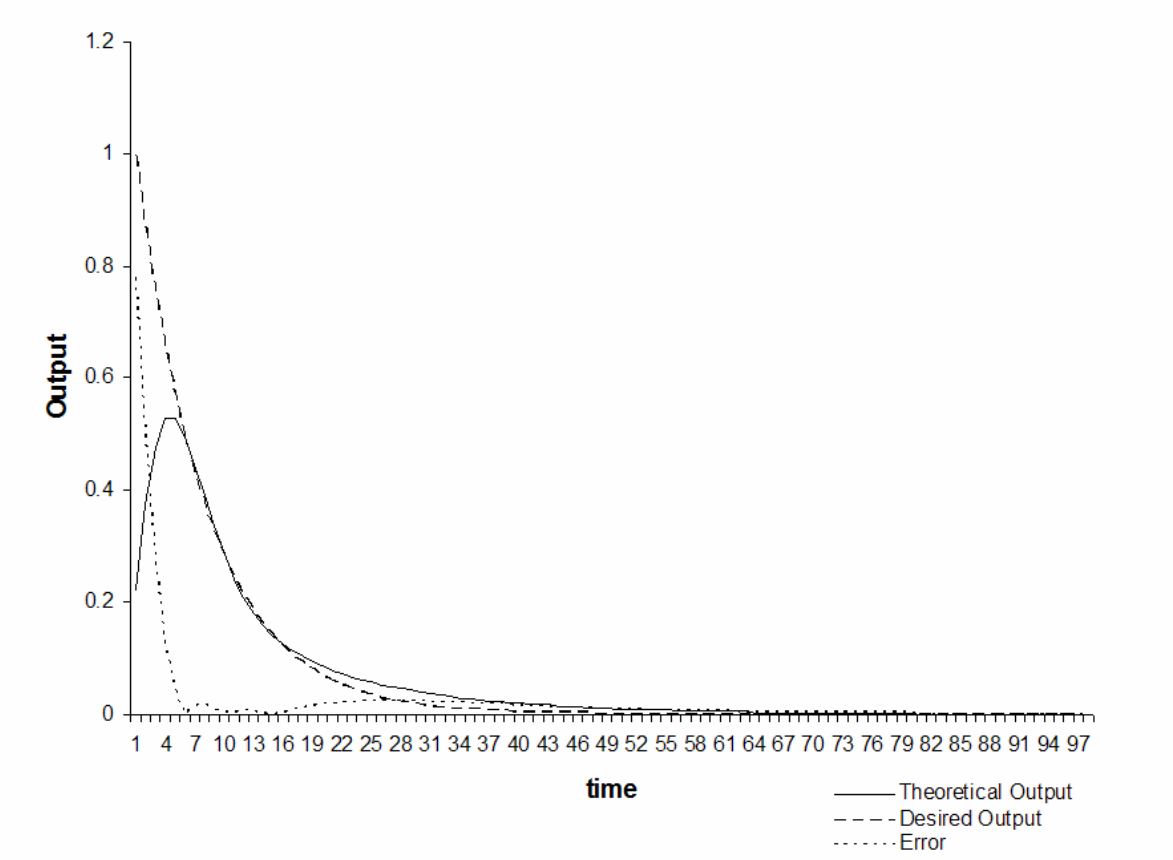


Figure 5.40 – Time-Series exponential decay – theoretical output

The units were altered incrementally based on the coefficients of their Taylor Series to give the best performance of the network, (see figure 5.40), which shows the output from one of the neurons. The desired exponential decay is shown with the actual best performance neuron output. The action potential-like performance produces a resultant error in the first time units that decays rapidly; any further improvement relies on reducing this time period. The result of this experiment was then applied to the network using a GA, which evolved to give the solutions in figure 5.41.

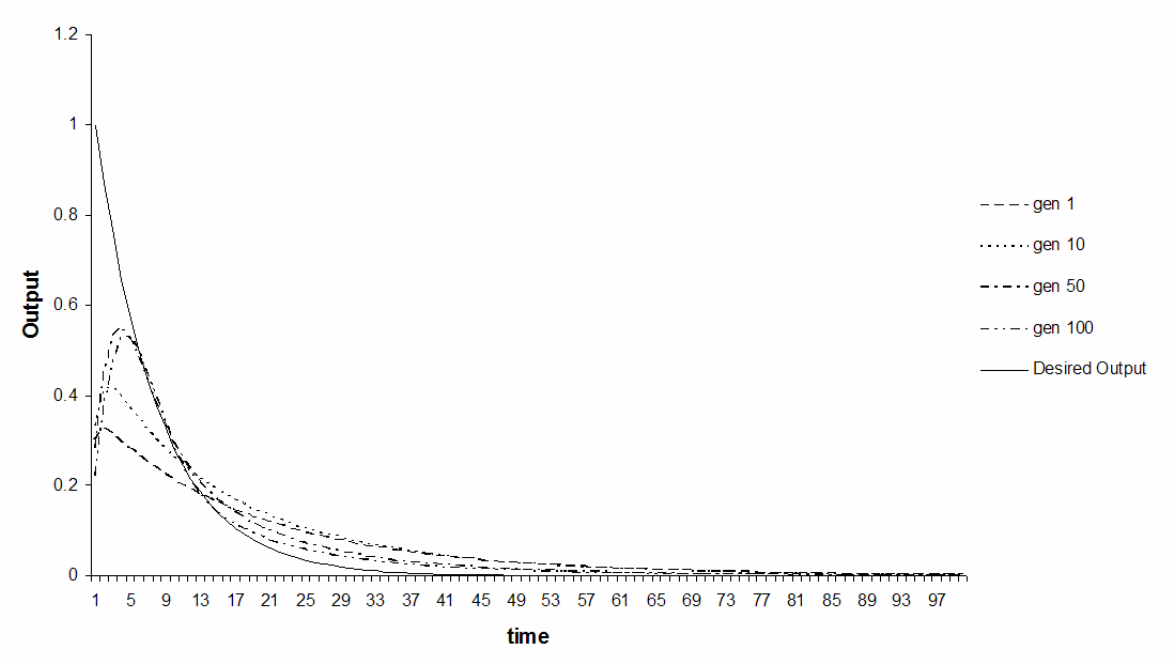


Figure 5.41 – Time-Series exponential decay - achieved

The network can produce a reasonable approximation of the desired output (a single neuron output is shown). However it was unclear as to how variation could be introduced in to the system to allow different pulses to be generated or different wavelengths to be produced, other than by training them into the network using a GA. However, such a network still could not respond to changes in its input signal.

To test the flexibility of the system, various pulses were trained with different target sets. The network topology and functionality were not altered. The performance was reasonable in production of triangular, square and sinusoidal waves, all of which are useful in walking gaits. The sinusoidal pulse is shown in figure 5.42.

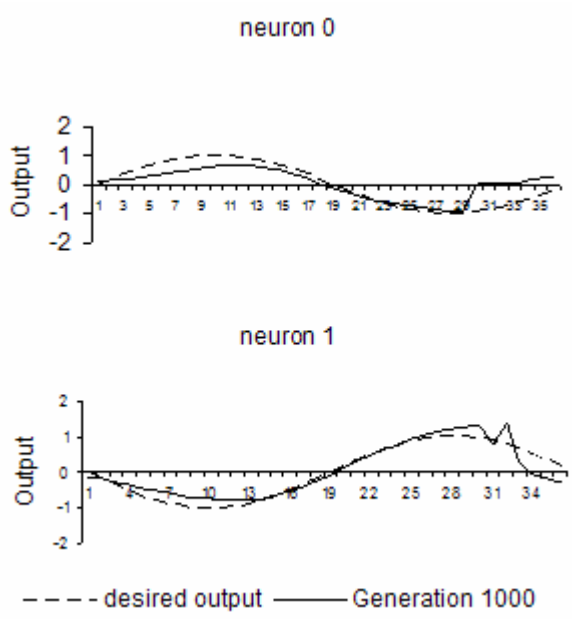


Figure 5.42 – Time-Series sinusoidal wave

Despite the reasonable performance, the evolution of such outputs required a considerable training time and the network was only capable of producing a single output without being retrained.

5.12.2 Summary of Time Domain Problems

The oscillators produced in these experiments were functional but limited. They lack convergent control (the ability to respond to inputs and produce different outputs). They can produce a time-varying signal, however they require to be reset each time they complete a pulse or as the signal degrades.

In summary, the Taylor Series neuron has proved highly capable in the spatial domain, in terms of generalisation and universality. Attempting to adapt it to time-domain behaviour produces interesting effects but the behaviour had an artificial quality and lacked adaptability. It may be possible to develop this with further work, although how to do this was not readily apparent.

From the difficulties experienced with time-domain behaviour, a new approach was sought and this led to the investigation into single-celled intelligence and the development of the Artificial BioChemical Networks outlined in the next chapter.

5.13 Literature Search of Other Highly Functional Neuron Types

There are many Artificial Neuron types that attempt to increase the unit functionality through alternative mapping functions. These include: Polynomial Neurons (GMDH), Product Units, Second-Order Neural Networks, Higher-Order Neural Networks, Sigma-Pi Units, Functional Link Units and Radial Basis Function Units. Other types exist, but these are the major units with some similarity in application or function to the approach presented here.

These methods have similarities to that of the Taylor Series neuron presented in this chapter. The similarities are mainly in the use of polynomial terms. The differences are related to interaction between the inputs, which results in the Curse of Dimensionality and restrictions on connectivity.

Polynomial approaches are based on the work of Ivakhnenko [1968], [1971], who produced "The Group Method of Data Handling" (GMDH) as a rival to the method of stochastic approximations. This was before Backpropagation had been introduced as a method of training multi-layer networks, and caused a brief revival in specific ANN research just before its decline due to Minsky and Papert [1969]. GMDH uses familiar ANN terminology and topology but has many differences as its conception pre-dates the modern expanse of research in ANNs. For example, each GMDH neuron has two inputs and its output is a six weight quadratic combination.

Since that time the terms "polynomial neural networks" and "GMDH" have become interchangeable. There have been several commercial applications, such as those of Barron Associates Inc [2005]. Barron Associates Inc was founded by Roger L. Barron who with his son Andrew R Barron contributed extensively to research on polynomial networks.

"Product Units" were introduced by Durbin and Rumelhart [1989]. In a two layer representation of this network, the hidden layer is usually replaced with product units and

the output layer remains as a summation unit. This can be applied to most problems that are solvable by Backpropagation trained feed-forward MLPs.

There are size advantages to this, as shown by Engelbrecht and Ismail [1999], who demonstrate that a quadratic function of the form ($ax^2 + c$) can be produced by a product unit network of 1 hidden and 1 output unit, (a MLP requires 2 hidden units to do this). They show this to be an increased "information storage capacity", as was also demonstrated with the TS MLT in this chapter. They also note the problems with gradient descent algorithms, due to the "turbulent error surface", and suggest algorithms including Particle Swarm Optimisation, Leapfrog and Genetic Algorithms, as they are global optimisers. This was found to be the case with the GA trained TS MLT.

Second Order neurons introduced by Giles & Maxwell [1987] allow a simple interaction between inputs and do not necessarily express a power term. These can be regarded as a type of polynomial neuron. An illustration of this can be shown by the summation function $y(x) = f(w_1.x_1 + w_2.x_2 + w_3.x_1.x_2)$. The second order unit is not restricted to such simple operations and can increase in complexity. It is used in applications such as Animat control by Crabbe and Dyer [2001] and has attracted interest in specific training methods that could assist other similar neurons, [Milenkovic et al., 1996].

There is a considerable difference in expert opinion on the uses of polynomial networks compared to other advanced neuron types. For example, Duch and Jankowski [1999] favour periodic and localised functions over polynomials, "For that reason we are quite sceptical about the use of orthogonal polynomials as output functions [Qian et al., 1990], [Chen, 1991], for high dimensional problems.", citing Barron [1993], amongst others as evidence. Barron, however is a strong proponent of polynomial networks and has produced a great deal of work, much through the previously cited Barron Associates Inc.

The universality of polynomials is in far less dispute; Nikolave [2003] shows this, "These PFNNs are … for their universal approximation abilities according to the Weierstrass theorem", citing Cotter [1990] who uses the well known Stone-Weierstrass theorem. This is supported by Bishop [1996], "that it can approximate any continuous mapping to arbitrary accuracy provided the number M of hidden units is sufficiently large."

Other related work has been investigated in the use of Higher Order, Sigma-Pi units which are explained by Bishop [1995b], who generalises second order units as having interaction between x_1 and x_2 but not having the power of either. As well as Functional-link networks Pao [1989] who makes the distinction between higher order terms in those that represent joint activations against those that, through functional expansion, increase the dimensionality.

The Taylor Series neuron presented in this chapter can be used as a single type within a network and it does not need supporting neuron types. There is no restriction on the number of inputs each neuron takes in comparison with similar topology networks and this reduces the design overheads of networks. This allows the TS neuron to be implemented in a modular style where the inclusion of a unit or connection does not affect the network as a whole. It also allows object style programming techniques to be more easily used and results in simpler implementation.

There is no interaction between the inputs of a Taylor Series neuron as this would results in the Curse of Dimensionality. Increase in orders of power have a summation increase in weight requirements. This is in part the reason why other neurons types restrict the number of inputs they take.

Taylor Series neuron coefficients can allow it to perform as a linear McCulloch-Pitts style neuron if linear separation is required, and implement increasing orders of power as the problem solution requires.

While some of the other neuron types are suited to global training algorithms, the SLT was easy to train with a modified Delta Rule Algorithm.

Chapter 6
Artificial BioChemical Networks

6.1 Introduction to the Chapter

This chapter considers the problem of producing time-varying behaviours in artificial neurons. Following on from the previous work, existing artificial neurons designed for time-domain behaviour are examined and their disadvantages highlighted. Then biological sources are re-examined; these lead to a consideration of environmental intelligence as expressed in single-celled organisms. From this initial inspiration, a new approach is developed - this is called the Artificial BioChemical Network.

6.2 Spiking Neurons

The various models of spiking neurons are the obvious units to consider when trying to produce an artificial time-dependent network. These attempt to emulate the signal spike or action potential which occurs when a biological neuron fires. The original model and the basis for many subsequent models is the "Hodgkin-Huxley Model" [1952]. The publication of this model achieved a Noble Prize for its authors. These neurons may be regarded as the third generation of neuron models. The first generation being the McCulloch-Pitts threshold logic units, and the second being the models employing a continuous activation function such as the sigmoid models. Many spiking neuron models have been developed but there are two frequently implemented models, which have their own sub-models based on the "formal spiking" or "general spike-response" model proposed by Gerstner [1994]. They are called the "spike response" model [Gerstner, 1995] and the "integrate-and-fire" model which incorporates the work of Stein [1967].

6.2.1 Biological Spiking Neurons

The original paper that began neural network research by McCulloch and Pitts [1943] was an attempt to produce mathematical algorithms to model the activity of neurons. It assumed that a neuron either fired or did not, and so the output was binary. Later

developments allowed the use of continuous functions, some of which are described in Chapter 4. Neither method took account of time as part of the encoding of information and it was due to biological evidence that later spiking models were developed to model time factors.

It is known that biological neurons fire at various rates, between their maximum and minimum frequencies, depending on stimulation [Maass, 1997]. The stimulations may be excitatory or inhibitory.

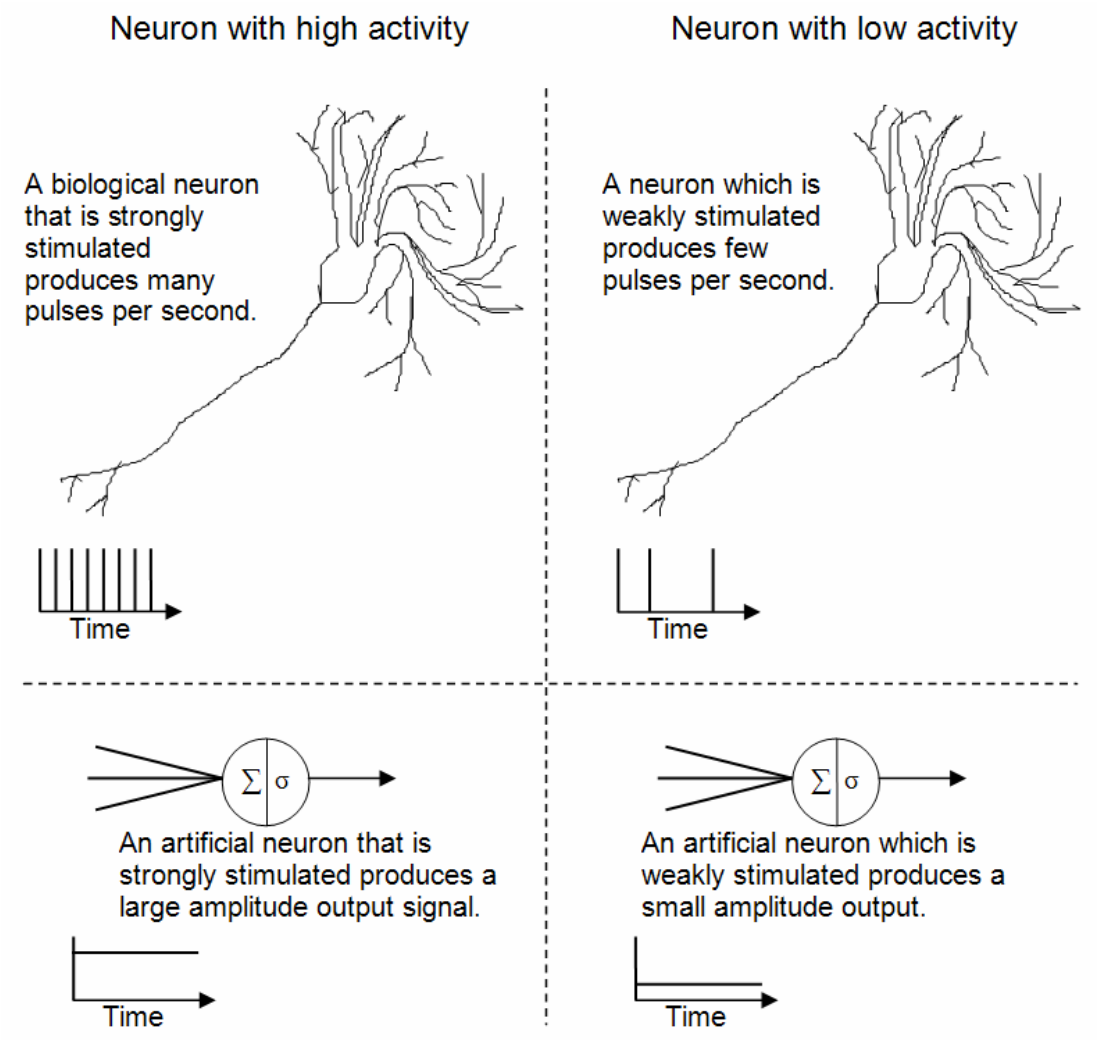


Figure 6.1 – Biological and Artificial Neurons

adapted from MacLeod [2004]

6.2.2 Hodgkin-Huxley Model

The Hodgkin-Huxley model is the model on which many artificial spiking neurons are based.

In this model, action potentials result from currents passing through ion channels in the biological cell membrane. Hodgkin and Huxley performed a series of experiments on the giant axon of the squid; they succeeded in measuring these currents and described their dynamics in terms of non-linear ordinary differential equations. Good descriptions are given by Vreeken [2003] and Gerstner and Kistler [2002a].

Hodgkin and Huxley's model was based on these experiments. They found that three different types of ion current were present in the axon; sodium (Na^+), potassium (K^+) and chlorine (Cl^-). The flow through the sodium and potassium ion channels is voltage-dependant, while the chlorine leakage current is assigned for all non-specifically described channels. These are represented as conductance-capacitance circuits in electronic engineering and simulated as such in software. Figure 6.2 shows a representation of the ion and circuit diagrams.

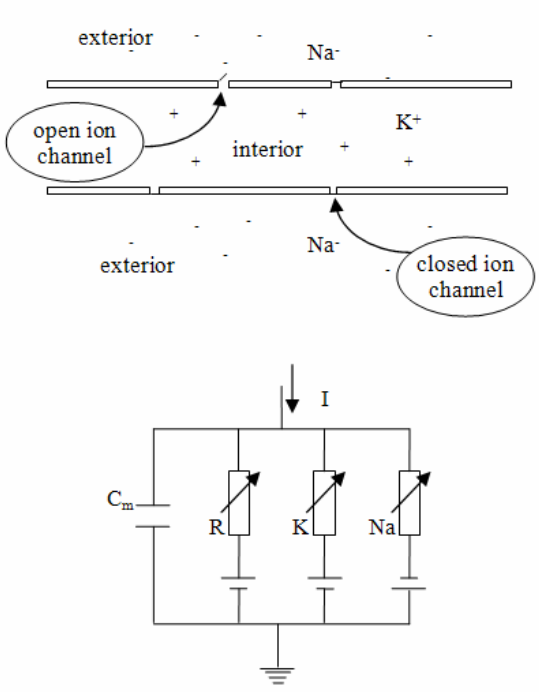


Figure 6.2 – Hodgkin-Huxley model

The value C_m is the capacitance of the membrane, the ion channels are represented by K^+ and Na^+ and the total leakage attributed to all other ions is R. This is a simplified diagram of a spiking model and is specific to the Hodgkin-Huxley approach. A full explanation is beyond the requirements of this thesis; however, the basic operation and equations are included below. In the model the ion flows are derived separately over time and the channels are expressed as resistance or conductance and capacitance of the circuit. This model has been pursued, and is in current use as a method of studying biological neurons. It relies on equating a stimulus (introduced current) with its effect on the various ion flows.

A summary of the operation of the model is as follows;

- An input current $I(t)$ is introduced in to the cell.
  This current causes an increase in charge across the capacitor C_m, and can leak out, through the channels in the cell membrane (represented by the channel resistances).

This gives a capacitor current I_C and the current I_K, the ion channels' component. This is expressed as;

$$I(t) = I_C(t) + \sum_K I_K(t) \qquad \text{equation 6.1}$$

In equation 6.1, ΣI_K is the flow over all ion channels. As capacitance is the amount of charge stored across an electrical potential, equation 6.2.

$$C = \frac{Q}{V} \qquad \text{equation 6.2}$$

- The charging current I_C can now be expressed as follows;

$$I_C = C\frac{\partial V}{\partial t} \qquad \text{equation 6.3}$$

- Combining this with equation 6.1 gives;

$$C\frac{\partial V}{\partial t} = -\sum_K I_K(t) + I(t) \qquad \text{equation 6.4}$$

The voltage V is the membrane potential. The Hodgkin-Huxley model describes three types of channel, which are characterised by their conductances (g_L, g_{Na}, g_K) and reverse potentials (E_L, E_{Na}, E_K). The leakage conductance g_L is voltage independent, while the conductance of g_{Na} and g_K vary with voltage and time. The conductance and reverse potential parameters are empirical parameters.

The operation of the model is controlled by gating variables that represent the probability that a channel is open. The (Na^+) channel is controlled by the actions of m and h; the potassium (K^+) channel is controlled by n.

$$\sum_K I_K = g_{Na} m^3 h(V - E_{Na}) + g_K n^4 (V - E_K) + g_L (V - E_L) \qquad \text{equation 6.5}$$

The function of the gating variables and the model require further calculus; however, it is not necessary to elaborate further here. A good expansion can be found in Gerstner and Kistler [2002b].

The gating variables can be expressed so that, for a fixed voltage V, the variable $x \in \{n,m,h\}$ approaches the value $x_0(V)$ with a time constant $t_x(V)$. These are shown in figures 6.3 and 6.4.

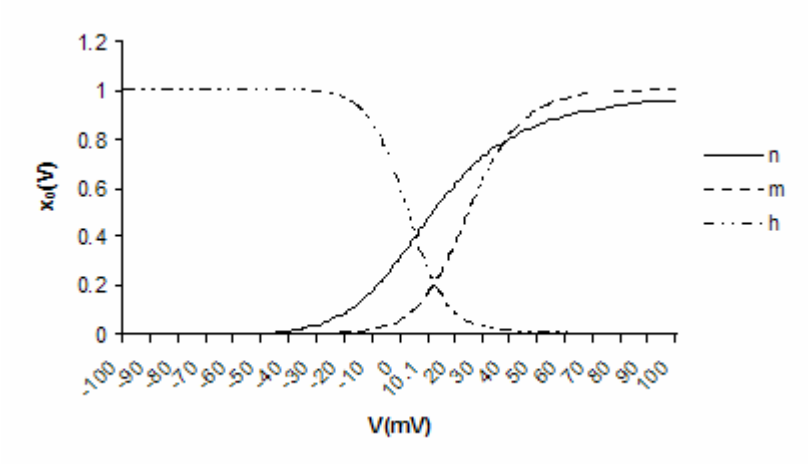


Figure 6.3 – Equilibrium function

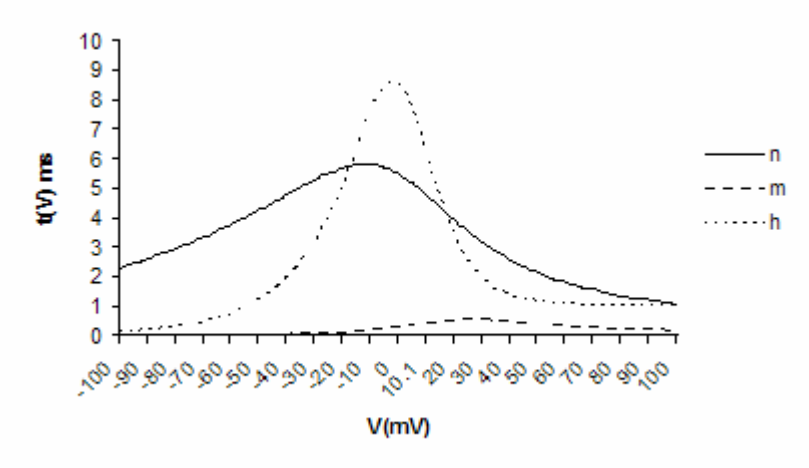


Figure 6.4 – Time constant

In the Hodgkin-Huxley model, the resting voltage was adjusted to V(0), figure 6.3. If a sufficiently large current is introduced to the system in a sufficiently short time as *I(t)*, figure 6.2, then the a spike is produced, as in figure 6.5.

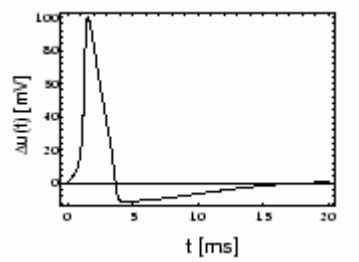


Figure 6.5 – Single spike

reproduced with permission from Gerstner and Kistler, [2002c]

It is the action potential-like spikes produced by these models that give them the name "spiking neuron".

It can be seen from these calculations that the computational resources required are large. This requirement is similar for the alternative spiking models previously mentioned. Although the signals produced by these models are biologically plausible, they lack flexibility – they only produce spikes. It is considered that a more flexible approach would be useful in engineering systems.

Having observed the limitations in operation and the difficulties in implementation of the present spiking neuron models, it was decided to look for a simpler and more flexible approach. After considering the alternatives, a review of alternative biological systems was undertaken. Only one other type of biological intelligence was obvious as a result of this reassessment and, fortunately, this turned out to lead to both flexible and simple time-varying models. This is the artificial biochemical approach described below.

6.3 Origins of Biological Intelligence

As presented by Hameroff et al., [1998], the majority of life on Earth is represented by single celled organisms. During the 3.5 billion years of the pre-Cambrian period, life was composed of only these organisms. Then, during the Cambrian period the first multi-cellular life appeared and with it the first neurons.

Amongst the single-celled life forms are a group called the Protoctists, which live in a variety of different environments including every environment that multi-celled life has colonised. Those which display animal-like behaviour are usually called Protozoa. The name literally means “first animals” and they evolved about 2.5 billion years ago.

The arrival of multi-cellular life did not render Protozoa extinct. They persisted and colonised new biological environments.

6.4 Single Celled Intelligence

Despite their primitive reputation, protozoa display remarkable abilities and behaviours, as documented by Alberts et al., [1994a]. Some have stinging darts with which they disable their prey; others have sensory hairs to feel their way about and sense the vibration of prey approaching and a few even have leg-like appendages for locomotion. They can avoid light with their sensitive eyespots and actively hunt for their food. Some even build shelters - shells with which to protect themselves from predators and the environment. They display many of the traits of intelligence.

As Protozoa have had the longest time of any non-bacterial cellular organism to evolve [Curtis, 1982] they exhibit an enormous variety of forms and behaviours. They have a range of relative sizes greater than that between a rabbit and a blue whale, even though they are largely microscopic, (mainly ranging in size from 10-200µm), [Alberts et al., 1994b] and include the most complex cells known, [Sleigh, 1989].

Protozoan anatomy can include structures that interact with the external cellular environment, receiving information through sensory bristles or photoreceptors, and moving via flagella, cilia and other appendages, absorbing food through mouth-like parts and reacting with muscle-like contractile bundles. They are so divergent in motility that this is the main method used to classify them.

All the similar actions performed by multi-cellular life-forms utilising dedicated types of cells are performed in protozoa via dedicated sub-cellular structures [Alberts et al., 1994c].

These animals exhibit intelligent behaviour in their reactions to the environment that increase their survival chances; they do so as a single cell, with no neural network to communicate. This has a significant influence on the later sections of this thesis.

6.4.1 Natural BioChemical Networks

Protozoa display the behaviours described above by means of interactions between proteins in their cytoplasm. Proteins are the chemical workhorses of the cell [Alberts et al., 1994d]. It is the cell proteins that the DNA genetic code specifies, as shown in figure 6.6. This scheme is so fundamental that it is sometimes referred to as the "Central Dogma" of biology.

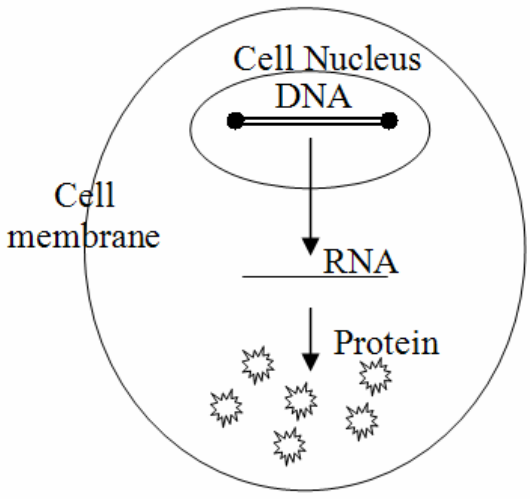


Figure 6.6 – Central Dogma

Proteins are the central part of cell biology, they mediate every biological action. As DNA cannot directly affect the organism, it may be said that it exhibits *devolved action* through the protein, [MacLeod et al., 2002]. Proteins are the biological *Universal Machines*, of the cell; they are responsible for all elements of movement, structure, communication and organisation; chemical activity is also under the control of proteins.

6.5 A Framework for Artificial Cellular Intelligence

Proteins perform all the important operations of the cell, from making new and destroying old material, to sensing and signalling changes in the cell's environment. Proteins achieve these operations through chemical interactions. All proteins bind to other chemicals; some synthesise new molecules by joining bound component parts together, others break them up - such chemically active proteins are called enzymes. Yet others use their ability to bind by joining to other proteins, changing their behaviour and forming signalling networks within the cell, as described by Alberts et al., [1994e]. Such a network is best illustrated by a simplified theoretical example - see figure 6.7.

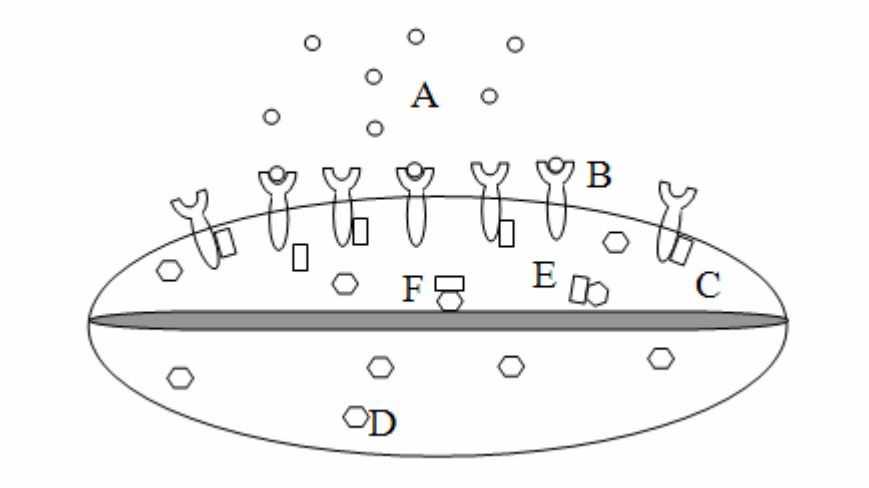


Figure 6.7 - Simplified signalling pathway

Figure 6.7 is a hypothetical example of a protein network. Molecules in the cell's external environment “A” bind to receptor proteins “B”, which straddle the cell membrane. This binding changes the shape of the receptor and causes a protein “C”, which was bound to the receptor, to disassociate from it. Protein “C” then floats freely in the cell's cytoplasm

and eventually binds with the protein “D” (chemicals in the cytoplasm are buffeted around by thermo-dynamic forces, which act to mix the constituents). When “C” and “D” are bound as shown at “E”, they can bind further to a motor protein “F”, a protein which can change its shape by a large amount, allowing it to move quite large objects. The motor protein is attached to the cell's outer membrane and this causes the cell to move towards or away from the molecules “A” by changing its shape [Capanni et al., 2005].

6.6 Artificial BioChemical Networks

Given the system described above and shown in figure 6.7, it is fairly obvious that it is possible to express such biochemical interactions as a network no different in appearance from other connectionist networks, (see figure 6.8).

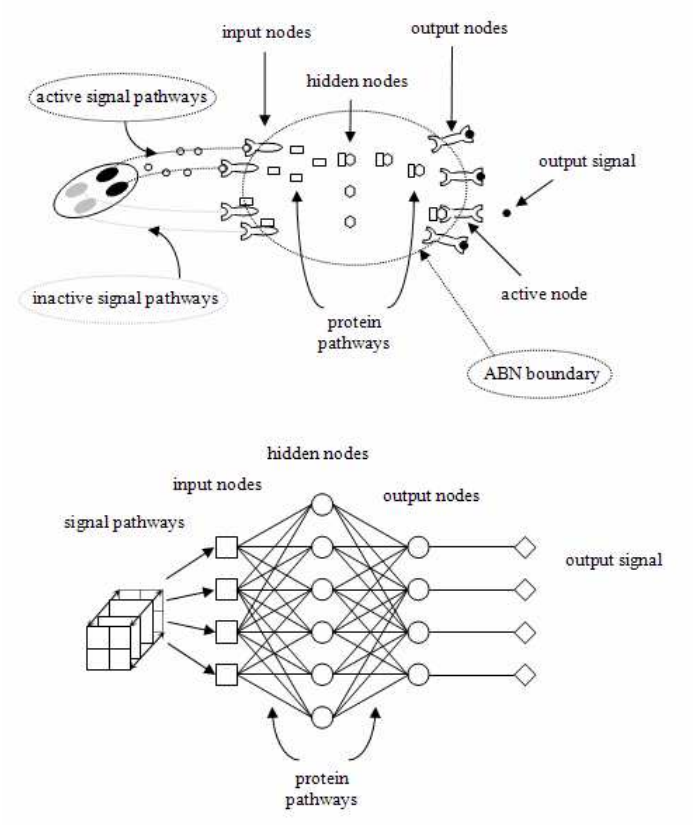


Figure 6.8 - ABN vs. ANN topology

Given this, one might present the signals in such a network as shown in figure 6.8. The lag time at a node, until the presence of the signalling protein, is “A” and time “B” is proportional to a node’s activity. Activity may be calculated using previously described

time-series techniques or a simple leaky-integrator technique as reported by Gurney [1997]. To create an Artificial BioChemical Network, “A”, “B” and the connection pathway strengths may be set using a Genetic Algorithm.

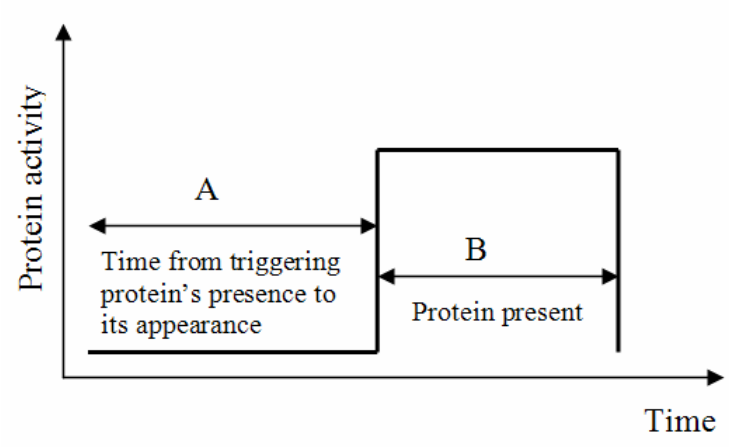


Figure 6.9 - Basic unit cycle

A GA may also be implemented to choose which of the time periods “A” or “B” is proportional (or inversely proportional) to the unit activity and which is fixed, as described previously. This additional evolvable parameter, introduced by MacLeod, C. and Maxwell, G., [2003] has been used in this thesis to produce pulse width or frequency modulated units as shown in figure 6.10.

Such flexibility allows for the production of more universal units from this basic type and it has been suggested that such dynamics may lead to the new perspectives on intelligence as proposed by MacLeod and Maxwell, [1999].

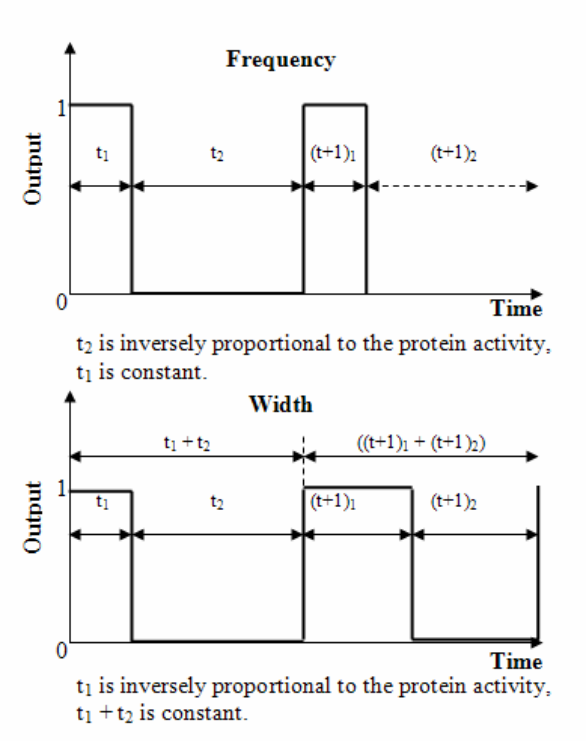


Figure 6.10 – Pulse modulated units

The units shown in figure 6.10 are an extension of the general behaviour shown in figure 6.9. If appropriate variations on this general behaviour can be produced by ABN units, then small modules of these should in theory be able to produce any pulse sequence in the binary time-domain; this would be a time-domain equivalent to the McCulloch-Pitts XOR or parity-bit problem.

The capabilities of protozoa show clearly that there are creatures that exhibit intelligence without neural pathways and it is from this observation that the ABN was developed.

This approach takes inspiration from the biochemical protein communications of the protozoa, however, it does not try to recreate them artificially. This results in a flexible approach which is far simpler to implement than spiking models. Future development can build on this initial model to add functionality through additional biochemical activity.

6.7 Literature Review on Cellular Models

There are various other models inspired by intercellular processes in the literature. They can be roughly divided into two classes, those that attempt to construct an Artificial

Intelligence system by basing themselves on rules governing cellular interactions, and those that attempt to model cellular biochemistry to better understand the operation of cells from this viewpoint. The later is frequently used to model the effect of pharmacological products or study biochemical pathways, as in Almogy et al., [2001], Hodgson et al., [2004], and Kiehl and Bonissone [2002].

The "standard Cellular Neural Network" (CNN) devised by Chua and Yang, [1988] at the University of California at Berkeley was proposed as a practical alternative to Hopfield-derived recurrent networks. It is a continuous time and state dynamical system, well suited for analogue circuit implementation. The CNN was designed to be a useful signal processing paradigm and was implemented in hardware with the advent of Very Large Scale Integration electronics.

This type of system is a logical development of Cellular Automata (CA), which were proposed by Stanislas Ulam in the 1940s and is related to Von Neumann's work dealing with self-reproduction and artificial life. In fact, Ulam suggested this framework to Von Neumann as a direction for his self-reproduction theories, [Von Neumann and Burks, 1997]. Nowadays, CAs are mainly used to prove theories or model physical processes, [Dogaru, 2003].
In practical terms, the implementation of CNNs uses equations of non-linear functions which define the cell function. Several developments have come from this work including "reaction-diffusion cellular nonlinear networks", [Chua et al., 1995] and Generalized Cellular Automata, [Chua et al., 1998], which includes CAs as an extension of the CNN.

The approaches that have approximated cells at a biochemical level have produced some interesting models and observations.

Elowitz and Leibler, [2000] propose a synthetic oscillatory network of transcriptional regulators. They base their work on observations that networks of interacting bio-molecules are responsible for the functions in living cells. However, they point out that there is poor understanding of any "design principles" underlying how such intracellular networks function. They propose examining a particular function and constructing a synthetic network based on it. The interesting part of this work, is that they construct an oscillating network, that over a period of hours, induces the synthesis of a specific protein.

This produces a noisy “artificial clock” which it is proposed may lead to the engineering of new cellular behaviours as well as a better understanding of naturally occurring networks.

Thattai and Oudenaarden, [2001] model specific stochastic biochemical reactions using noise terms in deterministic dynamic equations. They see this as a method by which they can implement threshold transfer and noise reduction simultaneously with the aim of mediating signal transfer in both artificial and biological networks.

Gontar [2004], presents Discrete Chaotic Dynamics (DCD) as networks of interacting agents capable of energy and information exchange. This appears as a biochemical equivalent to cellular automata and results in pattern generation, which Gontar characterises as emergent, self-organising behaviour. However;

> *“The mathematical structures of the difference equations we present do not include any form of time, neither classical continuous astronomic time, nor the so-called discrete time that has no clear meaning.*”

The principles used are of discrete time and space difference, resulting in artificial life systems.

The basis of much of the work outlined above is not to produce new AI models, but to better understand the biochemistry of biological systems,

> “*In the spirit of this analysis of the transcriptional regulatory networks, it is becoming possible to design artificial biological networks to implement desired functions, paving the way to new therapeutic approaches*.”, [Vandenbunder, 2001].

A small amount of work uses biological inspiration for AI models which are mainly of the CNN type. It is clear however that a consensus exists on the importance of protein function and that much future research depends on a the ability to model this behaviour.

In comparison with the proposed approach on ABNs, the cellular models that attempt to replicate cellular biochemistry are not in general concerned with Artificial Intelligence and therefore do not use the appropriate biochemistry to simulate this. Instead they attempt to

observe how the biochemistry reacts to proposed physical and chemical influences. The models that attempt to construct an AI system use neighbourhood models, which differ from previous ANN topologies by only having a multi-dimensional array layout. A variety of these can be implemented using analogue processing with continuous signal values and connecting only to neighbouring cells. They have a wide range of applications, in the same areas as continuous type ANNs. However they retain the rigid topology and connection as well as having an associated complexity of computation due to the implementation of each cell.

Chapter 7
Artificial BioChemical Networks - Design and Function

7.1 Introduction to the Chapter

This chapter presents the models of Artificial BioChemical Network units and the networks developed from them. These networks are based on the recognition and control tasks of previous research as detailed in Chapter 2.

7.1.1 Design

The artificial biochemical node is the equivalent of an artificial neuron. As described in the previous chapter, its activation represents the occurrence of a protein or a similar bio-active chemical.

The ABNs were all trained using a Genetic Algorithm. However, it should be noted that because a GA relies on randomly generated parameter values with their attendant quantisation, they have a poorer response to noise than those using a gradient descent algorithm. To allow a more direct comparison, a Backpropagation Algorithm was constructed for the pulse-width ABNs.

7.1.2 Experiments

The experiments below are presented in three main sections. They are accompanied by the implementation of Backpropagation-SLP and a Backpropagation-MLP trained on a simple pattern recognition problem - this provided a benchmark for comparison with the ABNs.

The first experimental section investigated an ABN sensitive to pulse duration (that is, pulse width modulated) as described in the previous chapter, this is termed ABN_W. The second section considers an ABN sensitivity to pulse frequency, termed ABN_F. The third section combines the two pulse modulated unit types into a "Universal" pulse modulated ABN, termed ABN_U.

Having demonstrated the network's pattern recognition abilities, it was then used in robotic control systems to produce different walking gaits. This, in conjunction with the previous pattern-recognition experiments, demonstrates the ABNs' universality. Finally, the combination of two ABNs, where the pattern recognition outputs of the first are the inputs to a second, gait-generating network, shows that the networks can be used in a modular fashion.

7.2 Pulse-Width Modulated ABNs

The signal is propagated in a feed-forward manner in the pulse-width ABN_w. This is chosen as such a network is known to be intrinsically stable.

7.2.1 Time Concepts

The ABN_w approach is based on time-dependent behaviour. As such, every node in the ABN_w is examined as if they all operate in a parallel manner. Their state at time (t) is assessed and then time is incremented to ($t+1$).

For the purpose of visualising this and implementing the code, a tick is incremented in a programme loop and is the smallest measure of time in the system. During a tick cycle, the state of all the neurons is checked and incremented. In practical terms the time periods used in tests was in the range from 10 to 100 ticks. One fundamental change to the way a feed-forward ABN_w is coded compared to an ANN is that in an ANN the neurons are assessed starting with the inputs and progressing to the outputs, whereas in an ABN_w, the output nodes are assessed and then incremented and the program works back to the inputs (as the information flows forward, this allows parallel implementation).

7.2.2 Inputs and Interpretation of Patterns

The signals used in experiments (as inputs, outputs and between neurons) were digital pulse trains of values 0 and +1 or, in some tests, +1 and -1. The inputs were repeating waveforms of the type shown below.

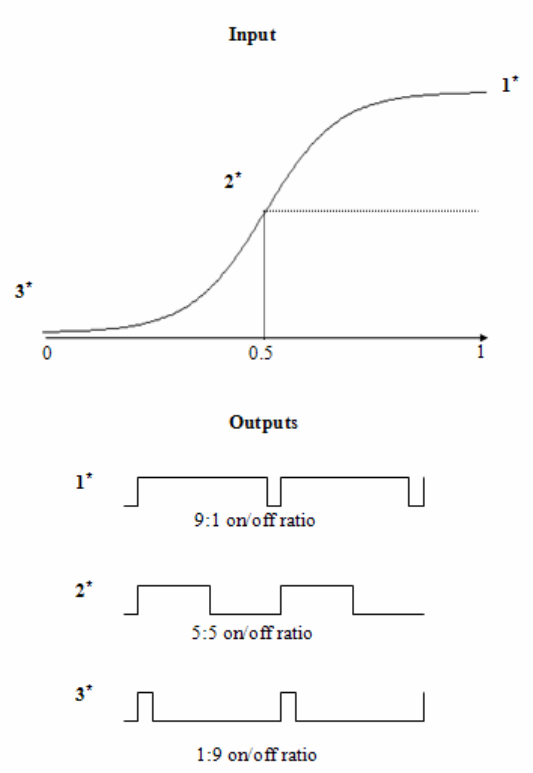


Figure 7.1 – Transfer functions – pulse-width

The patterns were left on the inputs until the ABN_w output was established.

7.2.3 Nature of the Pulse Dynamics

The pulse consists of an *on time* and an *off time*. As noted above, these have fixed amplitudes. The amplitude during the time period t_{on} is one unit, while during time period t_{off} it is 0 units.

When a constant pulse-width (PW) is implemented, it follows;

$$t_{on} + t_{off} = PW \qquad \text{equation 7.1}$$

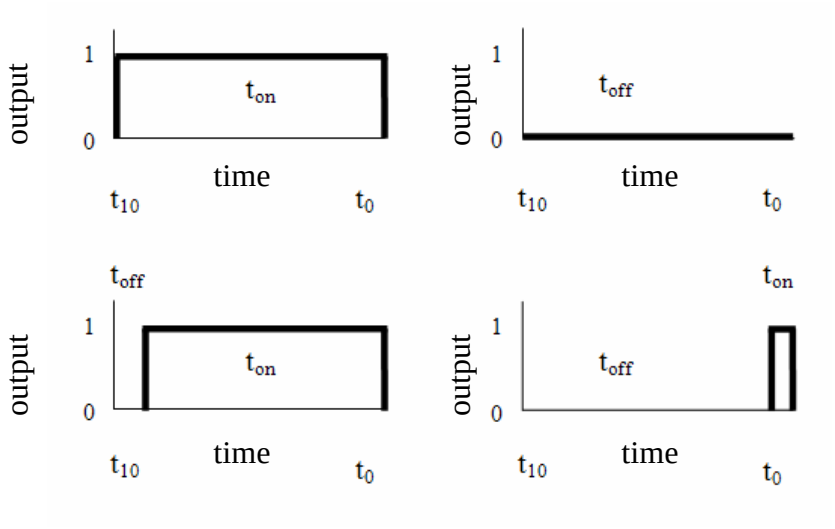


Figure 7.2 – Pulse-width max and min profiles

The length t_{on} encodes the information the node is sending. Minimum and maximum values of this set the limits of the pulse, as shown in figure 7.2. The possible interpretations are shown in pairs on each line. In the first set the entire pulse takes maximum and minimum values. In the second set the minimal pulse starts with a single t_{on} tick while the maximum pulse finishes with a single t_{off} tick.

The relationship between the time periods and the pulse width results in an automatic synchronisation of all nodes in the ABN_w that activate at initial time t (because, regardless of t_{on}, the cycle length $t_{on} + t_{off}$ is constant). This can be of benefit in many situations but is an obvious dynamic constraint. It is considered later in pulse frequency modulation whether the nodes should be capable of synchronous or non-synchronous action.

The duration of a pulse is measured in ticks, a pulse being produced at one node at time t cannot affect any other nodes until it reaches them at a later time. This means that it takes the pulse a period of time to move from the outputs of one neuron to the inputs of the next - this may be termed the “Propagation Delay”. In the initial implementation, this delay is a constant for all distances between the communicating nodes in the ABN_w. It is considered that in a biological system, the variation in this delay may have significant effects. One such effect, which is well documented, is that sound is localised by time differences between signals arriving from both ears, see Palmer et al., [2002]. Given this, in future research, this delay may be a system variable.

The implications of the delay are that the leading edge of the pulse has arrived at its destination before the firing node has finished generating it. In reality this is plausible in a biochemical interaction within a cell and between cells, however it would not occur in neurons, because of the time taken for an action potential to propagate down an axon. If this delay is not implemented, then a signal beginning at the inputs of the ABN_w would instantly propagate to the next layer, and so on from layer to layer, restricting the development of a pulse stream. The delay is further complicated in that a node completes its pulse before acting on the signal it has received - the node can receive stimulation during its pulse cycle and this contributes to the next pulse (due to the effect of the leaky integrator).

The total time it takes for the effects of an input to be observed at the outputs is called the "Reflex Time" of the network. This will equal ((propagation delay + *PW*) x number of layers). Likewise, once the stimulus is removed there will be an equivalent "Decay Time" for the stimulation at the ABN_w inputs to stop producing a signal at the ABN_w outputs.

7.2.4 Integration of Signals by the Node

The pulse duration is a representation of the strength of the weighted input signal to the node. The first stage is the calculation of node activation. This is the process by which the Sum (the node's activation value) is calculated, the Sum is represented as *S*.

There are two methods to initially consider for the calculation of *S* at time *t*. These incorporate the current weighted input values, the previous value and a leaky integration (*LI*) factor alpha (α**)**.

$$S_t = \sum i_n \cdot w_n + \alpha \cdot S_{t-1} \qquad \text{equation 7.2}$$

$$S_t = \sum i_n \cdot w_n + S_{t-1} - \alpha(LI) \qquad \text{equation 7.3}$$

These add a decayed version of the previous to the current activation, therefore they do not model the biological neuron, which when firing completely depletes its *S* value. However, they correspond more to biochemical signal integration. The equations show respectively a non-linear and a linear decay in *S* in the absence of further stimulus.

The second stage is the calculation of pulse time. The *S* values calculated in the previous section produce an output pulse of amplitude one unit and duration t_{on}. The calculation of this is the same for all nodes and utilises the logistic-sigmoid function, denoted (σ).

$$t_{on} = \sigma(S_t) \cdot PW \qquad \text{equation 7.4a}$$

This *S* is acted on by a sigmoid function which normalises it. From equation 7.1, the duration t_{off} can be calculated.

$$t_{off} = PW - t_{on} \qquad \text{equation 7.4b}$$

7.2.5 Outputs and Interpretation of Pulses

To understand what the outputs are, one has to think in terms of pulse time. First, consider the inputs to the ABN_w. When a pattern is presented to the network inputs at time *t*, the input node immediately produces a repeating pulse based on the magnitude of this input. Any subsequent node performs signal integration, as described in the previous section, and so functions differently.

$$t_{on} = \sigma(S) \cdot PW \qquad \text{equation 7.8}$$

In equation 7.8, *S* equals the value passed directly to the inputs from the data pattern, normalised to the range of the amplitude of the outputs which are [0,+1] or [-1,+1].

In figure 7.3, the pulse produced at the input nodes for the maximum and minimum pattern values is shown.

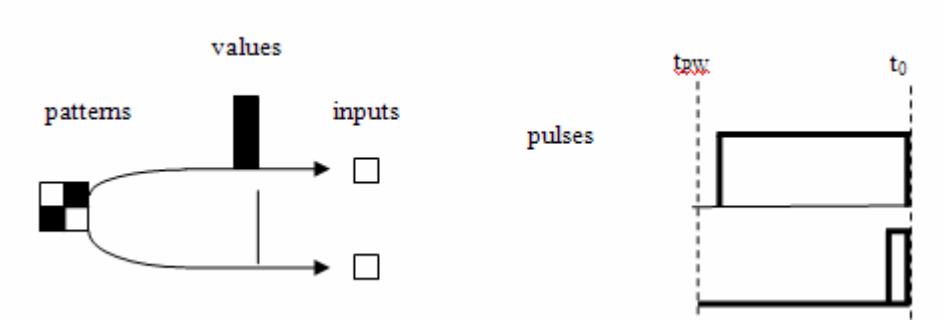


Figure 7.3 - Arrival of pattern values

In this interpretation there is no accumulation of a *S* value, the input node reinitialises each time it completes a pulse-width and reacts to the stimulus level it receives from the data pattern, preventing signal saturation. This is the only way inputs differ from subsequent nodes.

7.2.5.1 Synchronisation of Signals

The artificial mechanics in this stage results in synchronisation of the signals. All the input neurons will fire immediately they are initialised and after each pulse cycle ends. The time values for the (possibly zero) pulses, as shown in figure 7.2, correspond to the following equations.

$$t_{on} = t_{\min} \quad \text{where} \quad t_{\min} \in [0,1] \qquad \text{equation 7.9a}$$

$$t_{off} = t_{\max} \quad \text{where} \quad t_{\max} \in [PW, PW-1] \qquad \text{equation 7.9b}$$

The implemented version is shown in figure 7.3.

7.2.5.2 Observation of the Pulse

If a node is observed to begin a pulse of duration *PW* at t_n, this will finish at t_{n+PW-1}. The observation is broken down as follows; the first measurable increment lasts 1 tick, from $t = 0$ until $t = 1$. This means that the last measurable increment of the pulse starts when the node is assessed at $t = 0 + PW - 1$. Although this lasts until $t = 0 + PW$, when the node is assessed at $t = PW$ the subsequent pulse has started.

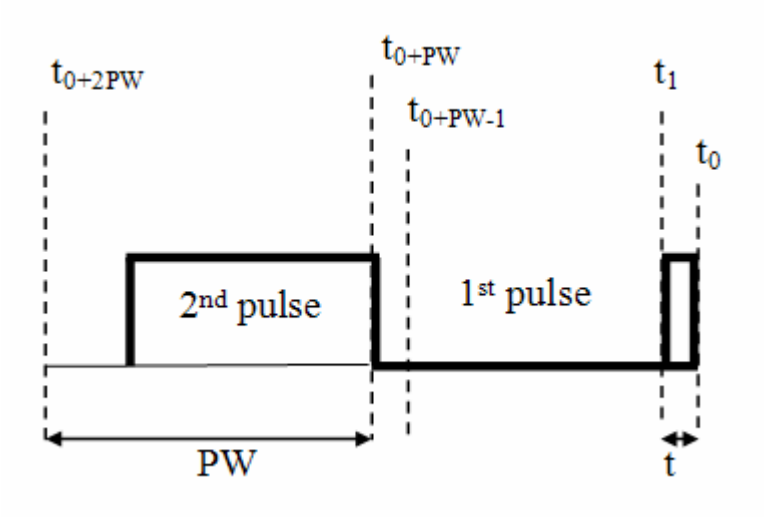


Figure 7.4 – Observation of pulse-width

7.2.5.3 Reflex and Decay Time of Network

In connectionist approaches to static-domain problems, the signal effectively propagates instantly from the inputs to the outputs. In a MLP the signal progresses from one layer to another in turn; however, there is usually no concept of the signal taking a time period to progress through or between nodes. In an ABN_w, there is a time period between the application of a pattern to the input nodes, to when it results in an observable change from the output nodes. This is the previously introduced reflex time of the network. This is of importance in calculating when the ABN_w has relaxed and a stable output has been produced. Likewise once the pattern has been removed from the input nodes, there is a delay until this no longer has an effect on the outputs. The delay time only has an effect between different patterns being presented to the ABN_w and has no other relevance here.

7.3 Pulse-Frequency Modulated ABNs

The pulse-frequency ABN_F topology was arranged in the same structure as the pulse-width ABN_w. As with the previous network this is intrinsically stable. However, in the case of a pulse-frequency network the matter is not as straightforward, as the network may become cyclically-stable with a repeating sequence of output values.

7.3.1 Nature of the Pulse Dynamics

As with the previous case, the pulse consists of an *on time* and an *off time*. Equation 7.10 gives the total signal duration; however, it is no longer constant.

$$t_{on} + t_{off} = PW \qquad \text{equation 7.10}$$

In contrast with the previous case, the length t_{off} encodes the information the node is sending, see figure 7.5. Again the possible values are arranged in pairs, one on each line. The calculations that determine the pulse-time set minimum and maximum values which determine the total pulse cycle time. The relationship between the time periods and the pulse cycle time results in the nodes of the ABN_F being non-synchronised.

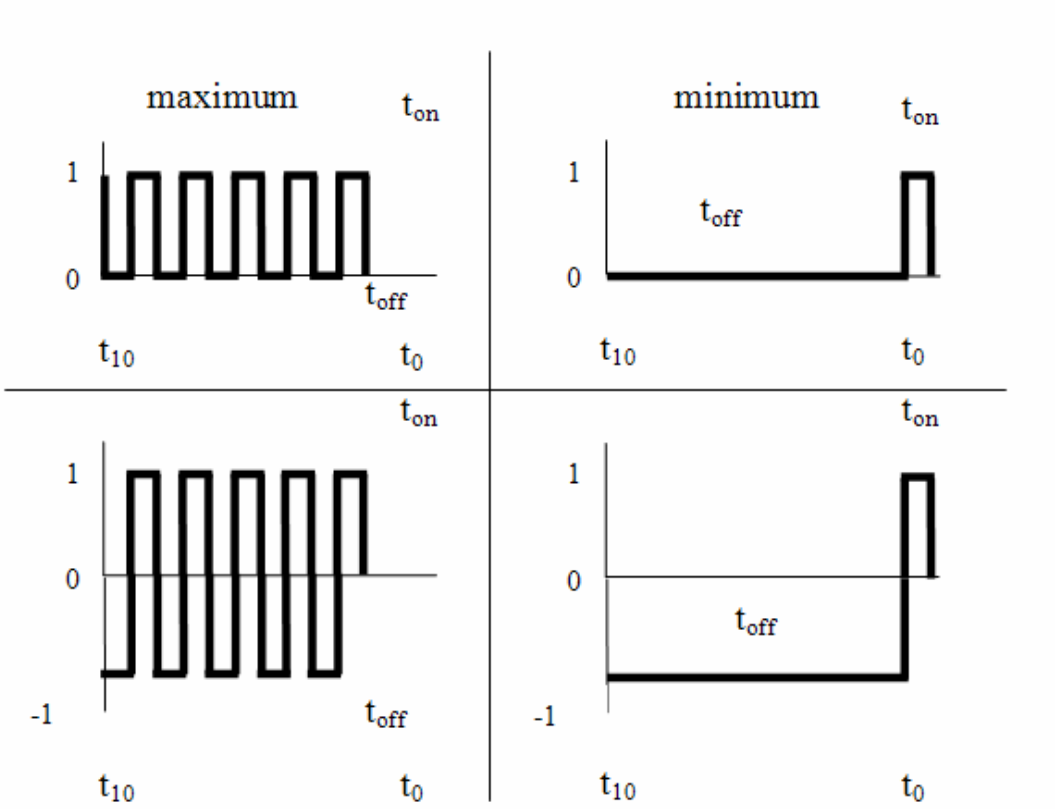


Figure 7.5 – Pulse-frequency max and min profiles

The different nature of the ABN_F dynamics results in various differences compared to the ABN_w. While a pulse-frequency node signal always begins with a measurable t_{on}, a pulse-width node can (in theory) have a minimum signal with no measurable t_{on}; however, the ABN_w implemented here used a minimum t_{on} signal of 1 tick. While a pulse-width node operates with a signal pulse duration, the pulse-frequency node does not. Therefore, it is necessary to note the t_{on} that starts the pulse, then measure the t_{off} that determines its value and is completed by the arrival of the next t_{on}.

As in the previous ABN_w, it takes the pulse a period of time to move from the outputs of one node to the inputs of the next, the propagation delay. Again, the total time it takes for the effects of an input to reach the outputs of the ABN_F is termed the reflex time.

7.3.2 Calculation of Pulse Time

This is the first significant difference from the pulse-width node. As before, the calculation is the same for all nodes. The start of a frequency pulse is signalled by an output pulse of amplitude 1 and duration 1 tick, this is still called t_{on}. The sum values S now produce an output pulse of amplitude 0 and duration t_{off}.

$$t_{on} = 1 \qquad \text{equation 7.11a}$$

The sum S is acted on by a sigmoid function σ which is normalised to the duration of the pulse to allow a constant pulse cycle time. From equation 7.10, the duration PF can be calculated.

$$t_{off} = \frac{1}{\sigma(S_t) \cdot PF_{\max}} \qquad \text{equation 7.11b}$$

With pulse-width nodes, limits were set by having a fixed pulse cycle time; in contrast, with the pulse-frequency nodes, the maximum duration is set by t_{off}.

The input nodes produce an equivalent continuous pulse, a zero pulse is used which has the following values;

$$t_{on} = 1 \qquad \text{equation 7.12a}$$

$$t_{off} = PF_{\max} - 1 \qquad \text{equation 7.12b}$$

This maximum is calculated to a number of ticks. The input neuron which presents this value cannot re-fire until PF is completed.

7.3.3 Observation of the Pulse

In the previous case, an ABN_W, all outputs completed simultaneously. With pulse-frequency nodes they are independent. The outputs are observed until all have relaxed, but these must be assessed at each tick, not over each complete pulse-width cycle.

When a pulse-frequency node begins a pulse of duration 10 ticks (PF_{10}) at $t = n$, then this observation completes at t_{n+PF-1}. In this case, this is $t = 10$. The second pulse begins; in this example it is of duration PF_3 and completes at $t = 13$, (see figure 7.6).

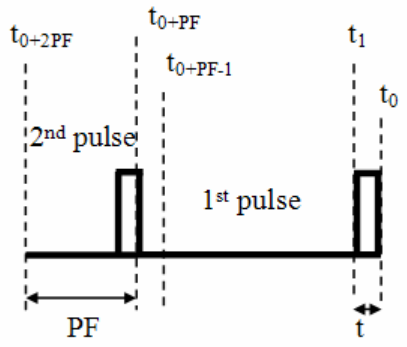


Figure 7.6 – Observation of pulse-frequency

With no constant cycle time, the ABN_F cannot be assumed to be synchronised and a count of the pulses has to be made.

7.3.4 Relaxation

The network does not relax in the same manner as the pulse-width network. The network instead can either produce a constant repeating pulse or a wave-train that cyclically repeats.

The result of this behaviour is that each output node can relax at a different time and each pulse in the cycle can take a different number of ticks. In comparison to other nodes there may be a different number of such pulses in such a cycle and the starting tick of this may be different for every output node.

7.4 Universal-Pulse Modulated ABNs

With the implementation of an ABN_U, there is a universality and robustness trade-off as was previously explained in Chapter 5.

7.41 Production of Locomotive Gaits

A good example of a control signal is that which encodes a locomotive gait in a mobile legged robot. This will be used to demonstrate the waveform producing capabilities of the ABN_U. Both McMinn [2002] and Muthuraman [2005] produced locomotive gaits for legged robots, however they produced only the gait profile; that is the signal controlled the timing of the limbs and the duration of each stride but it did not encode the speed of limb movement.

To produce a walking gait there are three factors to consider. Firstly, each limb must receive a separate signal that is in the correct phase with the other limbs. Secondly, the signal each limb receives must be for the correct duration. Thirdly, each limb must receive a signal which indicates how fast it is to move.

Receiving signals in the correct phase is a feature of the ABN_U. To receive them for the correct length of time, a variable pulse-width is biologically plausible and can be used for stride duration. This leaves the third requirement, encoding speed.

Encoding amplitude within the pulse-width could accommodate speed, but as this is not biologically plausible it was not attempted here (although the further work, Chapter 9, gives more detail). It is more useful to regard the stride as a construction of smaller units rather than a single unit attempting to carry more than one type of information. Therefore, a combination of pulse-width to encode stride duration and pulse-frequency to encode limb speed should suffice for a complete control signal for a locomotive gait; these are shown in figure 7.7. The specific phase will be controlled by the ABN_U.

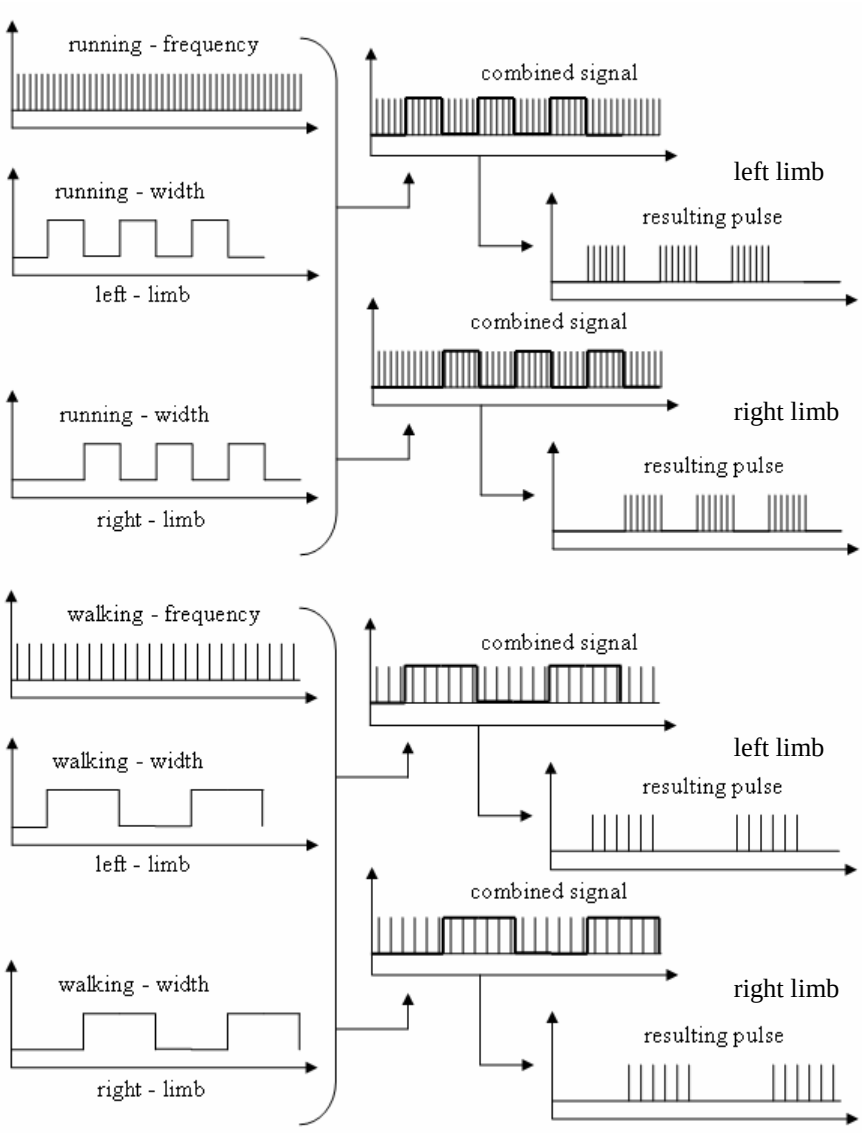


Figure 7.7 – Walking and running biped pulse

The importance of being able to combine pulse types is shown above. The combination of pulse-width and pulse-frequency ABN nodes can produce different pulses to act as gait generators. Any combination can use McCulloch-Pitts style nodes, which an ABN node can simulate with the appropriate leaky integration parameters. The above diagram can be expanded to other gait profiles or limb numbers. In the cases shown, the input data must be produced by an ABN module, while the gaits can be produced by a variety of topologies.

7.42 Pulse Profiles

Each pulse-width and pulse-frequency waveform consists of several phases, which have been shown in the previous diagrams. Let us now consider what is required of a universal pulse.

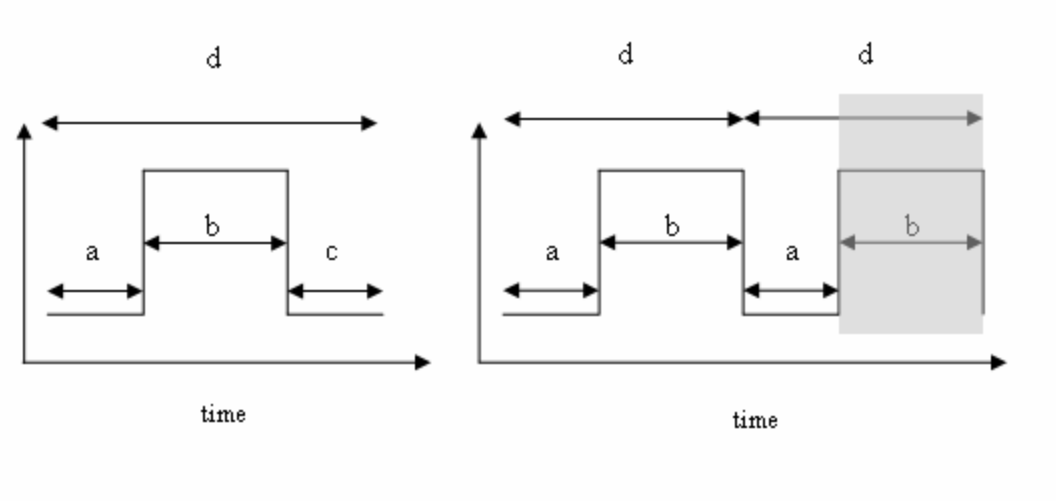


Figure 7.8 – Pulse profiles for universal pulses

The pulse can be broken down into different periods as shown in figure 7.8. These consist of a period “a” before the signal arrives (this, in biological terms, can be viewed as the presence of an inhibitory protein, but is more likely to simply be the absence of a protein). In the diagram, this is followed by the period “b” in which a signal is present. Then follows a period “c” which is the phase following the active phase during which the node does not produce any active signal.

From the two profiles shown it is clear that phase “a” may simple be the phase “c” from the previous pulse.

Given that only the periods {a,b} are considered and the period d = a + b, then what is to be determined is which of these periods are determined by the function of the node. Only one of these can be controlled by the ABN nodes created in this thesis, future work, Chapter 9, considers other cases. The activation of the node is a calculated variable and this through an output function determines the duration of one of the periods.

In the case of “a” the pulse width node previously discussed, period “d” is a constant, period “b” has a duration determined by the stimulation of the node and has output amplitude of 1, and period “a” is calculated as (d - b) and has output of 0.

In the case of “a” the pulse frequency node previously discussed, period “b” has fixed duration of 1 and has output amplitude of 1, period “a” is determined by the stimulation of the node and has output of 0, and period “d” is calculated as (a + b). In the computation of this, a maximum value was set for period “d” and was used in determining period “a”.

There are two variations allowed for by the universal ABN, as follows: Firstly, when period “d” is a constant or has a maximum value, this is now an evolvable attribute of the node. Secondly, the pulses always begin with a leading 1 followed by a 0, and the node may evolve to produce a leading 0 followed by a 1.

There are other variations to pulse profiles which may be considered, however these require more than one degree of freedom to be implemented in the outputs of the node (not currently practical) or produce no practical benefit. The pulse types presented above are sufficient to encode the walking gaits as intended.

Chapter 8
Artificial BioChemical Networks - Experiments and Results

8.1 Introduction to the Chapter

This chapter presents the implementations of the Artificial BioChemical Networks with tasks in pattern recognition and control and compares performance against specific Multi-Layer Perceptrons. Each type of ABN is shown in turn, with its associated experiments. Finally, these two tasks are combined into a modular system for robotic vision and locomotion.

8.1.1 Credibility of Software

As the purpose of this chapter is to show the validity of new connectionism methods, only simple tasks are performed. All patterns and control signals are recognised or produced in response to an input set of four continuous parameters. These are represented using a 2x2 grid. To ensure credibility, these experiments were independently replicated. William Clayton of Olin College was asked to validate the theory based on a requirements description of the ABN. He did this using a much larger pattern set which consisted of a 5x5 grid. Equivalent performance to the systems presented in this chapter was reported and this is included in Appendix A as a published paper.

8.2 Pulse-Width Modulated ABN_w – Trained using a GA

The first experiment set out to produce an ABN that was pulse-width modulated and trained by a Genetic Algorithm (denoted ABN_w-GA) for pattern recognition. The initial setup consisted of four patterns with associated targets.

8.2.1 Program Implementation and Interface

The experiments were implemented in C++. However, C++ does not provide a good graphical capability. Therefore, the programme's progress was displayed through command windows and by examining results in a text editor and a spreadsheet. Each programme activation created separate "event" files. Once the ABN_w was performing correctly, a C++ generated graphical interface was constructed.

This graphical interface consists of a tag structure using Dynamic HyperText Markup Language (DHTML) and follows from rules in an Interface Markup Language (IML). An IML separates data from function and presentation so that these components can be built up into a modular structure. Figure 8.1 shows the interface displaying the first successful event.

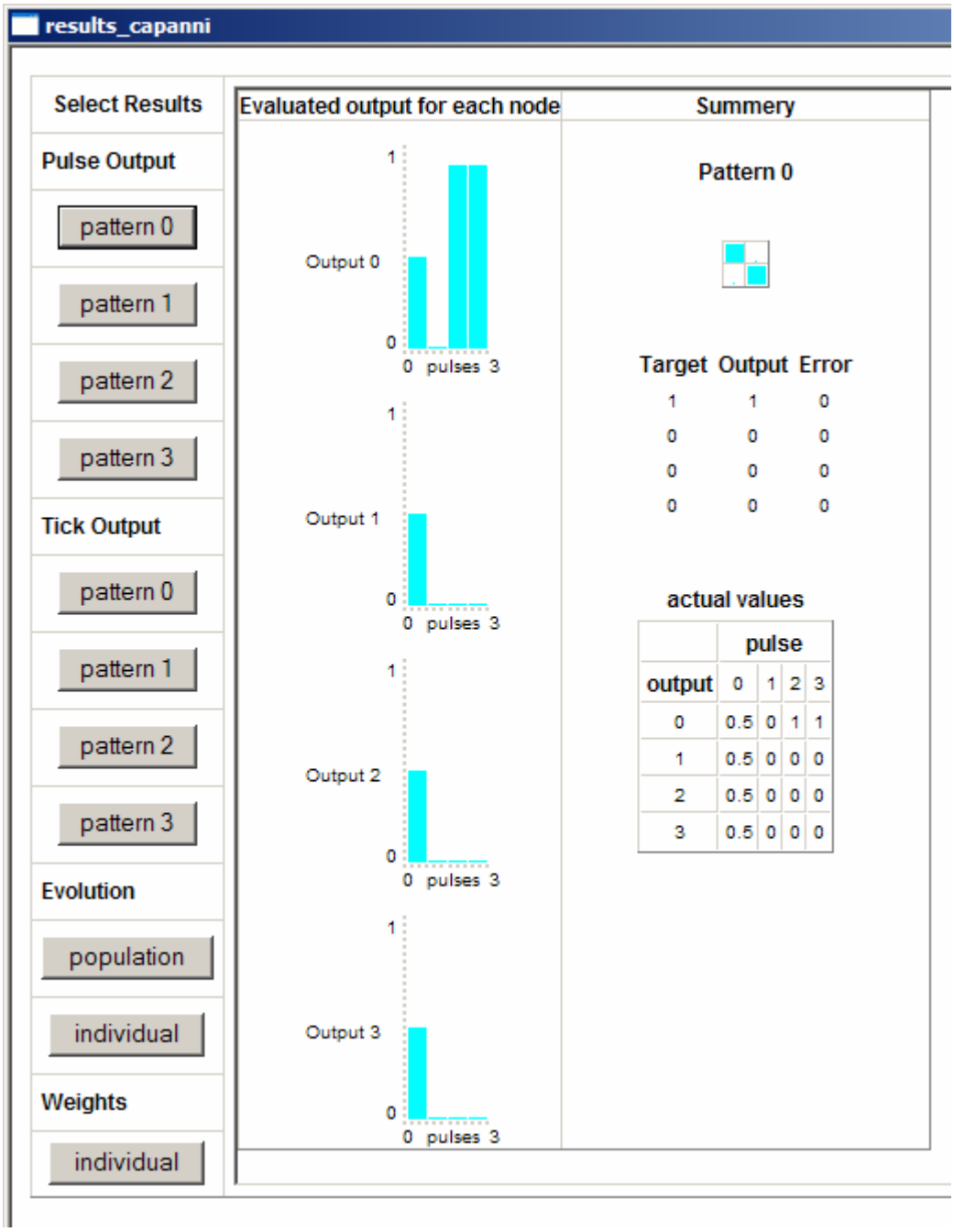


Figure 8.1 – Graphical Interface for ABN_w-GA - using IML

The design is robust and efficient, so as not to distract from the research by attempting to produce a higher quality GUI. Colour variations were implemented to display information which may not be obvious in non-colour figures, therefore relevant incidents are verbally explained.

8.2.2 Successful ABN_w – Trained using a GA - Success

The following figures 8.3 shows the results of the first successful event. Firstly, figure 8.3a shows the resolved ABN_w output pulse for the first pattern, the remaining patterns are included in Appendix G as figures G.1 to G.3. Secondly, figure 8.3b gives the individual ticks that are the basis for the pulses of figure 8.3a; in this event maximum pulse duration was set to 10 ticks, figures G.4 to G.6 show the ticks for the remaining patterns. Thirdly, figures 8.3c(i,ii) show the GA population fitness and the fittest individual. Finally, figure 8.3d shows the evolved signal-pathway weights, rescaling the smaller values for the output layer and comparing them to the original signal-pathway values.

The GA parameters used were as follows:

Parameter	*Value*
String size	One floating point number per signal-pathway
Population size	10
Crossover	Random 50% parent choice
Mutation rate	Uniform random 1%
Mutation	Uniform random ±(original range)
Selection	Roulette

Figure 8.2 – Genetic Algorithm - parameters

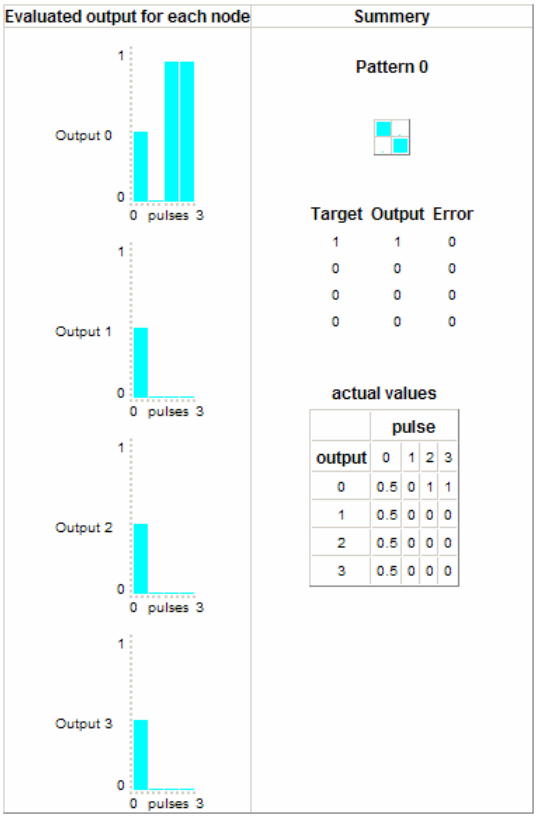


Figure 8.3a – ABN$_w$-GA output pulse – pattern 0

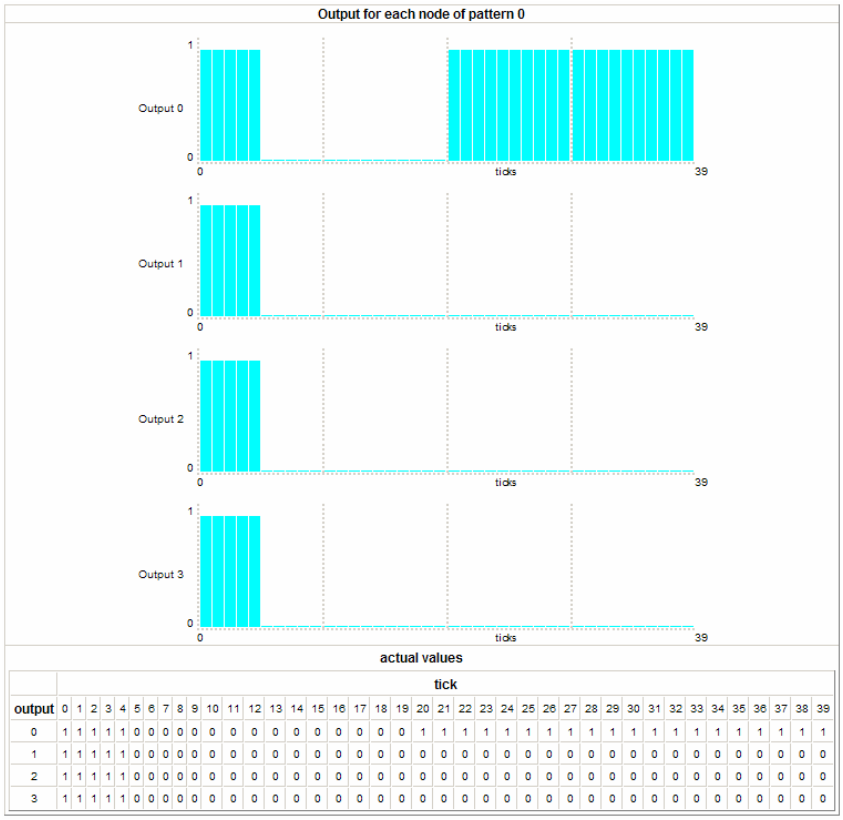


Figure 8.3b – ABN$_w$-GA output ticks – pattern 0

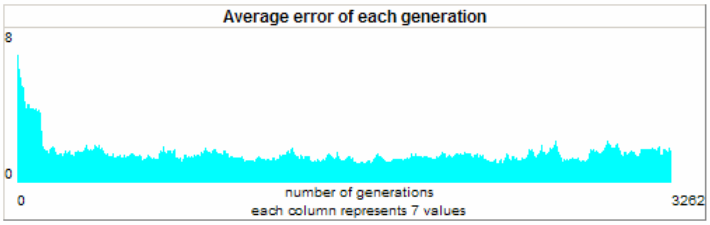


Figure 8.3c(i) – ABN_w-GA population fitness

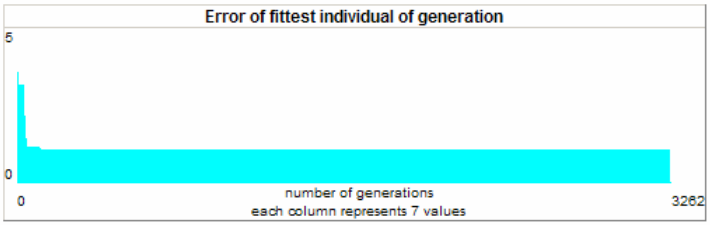


Figure 8.3c(ii) – ABN_w-GA fittest individual

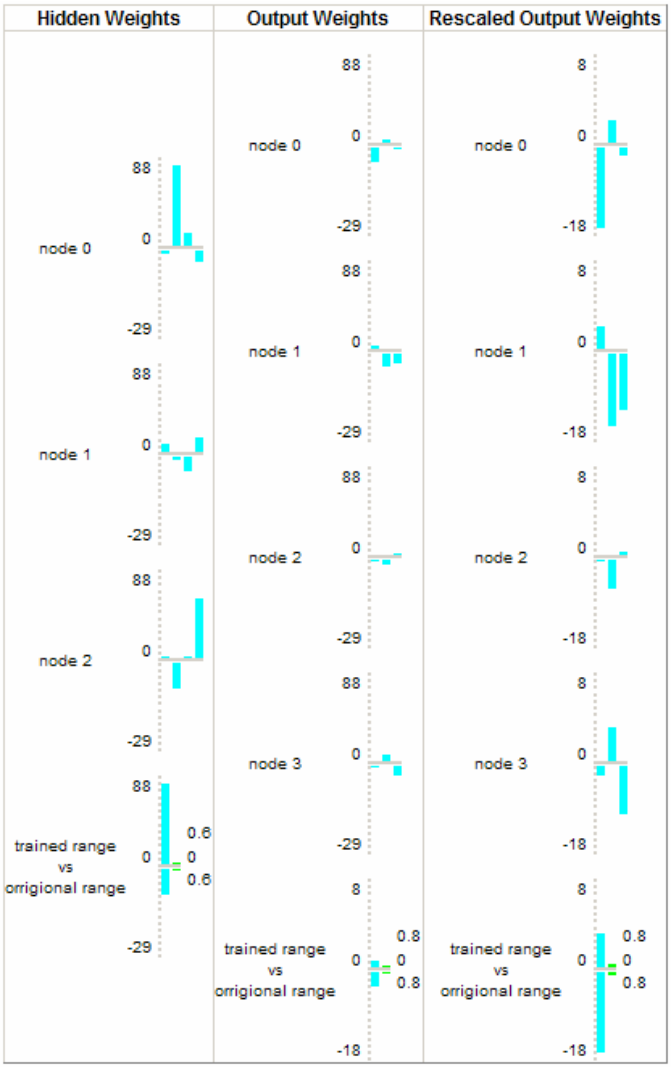


Figure 8.3d – ABN_w-GA signal-pathway strengths

It is shown in figures 8.3a and G.1 to G.3 that the first pulse of all nodes for all patterns is a 0.5 output. This represents the node output immediately on initialisation. The following 0.0 output is due to the resolution of the evolved signal-pathway parameters; hidden node outputs (with an initialisation value of 0.5). This is a chance occurrence, based on the signal-pathways and is different in later events.

Recognition time for pattern 2 is four pulses, the other patterns require three. An additional pulse is generated for all node outputs to evaluate ABN_w relaxation.

Overall the ABN_w error (all nodes, all patterns), designated e_{ABN}, is 0.0 (the target was set at 0.5). The nature of GA training can cause sudden individual improvements as indicated in figures 8.3c. In other iterations of the experiment, the GA produced an e_{ABN} between 0 and 0.5.

Figures 8.3b and G.4 to G.6 show the ticks that construct each pulse. In this event most patterns are rapidly recognised, with the exception of pattern 2 (figure G.5), which takes slightly longer. This may be a result of over-training and is examined later in this chapter.

The GA takes 3262 generations to resolve, however most of these do not affect the fittest individual, as shown in figures 8.3c. Due to the random nature of GA training there is variation in this, and acceptable solutions are found in under 200 generations.

There are major differences in signal-pathway strengths, (see figure 8.3d). Hidden values have a range of 117, outputs have a range of 26. These ranges vary over various events - however hidden strengths always have a greater range than outputs.

Once successful, this experiment was repeated 50 times to validate the results and examine specific functionality regarding pulse-duration and relaxation behaviour.

8.2.3 ABN_w – Trained using a GA - Relaxation Time

Over ten events the maximum number of pulses taken for a single pattern-node to relax was 7, while the average (mean) of all pattern-nodes was 3.48125. This includes the additional conformation pulse (on a basis of two identical pulses to indicate ABN_w relaxation). As expected, the node with a pulse-target of 1 usually has a longer relaxation time than those with a pulse-target of 0.

Most events show a pulse value increasing or decreasing on each subsequent pulse towards the target. Some exceptions are shown in figures 8.4.

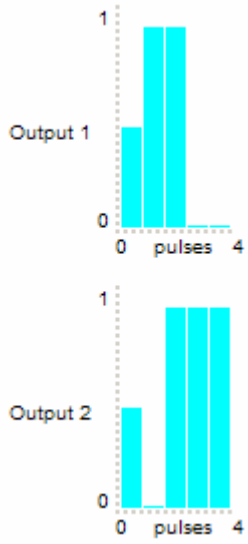


Figure 8.4a – ABN_w-GA output pulse – indirect target acquisition – pattern 2

Figures 8.4a shows outputs 1 and 2 for pattern 2. Output 1 rises to a maximum value of 1 before dropping to its target of 0. This results in a delay in the relaxation of the ABN_w, shown by the triple relaxation-pulse of output 2. Using the two pulse rule for relaxation would have produced an error on output 1 if output 2 had not caused the next pulse to be generated. While this shows a possible problem with only taking two pulses to show relaxation, this technique is designed for a human to extract information out of the ABN_w, not as a component part of its operation. It also indicates a more dynamic nature than is initially apparent, as shown in figure 8.5.

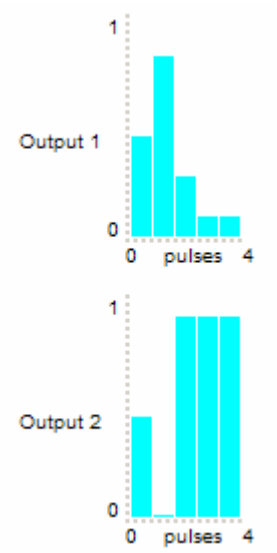


Figure 8.5a – ABN_w-GA output pulse – indirect target acquisition – pattern 2

8.2.4 ABN_w – Trained using a GA - Minimum Error Achieved

The majority of events achieved an error of 0, despite a target error of 0.5. The nature of the ABN_w results in an error increment proportional to maximum pulse-duration. Figure 8.5a displays an error of 0.1 for output 1.

Similar to ANN error, total error (e_{ABN}) is the sum of pattern errors (e_p) which are the sum of node errors (e_n). The node errors are calculated (target – output). The values they can take are increments of (1 / pulse-width), the total number of increments is (C), (see equations 8.1). An LMS error is not required due to quantisation. In the cases when an error remained, it was due to a single node and pattern.

$$e_n = \frac{C}{PW} \qquad C \in R \quad C \le \left| PW_{ticks} \right| \qquad \text{equation 8.1a}$$

$$e_p = \sum e_n \qquad \text{equation 8.1b}$$

$$e_{ABN} = \sum e_p \qquad \text{equation 8.1c}$$

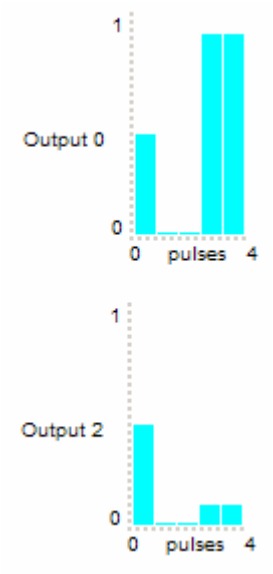


Figure 8.6a – ABN_w-GA output pulse –residual node error – pattern 0

ABN_w behaviour is not straightforward and has been shown not to be simple or unidirectional in changes in output values. In figure 8.6a, it can also be seen that the target of a node can be achieved and then lost if another node forces it to continue pulsing. In this example the relaxation time associated with node 0 causes the node 2 error. It is possible that this is a result of the stop criterion of the GA and if the stop criterion was altered, this type of error may not occur. This was not serious enough to investigate in detail at this stage, but may merit attention in future work.

8.2.5 ABN_w – Trained using a GA - Training Time

The training time (in generations) which produced an acceptable solution ($e_{ABN} < 0.5$) varied greatly. It ranged from 53 to 3263 and averaged (mean) 553. Whether an e_{ABN} of 0 was achieved bore little relation to the training time. The system displayed a tendency to achieve a low population average (mean) error; this fluctuated, with little change to the fittest individual, then a sudden rapid improvement to that individual. This is presumably a result of key values (in specific signal-pathways) evolving, this behaviour is shown in figure 8.7d.

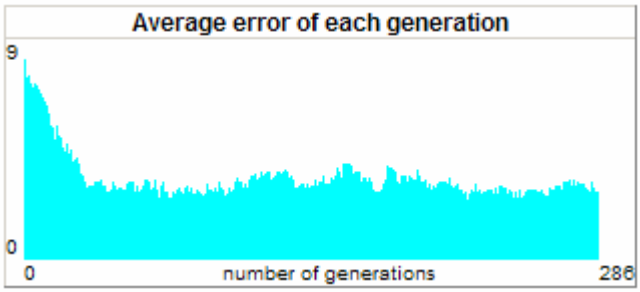


Figure 8.7d – ABN_w-GA population fitness – error vs. generation.

8.2.6 ABN_w – Trained using a GA - Signal-Pathway Strength

The hidden signal-pathways' range always exceeded that of the output signal-pathways. No consistency could be found in the ratio of hidden to output values. This shows that there are a great number of possible solutions to the problem presented. The average (mean) range for the hidden signal-pathways was 123 (-60 to +63) while for the outputs a range of 37.7 (-27.1 to 10.6) was evolved.

8.2.7 ABN_w – Trained using a GA - Noise Tolerance

The performance of the evolved ABN_w-GAs were tested with noise to assess their generalisation ability. The type of noise used was proportional and exact as explained below.

Proportional and exact noise was defined as damaging every data unit by the same proportion of the possible range that unit can take, rather than a random value. For example, a pattern with 5% noise would have a data unit showing a value of 0.95 instead of 1.0 and a data unit of 0.05 instead of 0.0. If this is increased to 10%, the values would become {0.9,0.1}.

This type of noise is used as different data units have different importance in the ABNs' ability to recognise a pattern. If, for example, 10% random noise is applied to patterns this may result in not affecting these data units and gives the network an appearance of noise tolerance. Then if 5% random noise is applied and some critical data units are damaged, the ABN cannot recognise the same pattern and so the comparison of noise tolerance becomes redundant. Therefore noise proportionality and exactness allows a strict

comparison of noise levels by avoiding different loading of critical and non critical data units.

Examine the following diagram. The undamaged T and I are shown at the left and right. Beside each is a damaged version of itself, but is the central pattern a T or an I?

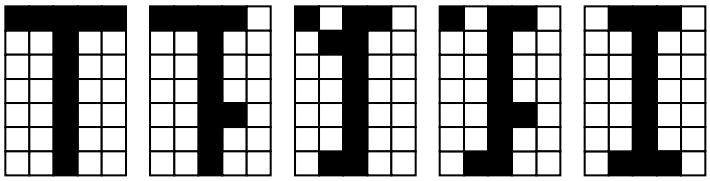


Figure 8.8 - Progressive damage to patterns

8.2.8 ABN_w – Trained using a GA - Results of Noise Tolerance

Like many experiments in this field, there is the potential to generate enormous quantities of information, so the reporting is restricted to the important information from several new events (only those where a specific % of noise had an effect on performance).

The first event (successful iteration of the program) examined noise from 0% to 50% and reported noise at 0% (undamaged) then 45%, 46% and 50% (all data units identical).

The ABN_w-GA showed very good noise tolerance. As the noise was increased (1% at a time) to 45%, all patterns were still recognised and there was no increase in e_{ABN} (which remained at 0.1). In practical terms, the ABN_w-GA recognised all patterns and remained confident in its outputs. At 50% noise all data units are identical, with values of 0.5 (effectively 100% damage). Therefore, a complete immunity to 90% damage is very good performance.

When the noise is increased to 46% there is a critical failure, and e_{ABN} jumps to 6.0 (out of a maximum 16.0). The effect of this is that only one pattern is recognised (this is accidental, as at 50% noise the pattern remains recognised and is the default network output).

Critical failure is extremely poor in engineering terms as there is no warning of the system's performance degrading. A gradual degradation is a strength of connectionist approaches. Figure 8.9a show the comparison between the 45% and 46% noise for pattern 0. Figures G7 to G9 show the comparison for the remaining patterns.

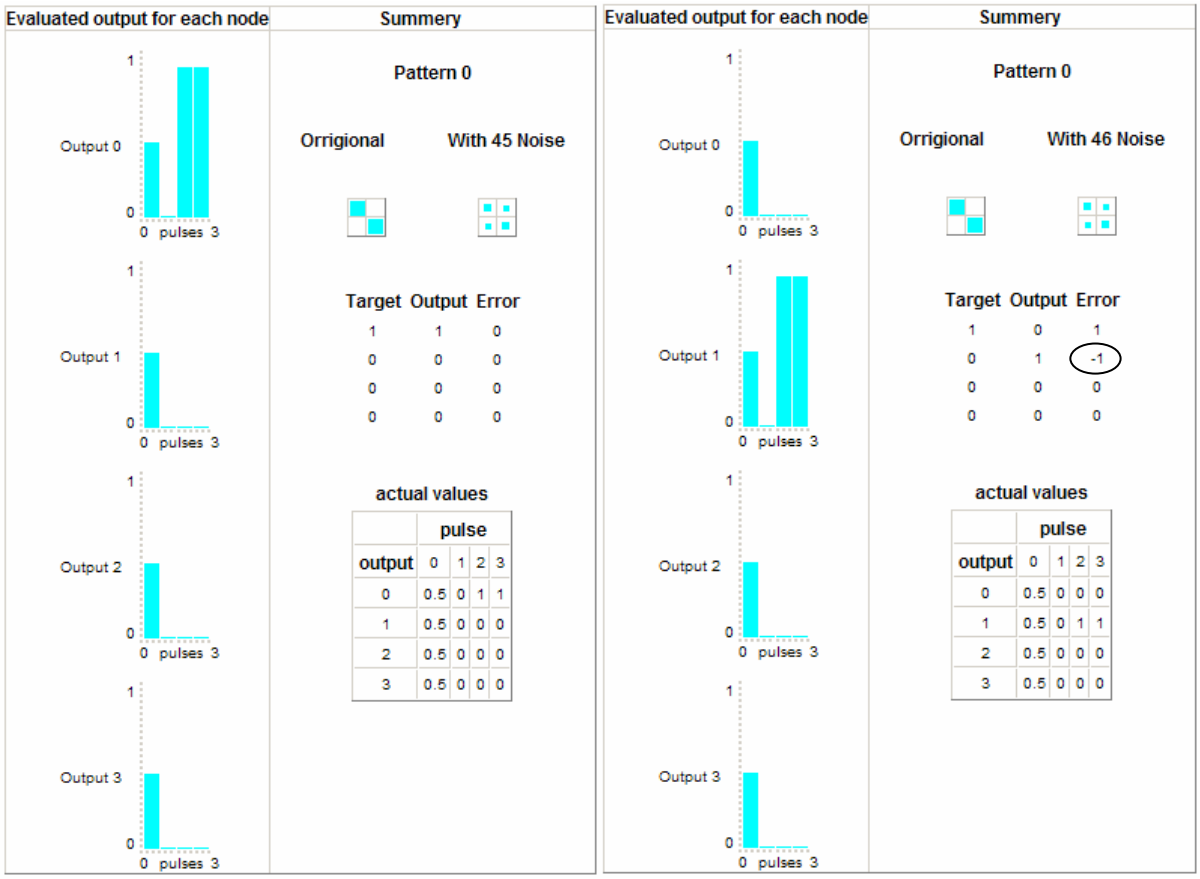


Figure 8.9a – ABN$_w$-GA output pulse - noise 45%,46% – pattern 0

The patterns for 45% noise show how little information the ABN$_w$-GA requires to recognise them. Unfortunately there is no indication that failure is imminent and occurs at 46%.

A second event (complete implementation of the ABN) reported an e_{ABN} of 0.1 for noise of 0%. Noise tolerance remains unchanged until 15%, where the ABN$_w$-GA fails on pattern 0, this shown in Appendix G, figure G.10.

Noise is increased, at 25% pattern 0 is misrecognised as pattern 2, and at 26% noise pattern 3 is miss-recognised as pattern 1; these are shown in figures G.11. Once the noise is

increased to 36% pattern 3 changes to be misrecognised as pattern 2. Once noise reaches 46%, there is complete failure and all patterns are identified as pattern 2. The result of this is that the ABN_w-GA is still undergoing a critical failure; however it affects patterns differently.

A third event is described to give an example of over-fitting. The previous two events took 646 and 1291 generations respectively to achieve an error of 0.1. This event took 2680 generations to achieve an error of 0. Once the noise was raised to 5%, there was a failure to recognise pattern 1, (see figure G.12).

In ANNs, overtraining has a detrimental effect on noise tolerance. It appears that the ABN_w-GA suffers similarly. The work on McCulloch-Pitts and Taylor Series networks in Chapter 5, examined overtraining and showed that amongst the affecting factors is training time.

An examination of ABN_w-GA parameters' tolerance to noise was therefore undertaken, and showed that the output function through the parameter ρ - had no effect; the variable of leaky integration α –also showed no effect; finally the signal quantisation was examined - once more no improvement was found.

The only difference between the ABN_w-GAs which showed different noise tolerance was their signal-pathways. Therefore the problem must be associated with these parameter values. This leads to the next experiments where an ABN_w was trained with a derived Backpropagation Algorithm and compared against a similar ANN. This training was chosen as the GA is a global search algorithm and can find local minima (which may be very different solutions to the same problem), while the BP is a gradient descent algorithm that seeks the globally minimum solution.

8.3 Pulse-width modulated ABN_w – Trained with BP

The pulse-width modulated ABN has been shown to work well in pattern recognition tasks. The main remaining problem is its critical failure in some or all patterns when the noise reaches a high enough level. A specifically derived Backpropagation Algorithm would allow an examination of noise tolerance in a well documented environment and a comparison against an equivalent ANN.

This next section presents the problems overcome in producing such an algorithm and the algorithm used. The abilities of an ABN_w, trained with such an algorithm, are then given.

8.3.1 Problems with ABN_w Backpropagation

The two major problems with ABN_w Backpropagation training concern quantisation and credit assignment. In addition there is a minor performance restriction due to quantisation.

The first problem is related to quantisation that result in a high error, e_{ABN}. When the algorithm was implemented, the ABN_w appeared to train with a reducing e_{ABN} in a manner which resembled gradient descent and indicated an attempt to fit the problem; however, the initial e_{ABN} range of (8,12) reduces to a range of (2,4) and shows no further improvement. From the previous section on ABN_w-GA it is known that this is not the lower limit. Through code and output analysis at each stage the problem was identified.

In Appendix C, the equations for backpropagation are given. These can be applied to ABN_w nodes in an appropriate manner. Equation 8.2 gives the standard delta calculation for an output node "α", designated δ_α. This factor is related to the node error e_n and node target and node output, and allows appropriate changes in the connecting signal-pathways.

$$\delta_\alpha = out_\alpha \cdot (1 - out_\alpha) \cdot (target_\alpha - out_\alpha) \qquad \text{equation 8.2}$$

A similar term is required for the hidden node, in this case node "A" and this is given as equation 8.3. This is the credit assignment portion of backpropagation.

$$\delta_A = out_A \cdot (1 - out_A) \cdot (\delta_\alpha \cdot w_{A\alpha} + \delta_\beta \cdot w_{A\beta}) \qquad \text{equation 8.3}$$

The problem noted as above regards the occurrence of 0 or 1 values. With the use of a sigmoid type function, (equation 8.4a), these are unlikely to occur and will only do so if the magnitude of the sum value gets so big that it has to be rounded to 0 or 1.

$$Output = \frac{1}{1 + e^{-(Sum)}} \qquad \text{equation 8.4a}$$

$$Sum = -\ln\left(\frac{1}{Output} - 1\right) \qquad \text{equation 8.4b}$$

It can be seen from equations 8.2 and 8.3 that if the outputs out_α or out_A ever produce a 0 or 1 value, then the associated delta term δ_α or δ_A will also be 0 which results in a 0 change in signal-pathway strengths. This usually occurs when the target has been achieved; however, there are two situations that can cause a delta of 0 when training has not completed.

Theoretically, a delta of 0 can occur in an ANN if a maximum e_n is produced; this is usually extremely unlikely and should be resolved when different patterns are presented. If it does occur, it may be due to computational rounding. In an ABN_w-BP, maximum e_n may be caused by quantisation. If this occurs, then e_{ABN} will not improve at all, but as it is clear that it can with the GA, this can be ruled out.

A delta of 0 may also occur in the opposite situation, that is a minimum e_n falsely occurring. In this case, the e_{ABN} will reduce prematurely and suddenly stop. This is the problem that actually presented itself in ABN_w-BP training. In an ANN, this will only occur if the sigmoid value is rounded due to computational requirements. In an ABN_w, it may be caused by quantisation.

The solution in an ANN is to use sufficient decimal places. An ABN_w operates using a different method; increasing accuracy can be achieved by increasing the number of ticks that the pulse-width contains. So a pulse-width of 10 ticks can only produce values of one decimal place. Obviously relying on large pulse-durations increases computational complexity in the ABN_w.

This hypothesis was tested by implementing a pulse-width of 100 ticks. The targets were altered to {0.1,0.9}, (replacing {0,1}). The first attempt immediately produced a successful result in 156 epochs. However, subsequent events were not always successful and the ABN_w-BP became stuck at unacceptable errors as before. This improvement showed that the solution may have been found but was still suffering from the quantisation effect. As this problem was solved with an ABN_w-GA, of pulse-width 10 ticks and targets {0,1}, these observations do not indicate that a viable solution has been found. When tested on noise, the ABN_w-BP showed no difference in performance to the ABN_w-GA.

Given the improvement and conflicts explained above, a variety of pulse profiles were assessed. As each was examined the sigmoid output was analysed graphically, (figure 5.7). This showed (due to quantisation) that the outputs {0,1} occur when the sum value exceeds ±2.197 for a pulse-width of 10 ticks and when the sum exceeds ±4.596 for 100 ticks, (equation 8.4b). As the Sum in an ABN is an accumulation value, this is a very small range of operation.

A solution was attempted that took into account the slope parameter ρ, (equation 8.5). From figure 5.17 this can be seen to stretch the range of Sum S, over which an output between (0,1) is produced. The ABN_w-BP with a PW of 10 ticks and targets {0,1} trained successfully (once the second problem was also solved).

$$Output = \frac{1}{1 + e^{-(Sum)/\rho}} \qquad \text{equation 8.5}$$

As the quantisation effect of ticks causes a rounding to $1/PW$ decimal places, this resultes in 0 delta values occurring when the node output comes within $1/PW$ of the target or (1 - target) for targets {0,1}. The use of ρ allows the deltas to take smaller values.

Various values for ρ were examined, both relating to pulse width and taking a constant value. As a mathematical rule, ρ should be less than the pulse-width, to allow the full range of outputs to be produced. For practical purposes, a value of 2.5 to 10 proved successful.

The second problem with ABN_w-BP is related to problems with credit assignment. The credit assignment problem was first identified by Samuel [1951] and described as a fundamental problem by Minsky [1961], to which a specific theoretical solution was presented by Minsky and Papert [1969]. The training algorithm for this specific solution was given by Rumelhart, Hinton and Williams, [1986], (see Appendix C). What must be basically considered is;

> *In a multi-layer network, how much does each hidden unit contribute to the error of each output unit (e_n) that it connects to.*

In the system presented and solved by Rumelhart et al., [1986], the signal and error were in the static domain; in an ABN_w these are both multi-layer and multi-dimensional in the time-domain.

In a MLP-BP, the pattern presented instantaneously produces the outputs and errors. All the information required for each neuron is known, as there is a single (sum, output) pair to assess. In an ABN_w, the pulse from a node evaluated at time t_n is the result of the inputs accumulated over a time-period t_{n-m}. Due to the effect of quantisation and leaky-integration, as m increases the effect of the inputs at t_{n-m} decreases. Therefore, the solution considered the inputs from t_{n-1}. (This is not the complete solution, but is sufficient at this stage. This point is also addressed in further work).

The solution is as follows:

1. The ABN has relaxed at time t_n, when each output node has produced a $pulse_n$, which is the same as the previous $pulse_{n-1}$. These values are used to calculate the relevant output delta values.
2. The error share for the hidden nodes is calculated as the sum of the deltas for the output nodes multiplied by their signal-pathway strengths (as in standard BP).
3. Then the delta values for the hidden nodes are calculated. However, the current output from the hidden nodes, $pulse_n$, has not yet reached the output nodes. It is the previous $pulse_{n-1}$ that must be used (this is the difference for the time-domain).
4. The output signal-pathways can be adjusted using the output deltas and the outputs from the hidden layer at time t_{n-1}.
5. The hidden signal-pathways are adjusted in the same manner as the outputs; however, this depends on what type of unit connects to them. If there is a previous hidden layer, then the $pulse_{n-2}$ would be needed, if they are input nodes (which send a repeating pulse) then any pulse values can be used. This latter option is far easier to implement.

8.3.2 The ABN_w Backpropagation Algorithm

The equations for the ABN_w-BP algorithm are presented; they have the same layer relationship as the equations given in Appendix C, and relate to an ABN_w with equivalent topology. This assumes that ABN_w-BP nodes use a logistic sigmoid function, as implemented in this thesis.

First, the deltas for the output nodes are calculated.

$$\delta_\alpha = out_{\alpha(n)} \cdot \left(1 - out_{\alpha(n)}\right) \cdot \left(t\arg et_\alpha - out_{\alpha(n)}\right) \quad \text{equation 8.6a}$$

$$\delta_\beta = out_{\beta(n)} \cdot \left(1 - out_{\beta(n)}\right) \cdot \left(t\arg et_\alpha - out_{\beta(n)}\right) \quad \text{equation 8.6b}$$

These are used to calculate the new signal-pathway strengths s^+.

$$s^+_{A\alpha} = s_{A\alpha} + \eta \cdot \delta_\alpha \cdot out_{A(n-1)} \quad \text{equation 8.7a}$$

$$s^+_{B\alpha} = s_{B\alpha} + \eta \cdot \delta_\alpha \cdot out_{\mathbf{B}(n-1)} \quad \text{equation 8.7b}$$

$$s^+_{C\alpha} = s_{C\alpha} + \eta \cdot \delta_\alpha \cdot out_{C(n-1)} \quad \text{equation 8.7c}$$

$$s^+_{A\beta} = s_{A\beta} + \eta \cdot \delta_\beta \cdot out_{A(n-1)} \quad \text{equation 8.7d}$$

$$s^+_{B\beta} = s_{B\beta} + \eta \cdot \delta_\beta \cdot out_{\mathbf{B}(n-1)} \quad \text{equation 8.7e}$$

$$s^+_{C\beta} = s_{C\beta} + \eta \cdot \delta_\beta \cdot out_{C(n-1)} \quad \text{equation 8.7f}$$

The deltas for the hidden layer are calculated.

$$\delta_A = out_{A(n-1)} \cdot \left(1 - out_{A(n-1)}\right) \cdot \left(\delta_\alpha \cdot w_{A\alpha} + \delta_\beta \cdot w_{A\beta}\right) \quad \text{equation 8.8a}$$

$$\delta_B = out_{B(n-1)} \cdot \left(1 - out_{B(n-1)}\right) \cdot \left(\delta_\alpha \cdot w_{B\alpha} + \delta_\beta \cdot w_{B\beta}\right) \quad \text{equation 8.8b}$$

$$\delta_C = out_{C(n-1)} \cdot \left(1 - out_{C(n-1)}\right) \cdot \left(\delta_\alpha \cdot w_{C\alpha} + \delta_\beta \cdot w_{C\beta}\right) \quad \text{equation 8.8c}$$

These are used to calculate the strength of the hidden signal-pathways; however, as the outputs from the input nodes remain constant while the data pattern is presented, there is no need to calculate a pulse at t_{n-2}.

$$s^{+}_{\Omega A} = s_{\Omega A} + \eta \cdot \delta_{A} \cdot out_{\Omega} \qquad \text{equation 8.9a}$$

$$s^{+}_{B\alpha} = s_{B\alpha} + \eta \cdot \delta_{\alpha} \cdot out_{\mathbf{B}} \qquad \text{equation 8.9b}$$

$$s^{+}_{C\alpha} = s_{C\alpha} + \eta \cdot \delta_{\alpha} \cdot out_{C} \qquad \text{equation 8.9c}$$

$$s^{+}_{\lambda A} = s_{\lambda A} + \eta \cdot \delta_{A} \cdot out_{\lambda} \qquad \text{equation 8.9d}$$

$$s^{+}_{\lambda B} = s_{\lambda B} + \eta \cdot \delta_{B} \cdot out_{\lambda} \qquad \text{equation 8.9e}$$

$$s^{+}_{\lambda C} = s_{\lambda C} + \eta \cdot \delta_{C} \cdot out_{\lambda} \qquad \text{equation 8.9f}$$

This completes the training for one data pattern presented to the ABN_{W}.

8.3.3 Performance of ABN_w - Trained with BP

The ABN_w-BP was presented with the same problem and parameters as the ABN_w-GA. The interface was enhanced as shown in figure 8.12.

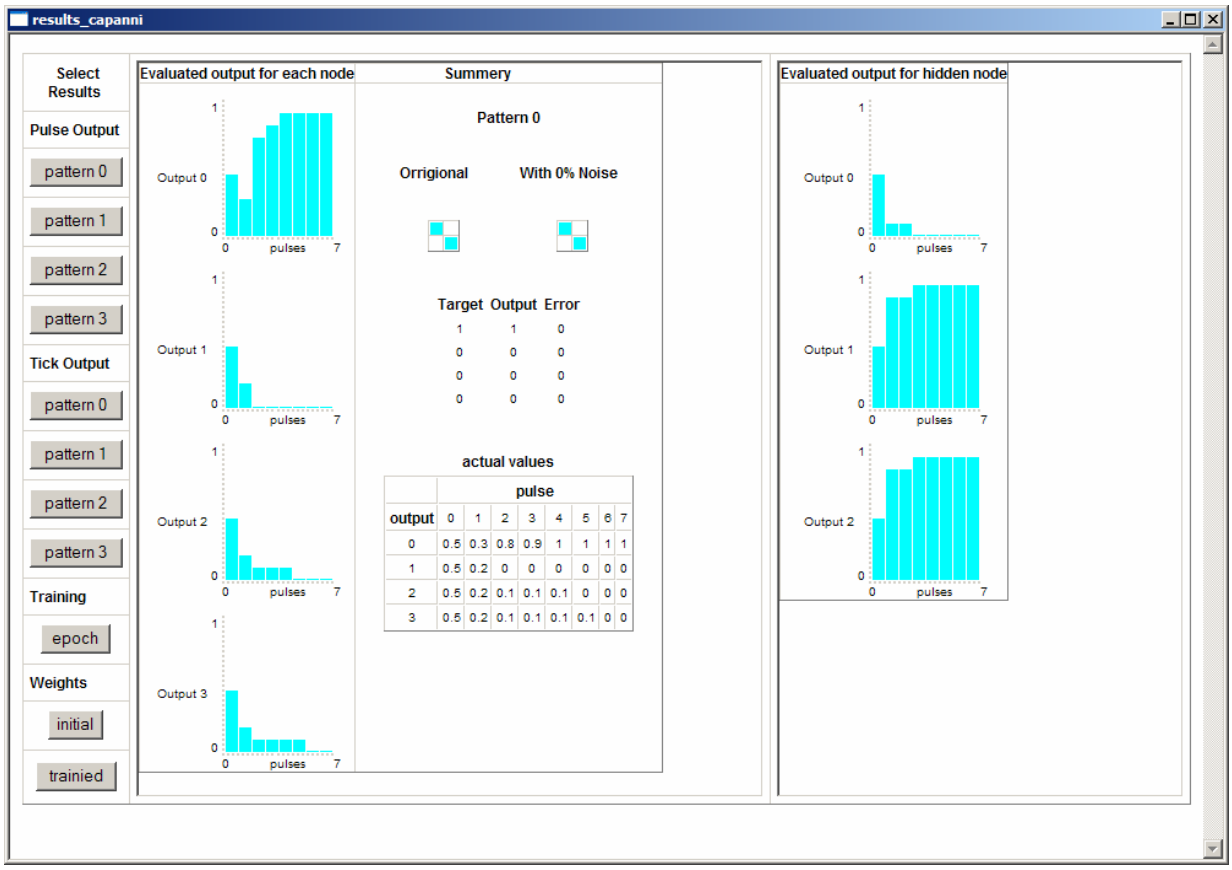


Figure 8.10 – Interface for ABN_w-BP

The signal-pathway strengths (pre and post training) are recorded, in the ABN_w-GA there was a family of individuals and so pre-training values were omitted. The hidden node outputs are included; they were examined for the formulation of the ABN_w-BP algorithm. The number of epochs to reach the target error replaced the number of generations. The interface shows a successful training event output in figures 8.11 and G.13 to G.21, some screens are cropped due to size.

8.3.4 Comparison of ABN_w – BP and ABN_w – GA

This section refers to specific training events where examples are shown - many events were assessed to ensure the ones reported accurately reflect system performance.

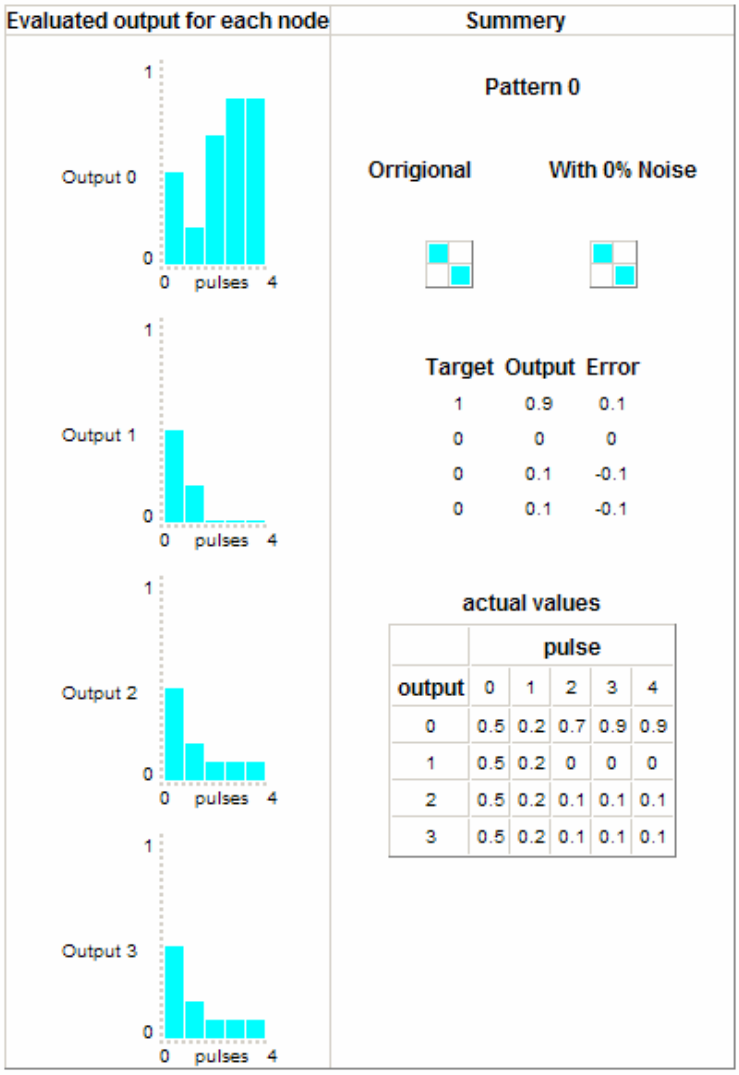
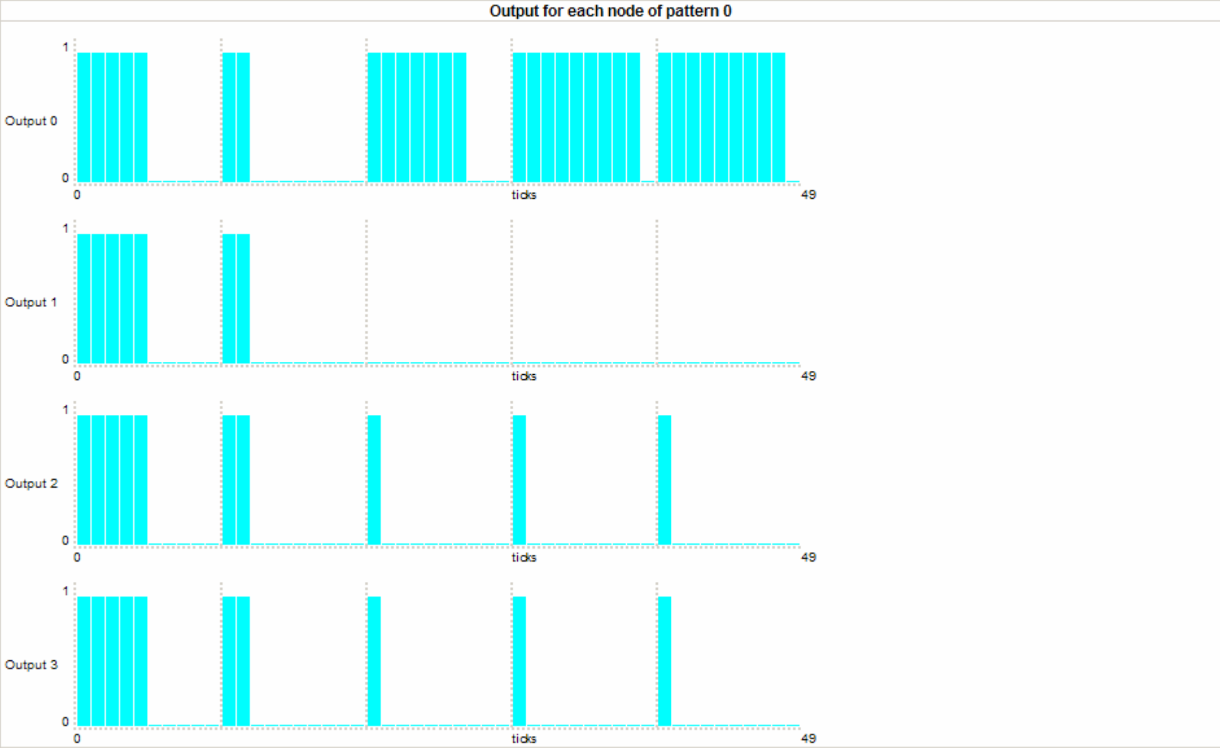


Figure 8.11a – ABN_w-BP output pulse – pattern 0



actual values

output	tick 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1
1	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0
3	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0

output	tick 25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49
0	1	1	0	0	0	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

Figure 8.11b – ABN_w-BP output ticks – pattern 0

An examination of the ABN_w-BP output pulses, (figures 8.11a and G.13 to G.15), and their associated output ticks, (figures 8.11b and G16 to G18), show two immediately obvious differences to the ABN_w-GA.

Firstly, the ABN_w-BP takes more pulses to relax. This is a feature that can be reduced; however, it may be evidence that the ABN_w-BP is more robust. This theme is developed later in noise tolerance analysis.

Secondly, the e_{ABN} is contributed to by multiple e_n values. Previously, all e_{ABN} were loaded onto a single node and pattern. These differences are related, in that spreading the error around the connections appears to give a higher noise tolerance, as was seen in Chapter 5.

The nature of BP is to implement small changes to signal-pathways, individually and progressively moving the e_n for each node-pattern towards the target. This increases the likelihood of achieving the target error without leaping past it, as a GA is prone to do. The residual e_{ABN} is therefore spread around the ABN_w-BP, not concentrated in one particular area. This error distribution is a feature of good generalisation.

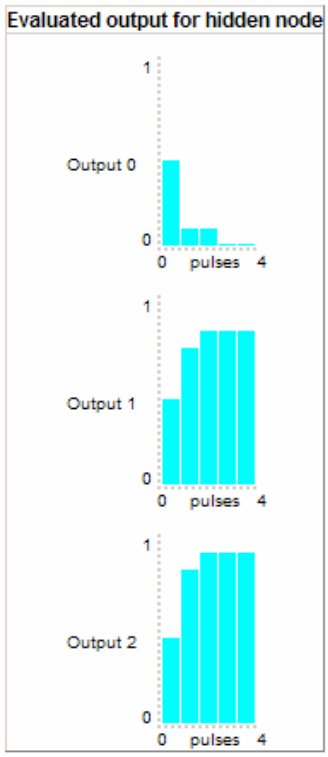


Figure 8.11c – ABN_w-BP hidden nodes – pattern 0

In the hidden node outputs, (figures 8.11c and G.19 to G.21), the last pulse output does not reach the output nodes before the ABN_w relaxes. All the hidden nodes have relaxed to a range of [0,1]. This indicates that the hidden nodes are performing a recognition function, not just echoing the inputs, and that backpropagation is making use of these values.

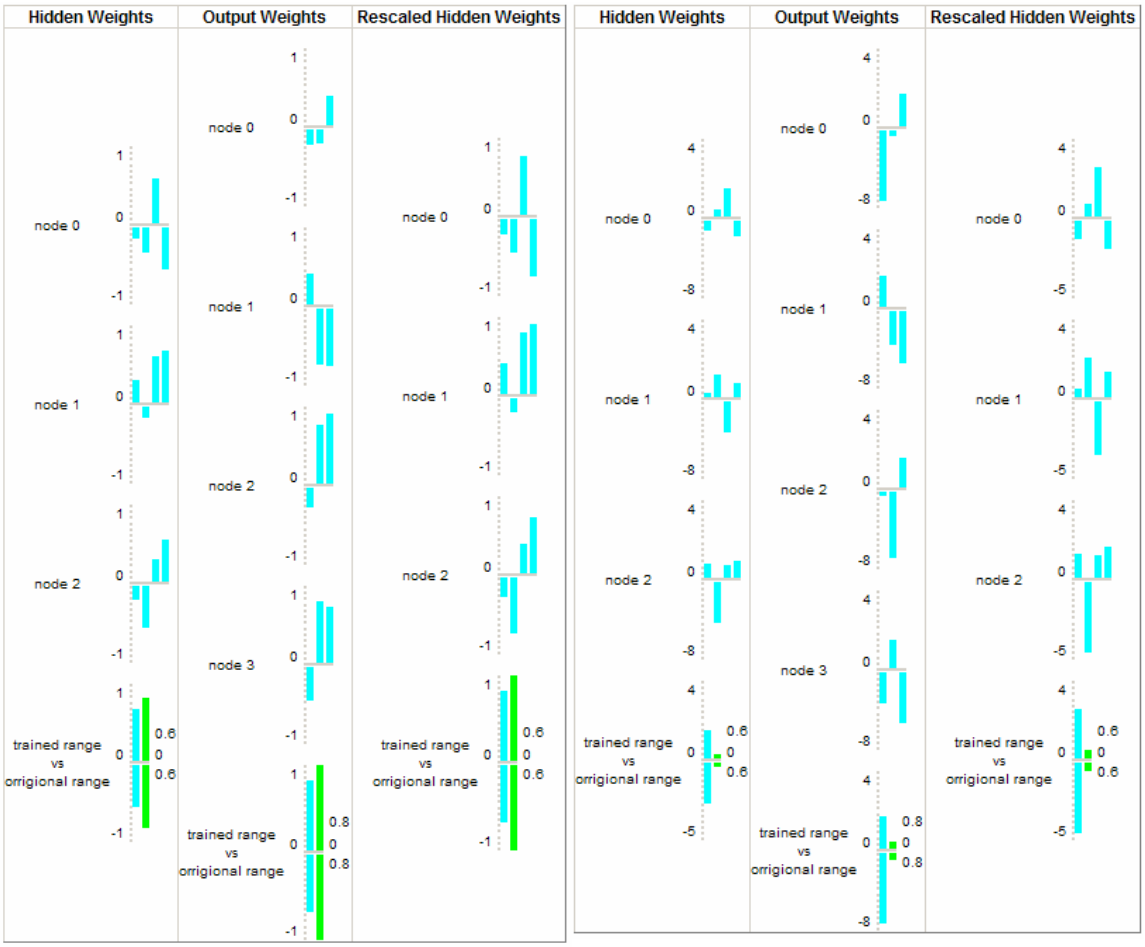


Figure 8.11d(i,ii) – ABN_w-BP pre & post training signal-pathway strengths

The signal-pathway strengths, (figures 8.11c), account for the functional difference of the ABN_w-BP when compared to the ABN_w-GA. Initial values and then the trained values are shown. In contrast to the GA, the hidden values are smaller than the output values - this was consistent over several events. Importantly, the trained values are far smaller than those found by the GA.

Another observation is that if several ABN_w-BP are trained to the same error, then they always finds the same (or very similar) signal-pathway strengths; however, these are not necessarily assigned to the same nodes. This implies that the ABN_w-BP is finding the global-minimum solution to the problem, while the ABN-GA is finding local-minima (some of which may be magnifications of the global-minimum).

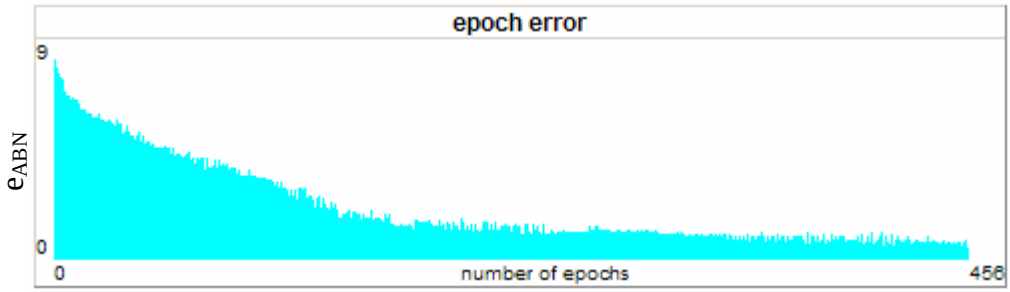


Figure 8.11e – ABN_w-BP – error vs. epoch

The improved error minimising of the ABN_w-BP is shown in figure 8.11e. This emulates the classical and gradual reduction in e_{ABN} of gradient descent. The spikes that occur during the descent may be due to the approximations that are implemented, or may be an exaggeration of known anomalies in BP (errors are capable of rising in adjacent epochs).

8.3.5 ABN_w - Trained with BP - Noise Tolerance

This section reports on the noise tolerance of the ABN_w-BP. Some comparisons with the ABN_w-GA are made, however the main comparison is with the MLP-BP in the next section. Examples are used from different events, which were tested to confirm that they are typical performances.

The ABN_w-BP was trained on two target errors {0.5,0.05}. The ABN_w-BP took 519 epochs to achieve an e_{ABN} of 0.4 with target 0.5, and 759 epochs to achieve an e_{ABN} of 0.0, with target 0.05. The minimum e_{ABN}, due to quantisation, is 0.1 so the ABN_w-BP is forced to over-fit.

The effect of noise on the ABN_w-BP trained to target e_{ABN} {0.5} appears at 5%, far lower than in most of the ABN_w-GA events; however, the error effect is dispersed amongst the nodes and patterns, (figures 8.12 and G.22 to G.24). The rise in error from 0% to 5% noise causes graceful degradation, (observed at the output nodes), as desired, while the hidden nodes are similarly gradually affected.

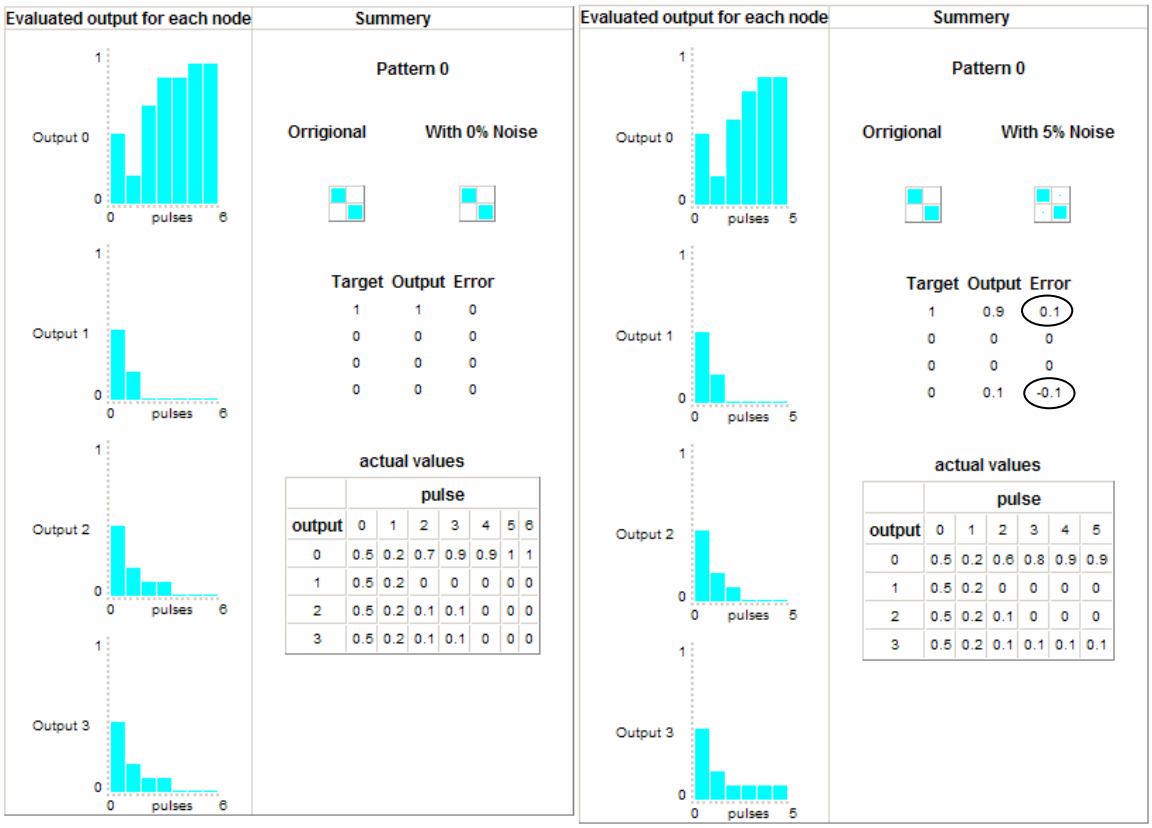


Figure 8.12 – ABN_w-BP output pulse - target e_{ABN} 0.5 - noise 0%,5% - pattern 0

An examination of noise across several events produced the results shown at figure 8.13, where at specific noise there is an effect in e_{ABN}. The individual pattern errors e_p are shown.

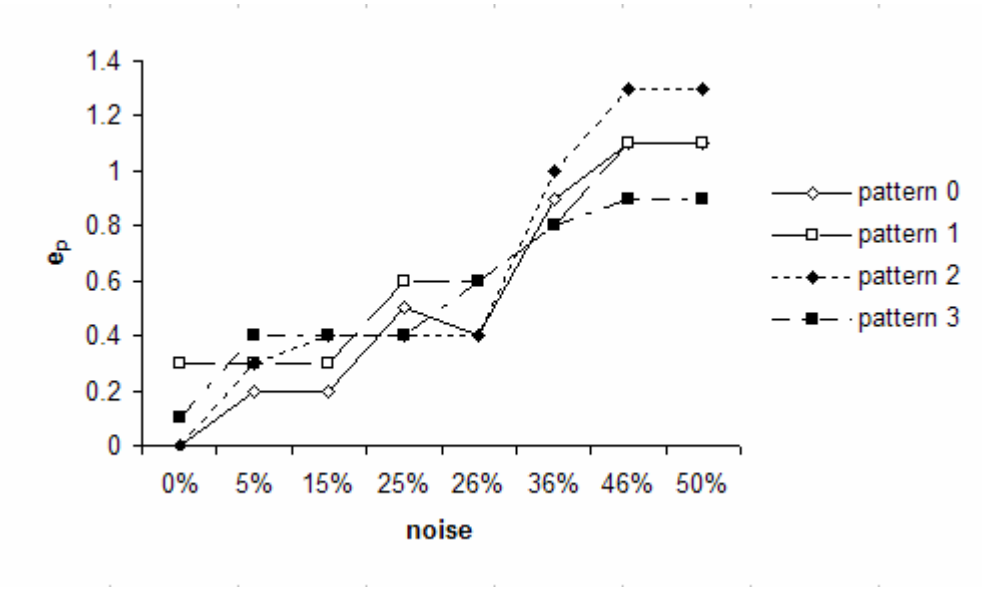


Figure 8.13 - ABN_w-BP - target e_{ABN} 0.5 - error vs. noise

The trained ABN_w-BP has an e_{ABN} of 0.4 at a noise of 0% (undamaged) and the error increases progressively with rising noise. At 5% noise, three outputs e_p {0,2,3} degrade slightly but there is no more damage until 15% noise where e_p {2} is affected, and then at 25% noise where e_p {0,1} are affected. At 26% noise two patterns are affected; e_p {0} improves, e_p {3} degrades. All e_p then remain unchanged until 36% noise where all are affected, and then until 46% where maximum e_{ABN} occurs.

Despite the rising e_{ABN}, the ABN_w-BP can correctly identify its inputs, performing signal separation, at 45% noise when the input domain is constrained to 10% of its original range.

The effect of noise on the ABN_w-BP trained to target e_{ABN} {0.05} is shown in figures 8.14 and G.25 to G.27.

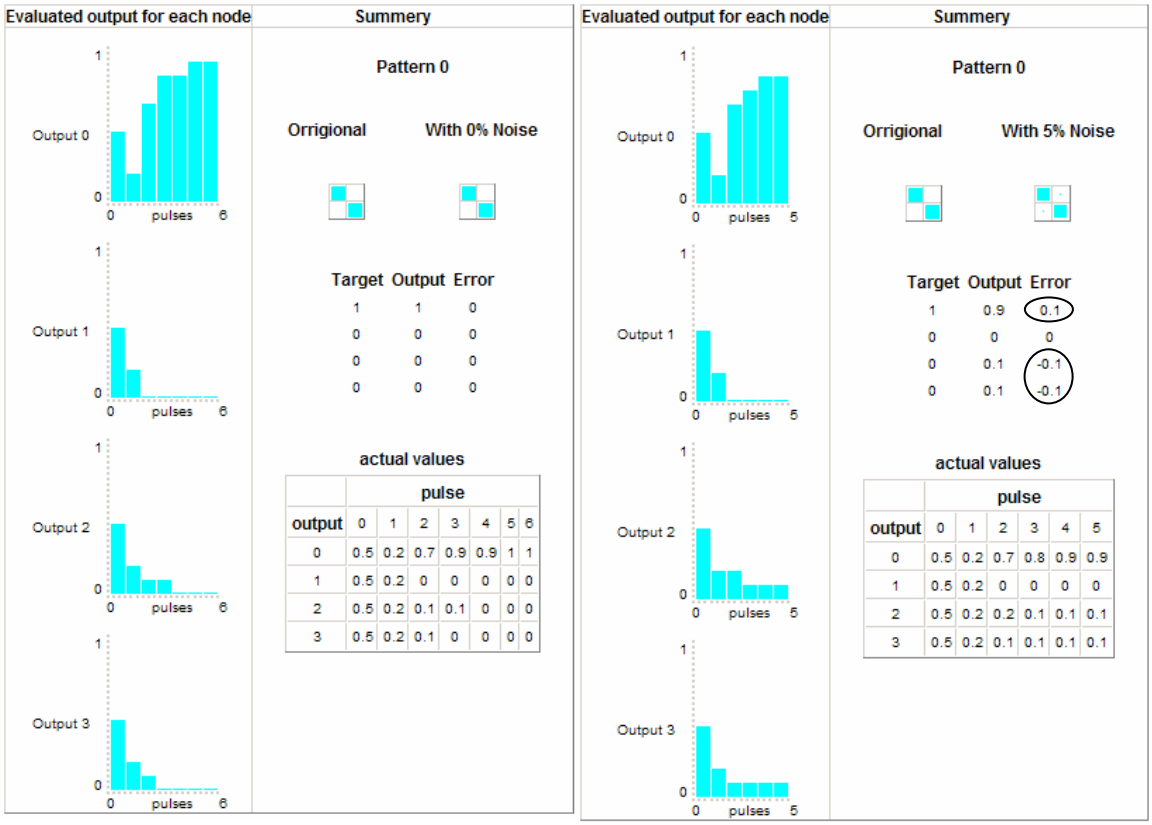


Figure 8.14 – ABN_w-BP output pulse - target e_{ABN} 0.05 - noise 0%,5% - pattern 0

The same examination of the effect of noise for this ABN_w-BP as the previous one is shown in figure 8.15.

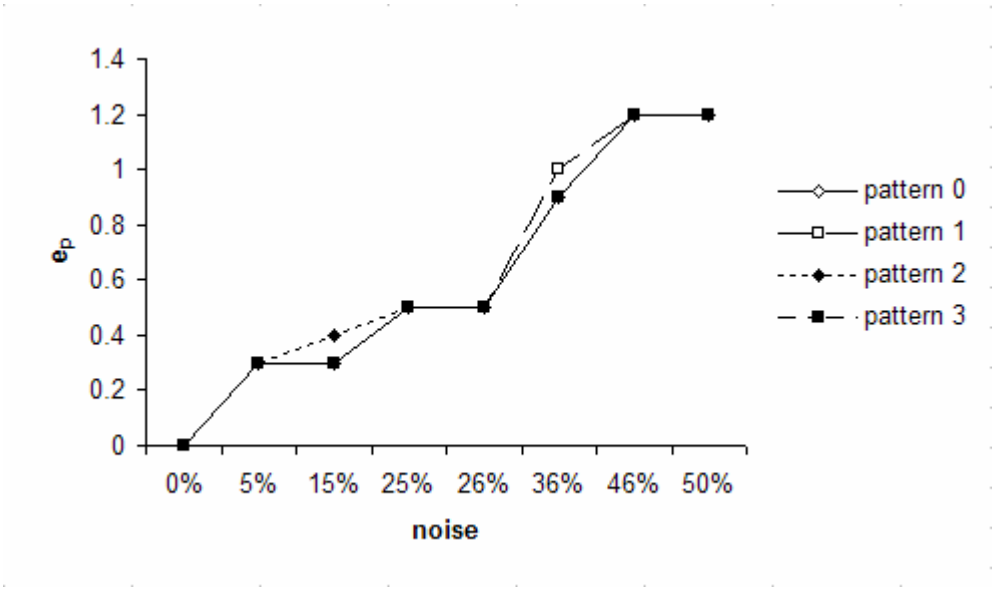


Figure 8.15 - ABN_w-BP - target e_{ABN} 0.05 - error vs. noise

The trained ABN_w-BP has an e_{ABN} of 0.0 at a noise of 0% (undamaged). The effect of noise appears at 5%, as it did in the previous example. The overall performance between the ABN_w-BP trained to e_{ABN} {0.05} and e_{ABN} {0.5} is so similar that a comparison of e_{ABN} vs. noise, shown in figure 8.16, shows almost no difference.

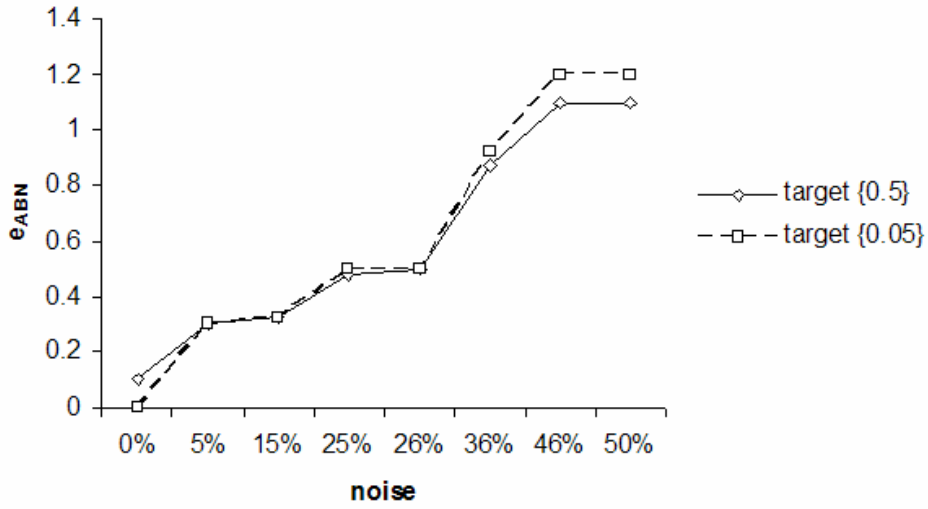


Figure 8.16 - ABN_w-BP - targets e_{ABN} 0.5, 0.05 - error vs. noise

The initial advantage of the ABN_w-BP trained to e_{ABN} {0.05} over e_{ABN} {0.5} disappears at 5% noise and performance remains slightly worse as noise increases. This indicates that

training to the lower error results in an over-fitting by the ABN_w-BP. The network remains highly noise tolerant with complete recognition at 45% noise.

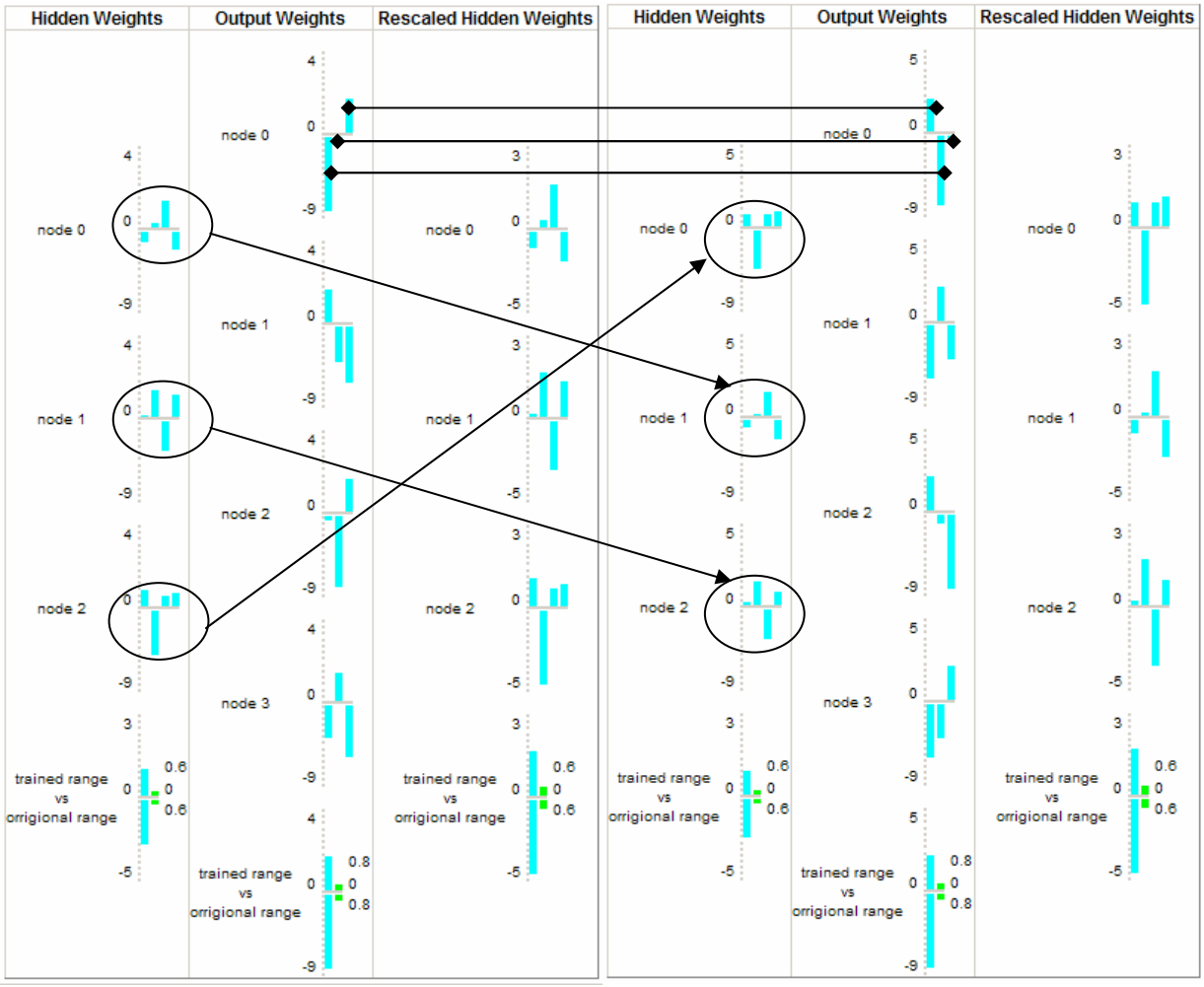


Figure 8.17 - ABN_w-BP - targets e_{ABN} 0.5, 0.05 - trained signal-pathways

If the signal-pathway strengths are compared, (figure 8.17), for the ABN_w-BPs trained to different targets, their values show similarities. The hidden signal-pathways are almost identical, despite belonging to different nodes. They are matched between the ABN_w-BPs as nodes {0,1}, nodes {1,2} and nodes {2,0}. While the output signal-pathways appear unrelated, if examined as properties of the hidden node from which they connect, then they too are almost identical. This is shown for the first node, the others match in the same way. The original random signal-pathway has no obvious correlation. Therefore, the similar end point came from different starting points.

These similarities in the pathway strengths are the result of the nature of the problem. This is a simple problem domain and results in only one global-minimum solution local to the initial random signal-pathway strengths. Networks trained to the same error are likely to end up with the same weights, even if the starting values are different. The ABN_w-GA produces vastly different pathway strengths because the GA allows for a global search in the problem domain.

In the comparison between the ABN_w-BP and the ABN_w-GA, training time is a factor; however once these networks are trained, the only difference is in values of their signal-pathway strengths. The result of the larger signal-pathways of the ABN_w-GA is that the ABN_w-BP degrades gracefully with noise, while the ABN_w-GA suffers critical failures. The ABN_w-GA's larger signal-pathways amplify small changes caused by noise and have a greater influence on network performance.

The ABN_w-BP's ability to achieve low target e_{ABN} while retaining noise tolerance and graceful decay is compared with a standard MLP-BP in the next section, as these later two effects are usually incompatible.

8.4 Multi-Layer Perceptron – Trained with BP

A MLP was constructed and trained with the Backpropagation Algorithm as shown in Appendix C. The MLP is of equivalent topology to the ABNs; 4 input nodes, 3 hidden neurons and 4 output neurons. The input range is normalised to [0,1], and the neurons use a Sum activation and a logistic sigmoid output function.

The same patterns and target errors were presented to the MLP-BP and its noise tolerance was tested with the same values as used before.

8.4.1 MLP – Trained with BP - Memory Capacity

The MLP-BP and the ABN_w-BP achieve low errors (e_{MLP} or e_{ABN}) with 3 hidden units (nodes or neurons), producing a complete problem solution. Both recognise patterns with 2 hidden units; however, their confidence is low, these networks are unable to achieve a

low error. On various events, a network error of approximately 2.0 was achieved. When trained with a single hidden unit, neither MLP-BP nor ABN_w-BP came close to a solution, with an error of approximately 6.0. In both topologies, the ABN_w-BP can achieve a slightly lower error by adapting quantisation effects.

8.4.2 MLP – Trained with BP - Training Time

The MLP-BP requires many more training epochs to reach the same errors as the ABN_w-BP and takes progressively longer as the target is lowered. When a slope, ρ, is included in the MLP-BP, training time increases further. This is to be expected, however it benefits MLP-BP noise tolerance; this is addressed later.

network	*error target*	*approximate epochs to reach*
ABN tick (10)	0.5	500
ABN tick (10)	0.05	750
MLP ρ (1)	0.5	1200
MLP ρ (10)	0.5	10000
MLP ρ (1)	0.05	90000
MLP ρ (10)	0.05	900000

Figure 8.18 - Epoch comparison

The advantage of the ABN_w-BP increases as the target error is lowered. While the ABN_w-BP training time rose from 500 to 750 epochs, the MLP-BP ρ (1) rose from 1200 to 90,000 epochs. When a ρ of 10 is set for the MLP-BP, training time increases proportionally, as expected.

8.4.3 MLP – Trained with BP - Noise Tolerance

The MLP-BP and the ABN_w-BP show high noise tolerance for this problem. The differences are discussed on a case by case basis.

The ABN_w-BP and the MLP-BP were trained with a target error of 0.5. The ABN_w-BP performance was shown in figure 8.13. For the MLP-BP $1 \leq \rho \leq 10$, equivalent examples are shown in figures 8.19.

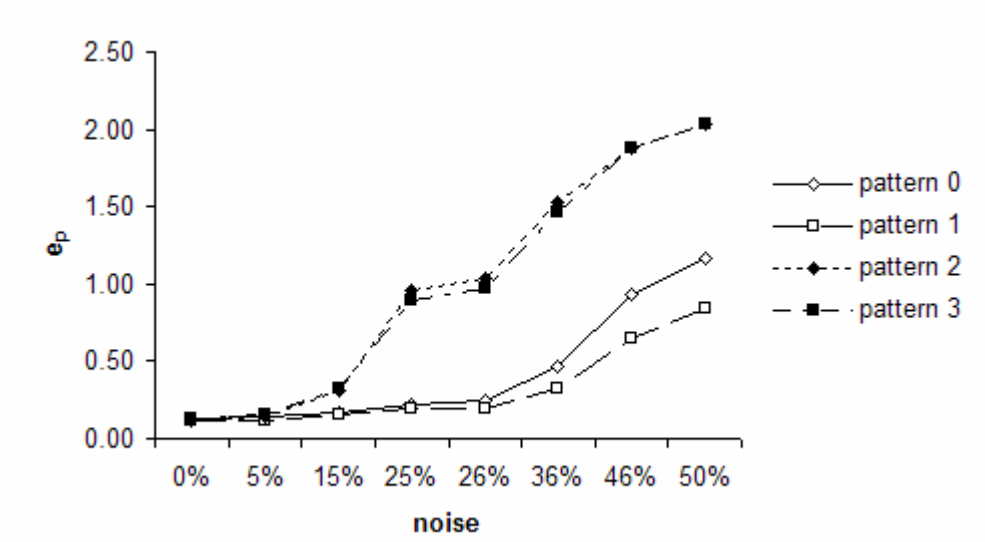


Figure 8.19a – MLP-BP ρ (1) - target error 0.5 - error vs. noise

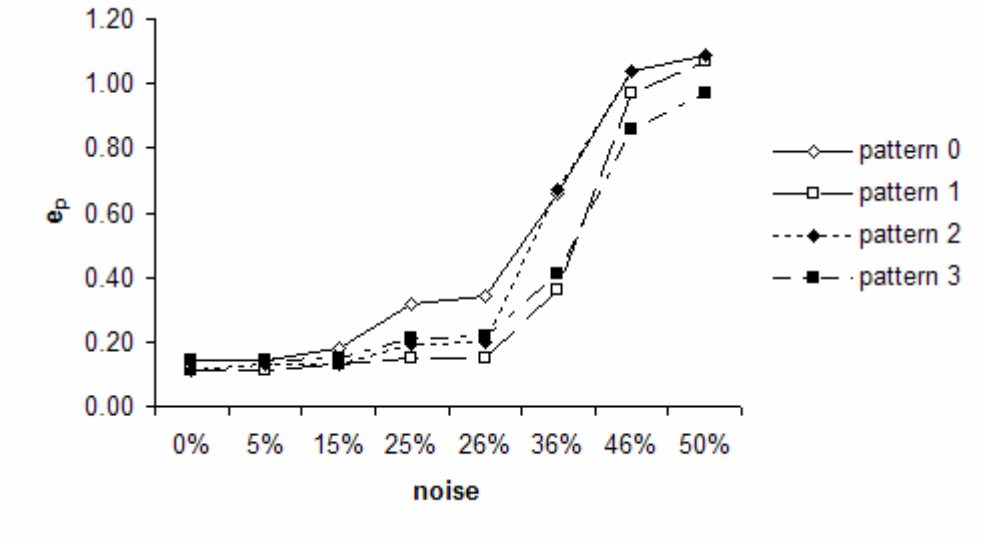


Figure 8.19b – MLP-BP ρ (10) - target error 0.5 - error vs. noise

Comparing figures 8.13 and 8.19 shows that while at low noise the ABN_w-BP produces on average a higher error than both MLP-BP networks, as the noise rises the ABN_w-BP performance improves compared to the MLP-BP ρ (1). However, it remains poorer than the MLP-BP ρ (10). Average errors are compared in figure 8.20.

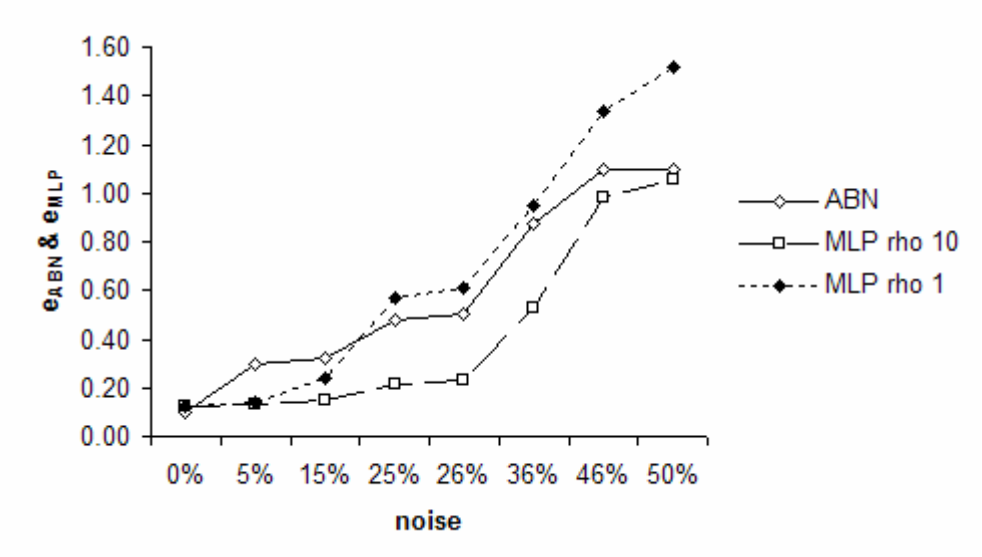


Figure 8.20 - ABN_w-BP and MLP-BP - target error 0.5 - error vs. noise

It appears that the ABN_w-BP has a performance which is between the two MLP-BPs. This is actually slightly more subtle, at 25% noise, the MLP-BP ρ (1) fails to recognise all patterns, the MLP-BP ρ (10) recognises all patterns at up to 44% noise, the ABN_w-BP recognise all up to 45% noise.

The ABN_w-BP performance when a target error of 0.05 was set is shown in figure 8.15. Equivalent examples are shown for the MLP-BP $1 \leq \rho \leq 10$, in figures 8.21.

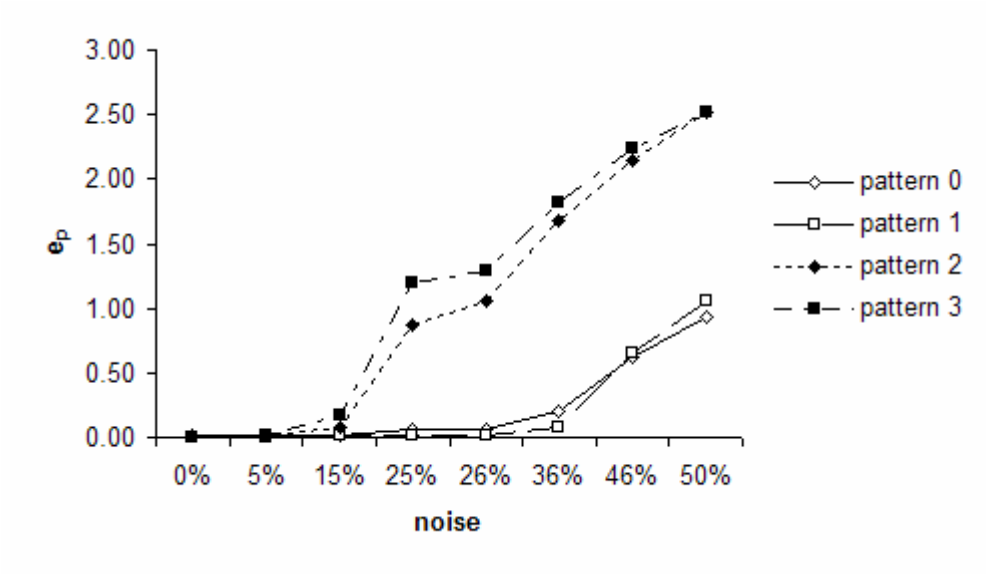


Figure 8.21a – MLP-BP ρ (1) - target error 0.05 - error vs. noise

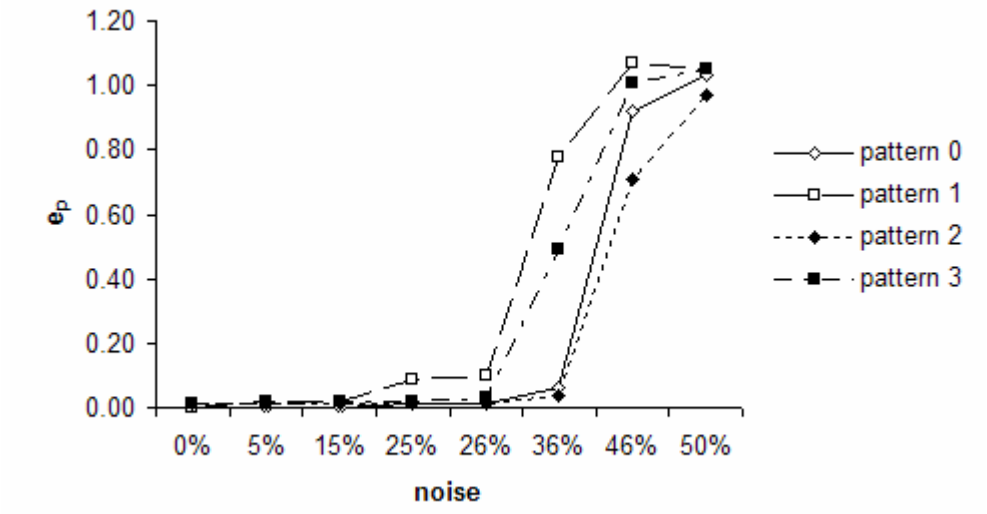


Figure 8.21b – MLP-BP ρ (10) - target error 0.05 - error vs. noise

Comparing figures 8.15 and 8.21 shows that at low noise the ABN_w-BP, similar to before, produces an average error higher than both the MLP-BPs. As the noise rises, the ABN_w-BP and MLP-BPs behaviour is similar to that shown in the previous example. The error changes in response to noise are more sudden for the MLP-BPs. Average errors are compared in figure 8.22.

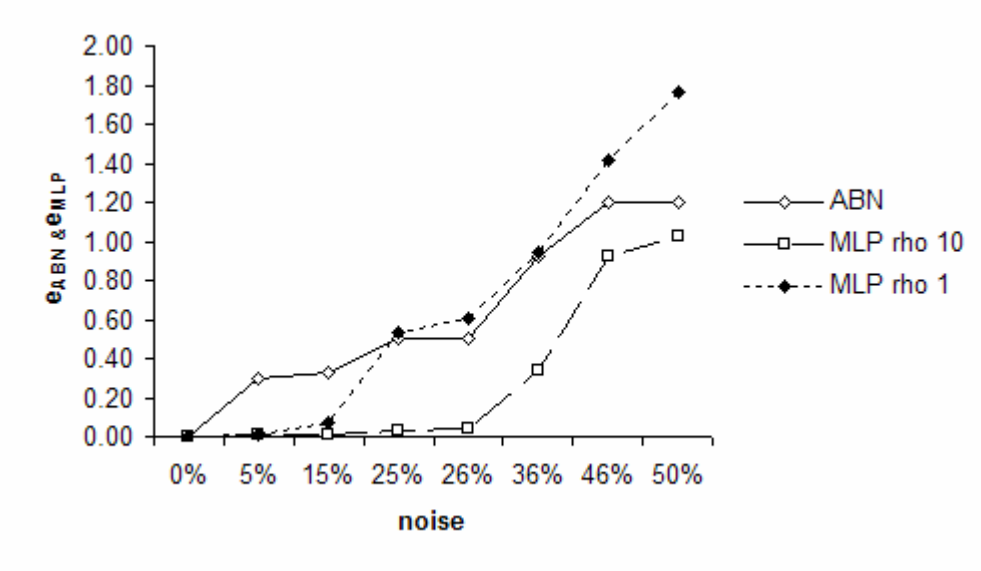


Figure 8.22 - ABN_w-BP and MLP-BP - target error 0.05 - error vs. noise

The ABN_w-BP performance still lies somewhere between the two MLP-BPs. Again, the actual performance on patterns recognised is more subtle. At a noise of only 23% the MLP-BP ρ (1) fails to recognise all patterns, the MLP-BP ρ (10) recognises all up to 36% noise, as does the ABN_w-BP.

8.4.4 MLP – Trained with BP - Summary

The ABN_w-BP and MLP-BPs suffer a generalisation depredation as target errors are lowered indicating classic over-fitting. In training time and recognition tests the ABN_w-BP performs better than either MLP-BP and it is better at avoiding critical failures.

The ABN_w-BP demonstrates excellent noise tolerance in tasks that MLP-BPs specialise in; however, the MLP-BP can only be presented with an idealised signal (a snapshot of the input data), while the ABN_w-BP can receive a continuous signal that is required for interacting with real world systems, for example artificial vision.

The comparison is also biased in favour of the MLP-BP, as it is allowed to operate unrestrictedly while the ABN_w-BP is restricted in decimal places by the pulse-width.

8.5 Pulse-Frequency Modulated ABN_F – Trained using a GA

A major change in the ABN was implemented for pulse-frequency modulation, (denoted ABN_F). The topology was the same as that of the ABN_w, and the network was presented with the same pattern. The network was trained by a Genetic Algorithm, ABN_F-GA.

8.5.1 ABN_F – Trained using a GA - Apparent Success

During the incremental changes in producing the ABN_F performance was inconsistent with apparently successful outputs changing to give errors. This was due to the more complicated behaviour of the ABN_F rendering the relaxation criteria used for the ABN_w unreliable.

The reason for inconsistent success was that the ABN_F is capable of producing the same type of relaxation output as the ABN_w. However, in other events when the network appeared to relax, the pulses were part of a cyclic-stability output.

An ABN_F therefore has two relaxed states. When a constant repeating pulse was produced, this was taken to be the node value. When there was cyclic-stability then the average value of all pulses in the cycle was taken as the node's output; in some cases this was not detected and a relaxed value was assumed from the cycle.

8.5.2 Successful ABN_F – GA Implementation

The following figures 8.23 and G.28 to G.33 show the first successful event with both relaxed and cyclic behaviour. Firstly, figures 8.23a and G.28 to G.30 give the resolved ABN_F output pulses. Secondly, figures 8.23b and G.31 to G.33 show the individual ticks.

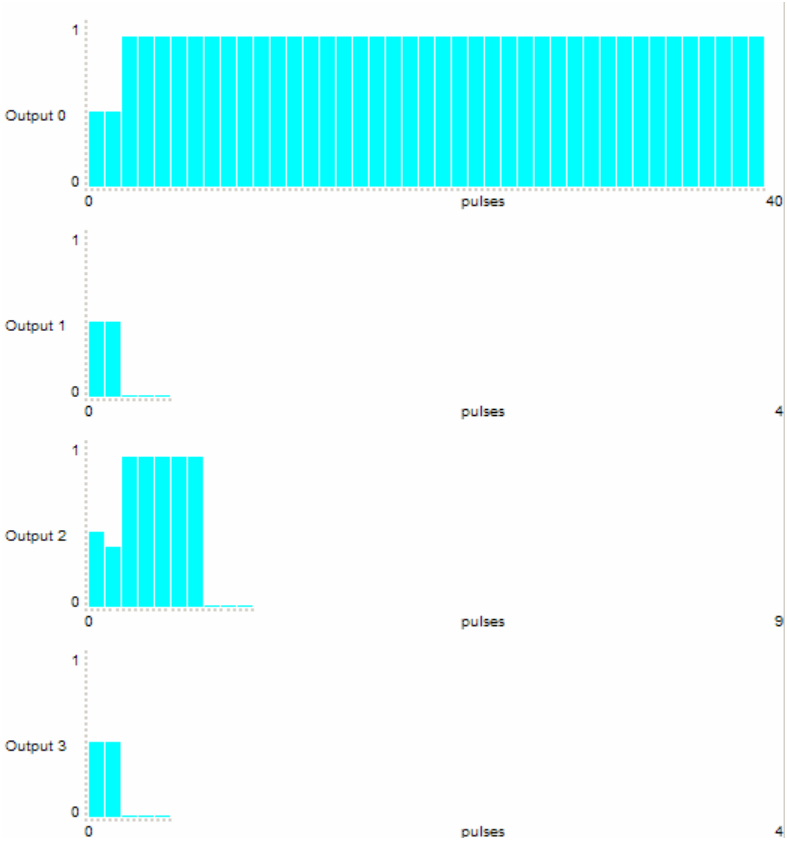


Figure 8.23a – ABN$_F$-GA output pulse – pattern 0

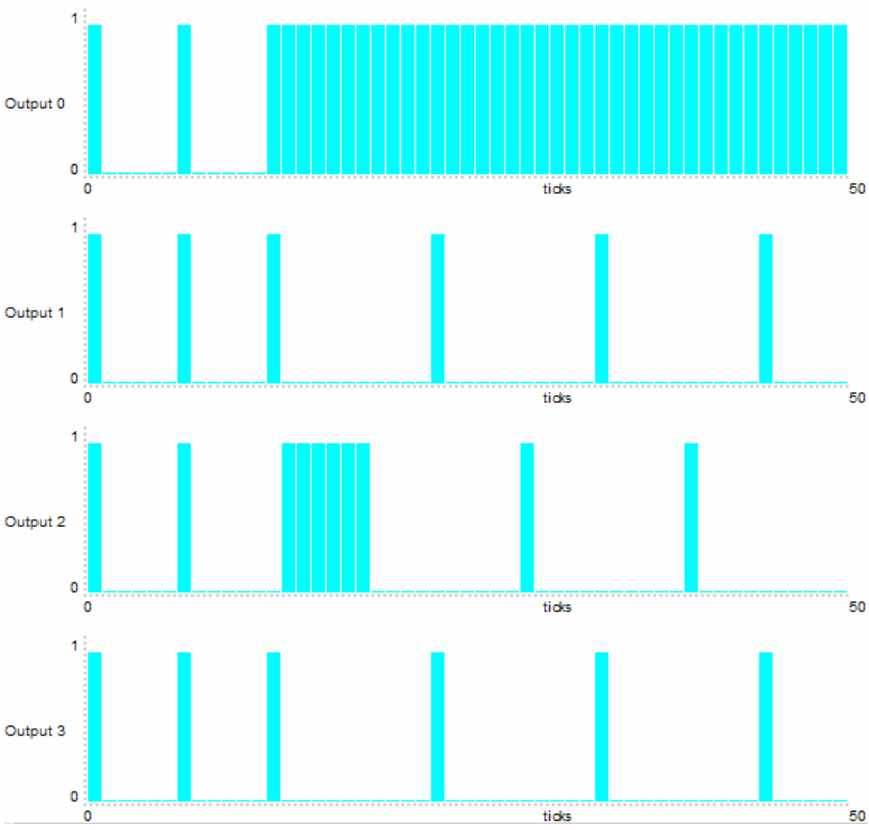


Figure 8.23b – ABN$_F$-GA output ticks – pattern 0

When presented with a pattern, the 4 output nodes produce different numbers of pulses. This is a result of the ABN_F having a variable pulse-duration. Examining the tick count shows that each pattern presented undergoes the same tick count, as every node continues to pulse until all nodes have relaxed.

One consequence of the nodes producing the same number of ticks, is that some nodes are in mid-pulse when the last ABN_F node relaxes, so that any partial pulse information is discarded.

Cyclically-stable behaviour occurs in some of the nodes, see figure G.28 for node 1 and figure G.29 for node 2. In the first case there is a clear 5 pulse cycle with one pulse of 0.4 followed by 4 pulses of 1.0. This should give an output average (mean) of 0.808, however the value was taken as 1.0, the relaxed value of the last two pulses based on pulse-width relax. The second case is different; the cyclic values are (0.3,0.7,0.5,0.7) and this gives an average value of 0.55 which is the value reported. This averaged pulse is added onto the end of the pulse output.

The relaxed evaluation takes priority over the cyclic-stability. The results of which is longer cycles or, for those with little fluctuation, the average (mean) value is replaced by average (mode) value.

8.5.3 Comparison of ABN_F – GA and ABN_w – GA

The pulse-width ABN_F-GA was evaluated on four criteria; relaxation time, minimum error, evolutionary time, and signal-pathway strength. All the relevant ABN_F and ABN_w parameters were equivalent. These included 3 hidden nodes, an α of 0.9 and a ρ of 1.0. The target e_{ABN} was {0.5}.

8.5.4 ABN_F – Trained using a GA - Relaxation Time

The ABN_F-GA continues to pulse until all output nodes have relaxed. To determine whether the ABN_F has relaxed, an extra pulse is required on single outputs and an extra cycle for cyclic-stability. These are an observer requirement, not an ABN_F requirement. Given this, an observation of the ABN_w showed a range of (3,7) pulses (equating to 30 to

70 ticks) for an individual node and pattern, with an average in the range (3,4). The ABN_F had a range of (38,93) ticks with an average of 49. This shows that the ABN_F performs in a similar timescale to the ABN_w.

8.5.5 ABN_F – Trained using a GA - Minimum Error Achieved

The ABN_w achieved an average e_{ABN} of 0.07 when pursuing a target of 0.5. The ABN_F achieved an average e_{ABN} of 0.195. Both of these networks show a tendency to over-fit the problem.

8.5.6 ABN_F – Trained using a GA - Training Time

The evolutionary training time for the ABN_w had a range of (53,3262) generations with an average (rounded mean) of 553. The ABN_F range was (26,632) generations with an average (rounded mean) of 305.

The ABN_F trained in fewer generations when successful, however, while the ABN_w always found a solution, the ABN_F occasionally reached the maximum permitted generation count.

8.5.7 ABN_F – Trained using a GA - Signal-Pathway Strength

Recall that for the ABN_w, the average ranges were;

- hidden pathways 123 (-60 to +63)
- output pathways 37.7 (-27.1 to 10.6)

The ABN_F produces;

- hidden pathways 118.8 (-48.6 to +70.2)
- output pathways 28.3 (-19.3 to 9)

As both ABNs have a similar range of signal-pathway values, this indicates that a network of mixed nodes may be possible.

8.5.8 ABN_F – Trained using a GA - Noise Tolerance

Noise has greater effect on ABN_F performance than on the previous ABN_w. Examples of noise effects are shown and discussed, (see figures 8.24 and G.34 to G.41).

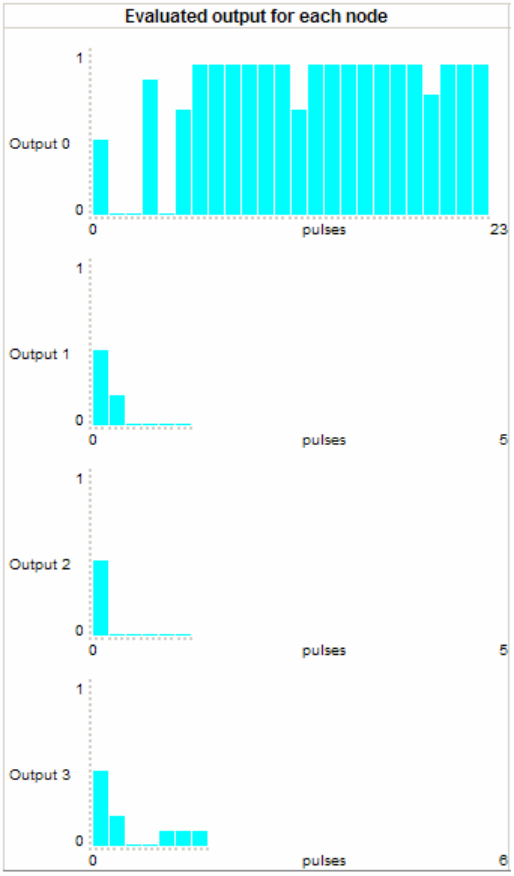


Figure 8.24a – ABN_F-GA output pulse - target e_{ABN} 0.5 - noise 0% - pattern 0

The pulse outputs show relaxed values, having resolved all cyclic-stability behaviour. The overall performance illustrated, in figures 8.24a and G.34 to G.36, shows an error of 0.1 for pattern 0 and pattern 3, indicating a relatively stable performance.

The pulses are resolved from the ticks shown in figure 8.24b. These are more difficult to interpret than ABN_w ticks and give an example of the differing pulse stream a node can produce.

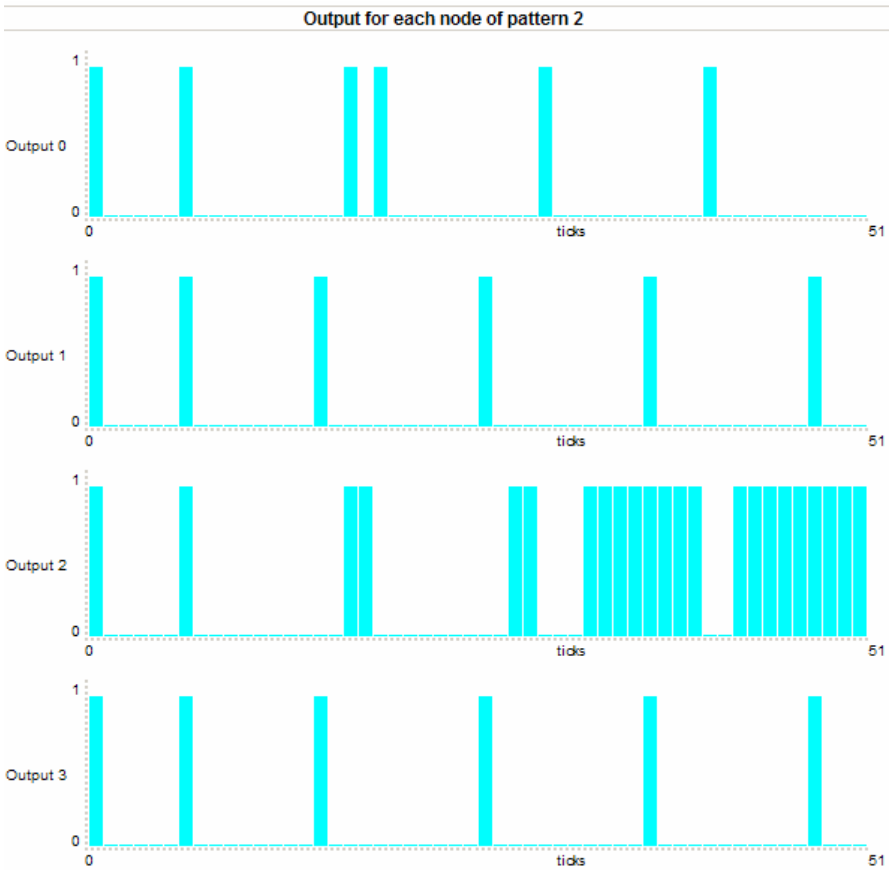


Figure 8.24b - ABN_F-GA output ticks - target e_{ABN} 0.5 - noise 0% - pattern 3

The ABN_F is affected by low levels of noise, beginning at 5%.

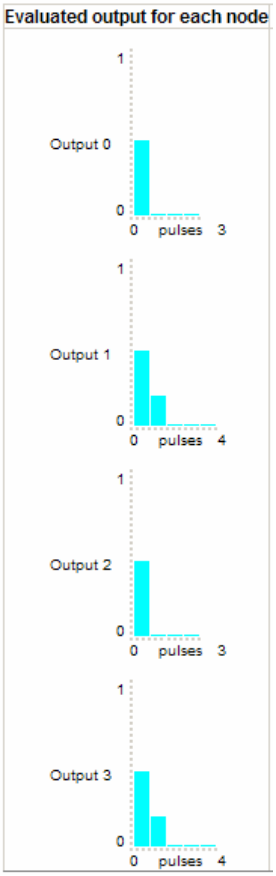


Figure 8.24c – ABN_F-GA output pulse - target e_{ABN} 0.5 - noise 5% - pattern 0

The output from pattern 0 is shown in figure 8.24c. When compared with figure 8.24a, it can be seen that the ABN_F does not recognised it. All other patterns are recognised with no e_{ABN} change. This is similar to the critical failure that occurred with the ABN_w-GA, however the noise tolerance is at a much lower level.

As noise increases to 15%, e_{ABN} reduces. The output for pattern 2, node 0 and 3, shows a reduced time to relax. This is shown in figures G.37 and G.38 and is a result of converting a cyclically-stable output to a relaxed output (on pattern 3), resulting in faster relaxation for the ABN_F as the other nodes relaxed in fewer ticks.

The ABN_F is unaffected by increase in noise until 25%. A cyclic-stable output is produced by a previously relaxed node, (node 1, pattern 0), as shown in figure G.39.

Various increases in noise affect the e_{ABN}. At 46% noise, the network is unable to differentiate between patterns; at 36% noise, cyclic behaviour occurs with extremely variable in-cycle pulses, figures G.40 and G.41.

The effects of noise were tested on several events and showed that the ABN_w is far more tolerant of noise than the ABN_F. This may be due to the instability of the ABN_F's cyclic-behaviour.

The previous effect of the slope ρ was considered and tested. There was no notable effect on the noise tolerance of the ABN_F-GA, which is in keeping with ABN_w-GA performance. The maximum pulse-duration was examined to assess quantisation. This is more detrimental to the ABN_F's training time, as it is assessed every tick, than to the ABN_w's training time.

8.5.9 ABN_F – Trained using a GA - Summary

In general the ABN_F is capable of the same performance as the ABN_w; however, as it produces cyclically-stable as well as relaxed outputs, it produces greater variation in its behaviour. As a consequence the ABN_w has a higher noise tolerance and is computationally less demanding than the ABN_F. Training factors, such as number of generations, favour the ABN_F slightly. The similarity in signal-pathway behaviour indicates that an ABN of combined node types is possible.

Both ABNs are viable alternative pattern recognition networks to the MLP. Their advantage is that they can be used in time domain problems as well, while the MLP must take a static view of time data.

8.6 Universal-Pulsing ABN

ABN success in pulse width and frequency modulated pattern recognition is the first part of a universal solution - ABN_U - in addition the production of a time-domain signal is desired. For this thesis, a robot walking gait time-domain signal was chosen, (in keeping with the author's research group's area of expertise) and it represents a general wave-form generation.

A walking gait is defined by the sequence of limb movements, the duration of each stride and the speed of each limb movement. To do this a network must produce a number of pulse-frequency "spikes" (amplitude 1 values) arriving in a time period.

A pulse-width signal could determine both limb sequence and stride duration; however, the stride would move at constant (maximum) speed for the duration of the pulse as the amplitude is constant.

A pulse-frequency consisting of the correct number of frequency pulses at the correct time would produce the correct movement. From the previous sections it appears unlikely that an ABN_F node can be this responsive on its own.

The implementation co-ordinates a different pulse-width for each gait and an associated pulse-frequency. The co-ordination of the different limbs is an effect of the ABN_U which consists of a mixture of the previous nodes and their variants.

8.6.1 Walking Gait

In a quadruped walking gait, a single limb moves at a time. There are many possible topologies for achieving any of the gaits, only one is required here. This gait is achievable in a three layer ABN_U. In the figure 8.25a, the limbs are shown as FR-front right, FL-front left, RR-rear right, and RL-rear left.

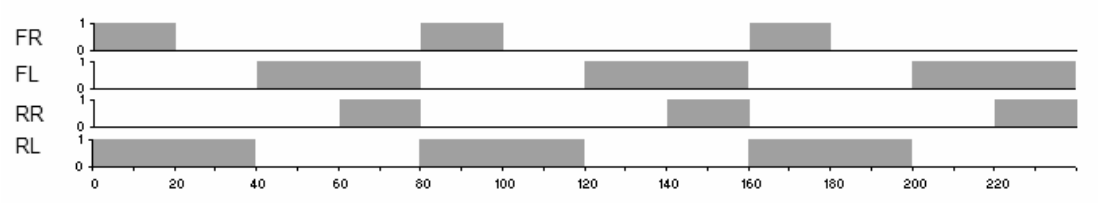


Figure 8.25a – Walking-gait - layer 1 outputs

In this example the pulse-duration is 80 ticks. The 4 each nodes produce a control pulse-width signal. If pattern value interpreted, from top to bottom, the nodes produce evaluated outputs of (0.25, -0.5, -0.25, 0.5). The pulse can generate a leading 0 or 1 amplitude.

The second layer combines the layer 1 signals, allowing pulse separation. This permits both a leading and trailing amplitude of 0.

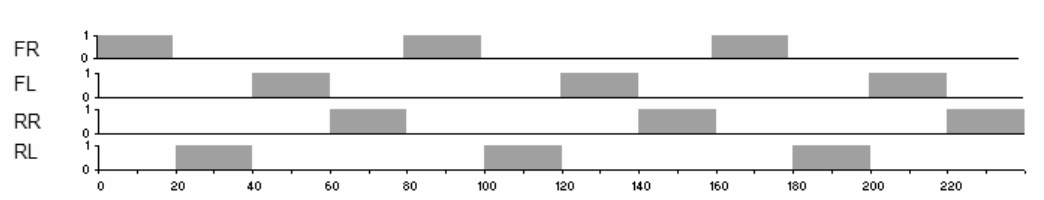


Figure 8.25b – Walking-gait - layer 2 outputs

This allows both limb sequence and stride duration to be produced by the ABN_U, while stride speed is still required. For this a pace signal is required, (see figure 8.25c).

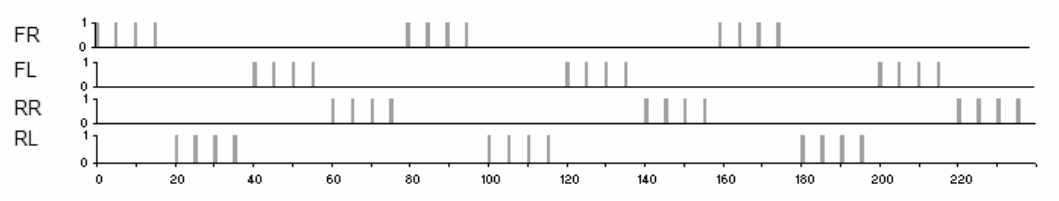


Figure 8.25c – Walking-gait - layer 3 outputs

It does not really matter when the pulse-frequency (speed) signal is integrated with the pulse-width (duration) signal. In this example, a third layer is used for a simpler demonstration.

The walking gait is the most difficult to generate as it requires independent movement of all limbs.

8.6.2 Trotting Gait

For a trotting gait, the limbs move in diagonally opposing pairs. Due to the limb pairing and stride symmetry, this is a achievable in a two layer ABN_U.

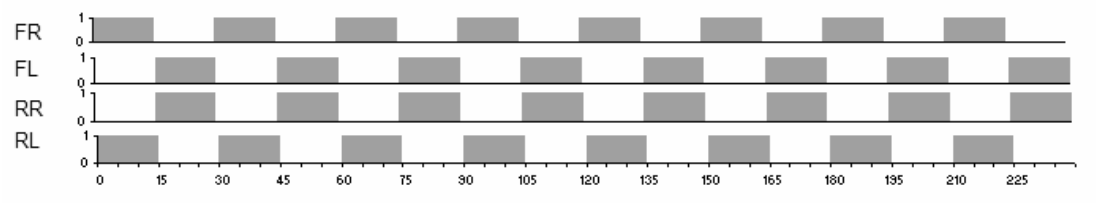


Figure 8.26a – Trotting-gait - layer 1 outputs

This example uses a pulse-duration of 30 ticks, (figure 8.26a). The nodes produce outputs of (0.5, -0.5, -0.5, 0.5). The second layer provides the pace signal.

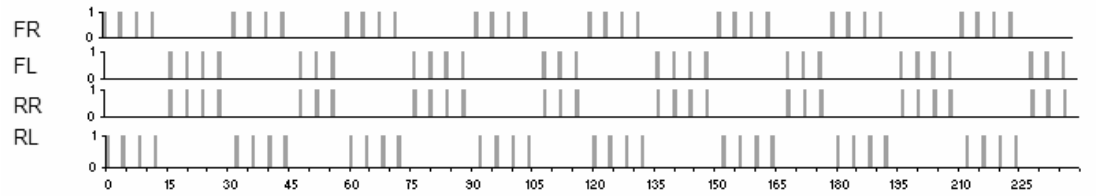


Figure 8.26b – Trotting-gait - layer 2 outputs

Limbs (FL,RR) move slower than the other pair, 7 pulses for every 8. This is caused when there is an imbalance in pulse-width and pulse-frequency. Components of a biological system would mask this with its more complex generation system (far more pulses and using thousands of ticks per second). This is an example of the trade off between universality and functionality, placing too much importance on single nodes. Increasing the number of ticks in a pulse-width reduces the effect. An alternative is selecting an

appropriate pulse-duration to include the missing 8th pulse, figure 8.26c. If more pulse frequency nodes are included then the effect can be countered without having to adjust pulse-duration, figure 8.26d.

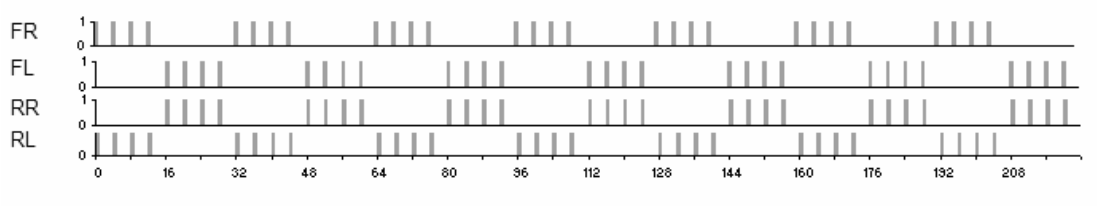


Figure 8.26c – Trotting-gait pulse - duration 32 - layer 2 outputs

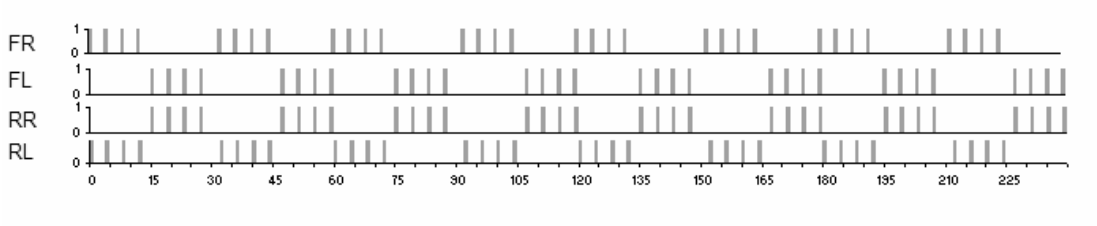


Figure 8.26d – Trotting-gait – additional pulse frequency nodes - layer 2 outputs

8.6.3 Gallop Gait

The galloping gait also pairs the limbs, front and rear. The only animal with a true gallop is the salt water crocodile as all other creatures have a delay between the paired limbs. This gait is of the same complexity as the trot, with different timings.

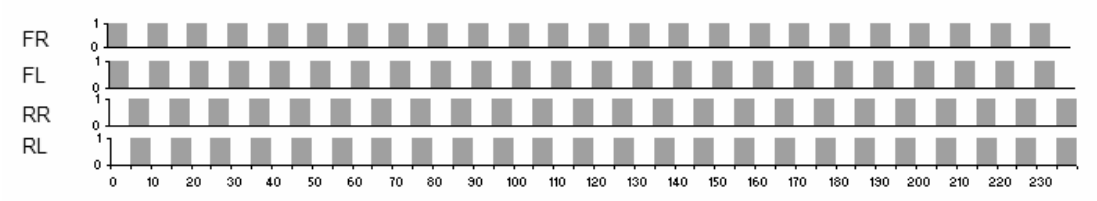


Figure 8.27a – galloping-gait - layer 1 outputs

This example uses a pulse-duration of 10 ticks, (figure 8.27a). The nodes produce outputs (0.5, 0.5, -0.5, -0.5).

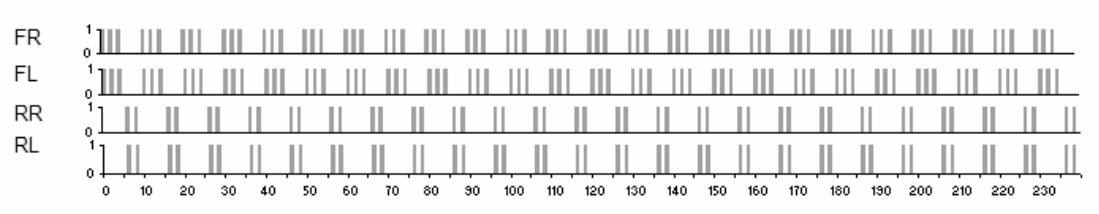


Figure 8.27b – Galloping-gait - layer 2 outputs

The second layer provides the pace signal. The addition of this layer produces the outputs in figure 8.27b. The same imbalance occurred with the trot recurs with the gallop; in this case the front limbs move faster than the rear. It is possible that this is actually required by a robot. All quadrupeds do not have a symmetric front rear body shape and therefore the front or rear limbs may be more powerful/move at different speed to compensate. As before this can be equalised with a change to pulse-duration or additional frequency nodes, (see figures 8.27c and 8.27d).

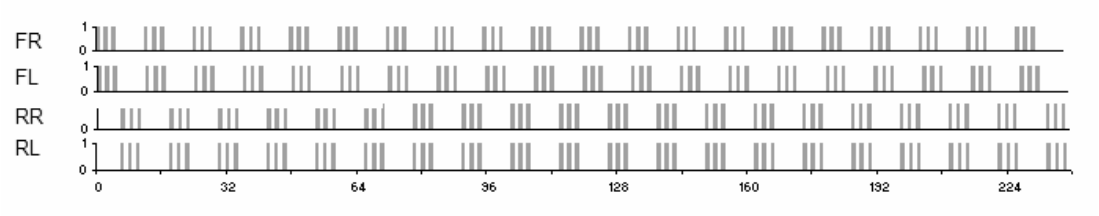


Figure 8.27c – Galloping-gait pulse duration 12 - layer 2 outputs

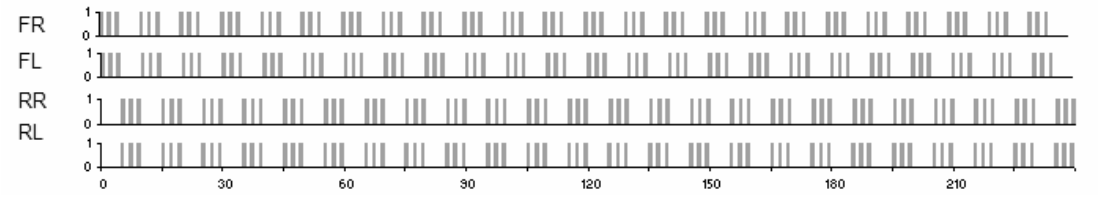


Figure 8.27d – Galloping-gait – additional pulse frequency nodes - layer 2 outputs

8.6.4 Universal-Pulsing ABN - Summary

All required locomotion gaits have been produced by the ABN_U (using pulse-width, pulse-frequency and their inverse pulse modulations).

The pattern recognition abilities of the ABNs are equivalent to that of the MLP; however, MLP networks have difficulty producing suitable time-domain control outputs (for example for pulse-width modulated motor control).

As Artificial BioChemical Networks can perform pattern recognition, they do not require other devices to construct a complete recognition-control system. Artificial BioChemical Networks have inherently time-domain functionality and do not have this disadvantage. They may therefore be trained to control the gaits of a simulated quadruped robot. The robot uses servo motors to control limb movement, (figure 8.28). These limbs have one active and one passive degree of freedom.

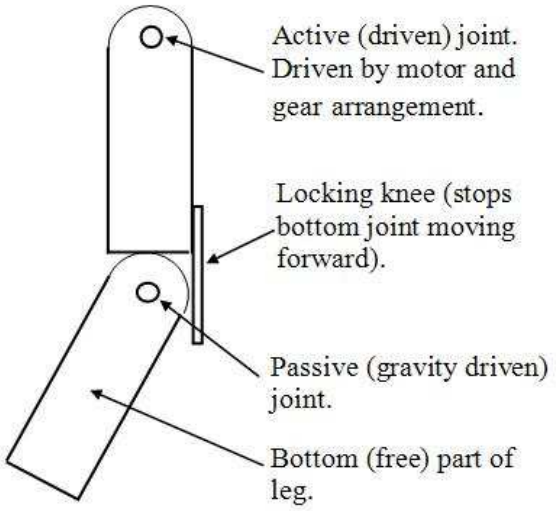


Figure 8.28 – Robot leg layout

This simulation has been used and reported many times previously. The dynamics of the legs and the robot are fully reported by Muthuraman et al., [2003] and McMinn [2002]. Figure 8.29 shows limb movements generated when the network was evolved to walk. The result corresponds well with the perfect pattern (a perfect pattern would have a repeat time of 60 time steps and a movement from position 80 to position 100).

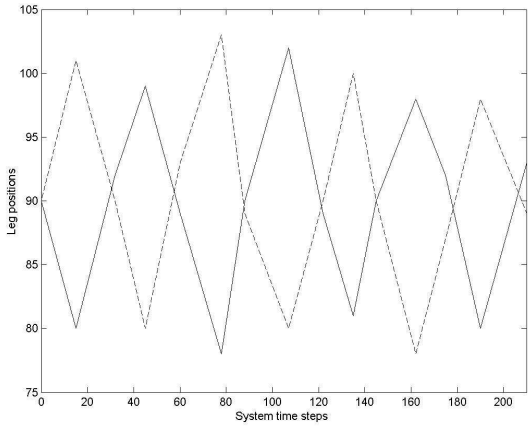


Figure 8.29 – Movement pattern of legs

8.7 Modular ABNs

The abilities of ABNs to perform pattern recognition and produce time-domain signals show that they are as functional as ANNs. ABNs have the advantage of being able to use both time-domain and spatial-domain data. This presents an alternative approach to connectionism. In addition, these behaviours are performed by the same unit types and the same topology type. ANN functionality typically comprises different units and topologies.

Modular ANNs have been extensively researched by the author's group, as in McMinn [2002] and Muthuraman [2005] and have demonstrated capabilities beyond that of individual ANNs. ABN modules have been connected together to produce a control signal response to a recognition data input.

The ABN modules can be connected in the same manner as ANN modules, which is straightforward. Outputs of a pattern-recognition ABN are the inputs of the control signal ABN, (figure 8.30a). If specific recognition outputs are suitable as control module inputs, either the recognition module is trained with new targets or a translation module is placed between them, (figure 8.30b).

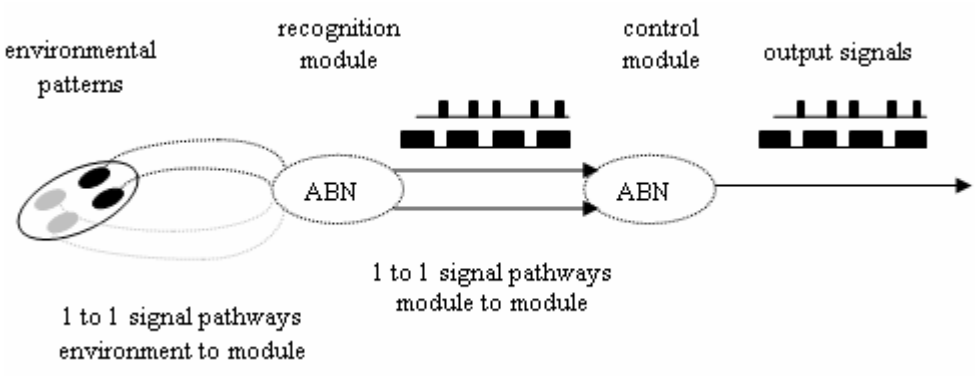


Figure 8.30a – Recognition - control modules ABN

In the two module ABN_U system, the environment inputs are continuous signals of fixed but different amplitude. The recognition module outputs are produced as time-domain pulses. These are a combination of pulse-width and pulse-frequency, depending on the nodes utilised. They arrive as control ABN_U inputs with time-domain behaviour.

When the control ABN inputs are of fixed amplitude but time variable, they are normalised by the input layer of the control ABN. Therefore second and later modules in the system receive information in the format that their input layer would expect for environmental inputs. This can cause problems with cyclic behaviour and so a translation module is used which produces the control output values.

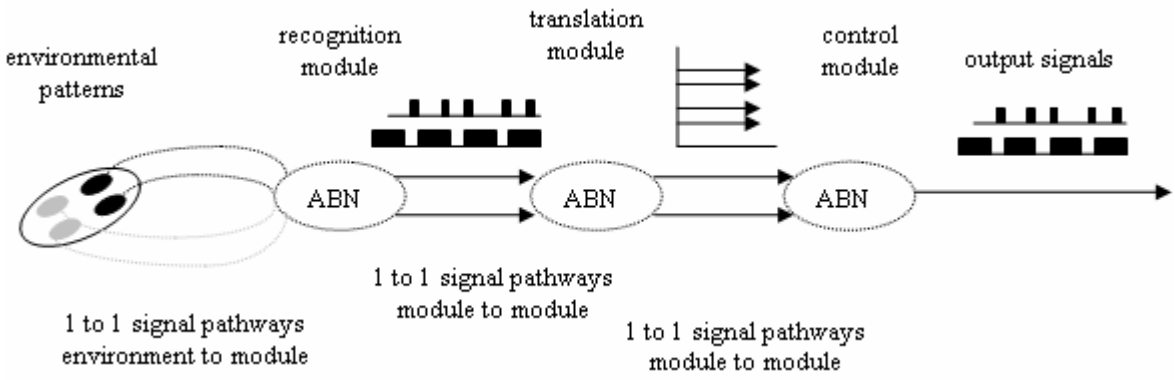


Figure 8.30b – Recognition - translation - control modules ABN

In the three module ABN_U system the operation remains consistent, however the recognition module output, with variable pulse-width and amplitude 1, is translated to a constant output of amplitude (0,1).

The difference between these module arrangements is that the first produces a signal that was relaxed for the first module before it was permitted to progress, while the second allows the ABN_U system to relax as a whole. The second is preferred as it is unsupervised free in determining signal progression; it is the method implemented in this thesis.

8.8 Summary

The ABN systems discussed in this chapter are a new and different approach to connectionist AI. Instead of a neural networks basis, they model the chemical signalling within cells. Of course, as observed, such signalling lies at the root of neuron functionality also, as the neuron is itself a cell.

The retention of generalisation and universality as discussed by Capanni et al., [2003] affects the ABN performance in pattern recognition and control systems, allowing for graceful decay as noise increases. Such "fuzzy" uncertainty is far more stable than a system that performs longer with higher accuracy then undergoes critical failure with little warning.

With regards to mobile robot operation there is a functional advantage of ABN pattern recognition. Most pattern recognition is achieved "in vitro" where time is not a constraining factor; here "snapshot" pattern recognition can be utilised. In an artificial organism that has to adapt to its environment "in vivo" then an ABN information flow pattern system can assimilate information as it appears.

The implementation of universal ABNs allows a single type of intelligent unit to perform all the operations of a modular AI used in robot control, and can be encoded as part of the evolutionary algorithm. This can be achieved without an operator placing specific units as shown in the systems presented by Muthuraman [2005]. In these he noted the importance of unit functionality without which (depending on the module purpose), specific units are required.

The ABN approach detailed here has several advantages. It simplifies the design of time dependant outputs which, in turn, allows the straightforward implementation of Central

Pattern Generator networks in robots, pulse-width modulation for motor control and other similar systems. However, the ABN networks are equally at home in traditional pattern-recognition tasks. They also allow systems to be developed which behave in many respects like spiking neuron models, but without the associated complexity.

Finally, ABNs may be trained using traditional methods and are suited to the development of new methods based on known training algorithms.

Chapter 9
Further Work

9.1 Introduction to the Chapter

There are five main topics in this chapter. Firstly, areas of exploration from the development of the Taylor Series networks. Secondly, investigations into Artificial Neural Network functionalities that were observed during the TS research. Thirdly, memory in connectionist networks. Fourthly, outstanding exploration in Artificial BioChemical Networks. Finally, combinations of TS and ABN techniques.

9.2 Taylor Series SLT and MLT

The problems presented to the Single-Layer Taylor Series network were solved with 3rd order TS neurons, and increasing the order beyond this gave no advantage. One direction for further work would be an investigation into more difficult problems, to assess if 4th order (or higher) terms can prove an advantage.

Firstly, the relationship of these results should be compared with the known advantages that 3rd order SLT networks show over 2nd. If higher order terms continue to show little or no effect until the network has achieved a low error, then an algorithm could be developed to perform initial training on 1st order terms and only introduce the higher orders as training improvements decrease. These higher order terms could be initialised with a weights range based on observing the trained state of such networks (as they may have a different profile).

Secondly, there is also the influence of the factorial divisor to consider. These factorial divisors reduce the influence that their weights may have. As it is, a 4th order weight has ¼ the effect of a 3rd order weight of the same magnitude. Given the dynamics of ANN training, it may be possible to discard this divisor and to allow the weights to accommodate the effect, with an appropriate initialisation step.

Thirdly, any advantage of a divisor could be assessed against the training overheads of the additional computations entailed. This is based on the observation that once a network is trained, any divisors could be incorporated into their associated weights before use. This would remove any later computation on these divisors and network performance would be identical.

The proposed research on increasing orders of power with regard to training time improvements should also take into consideration the effect of noise tolerance given any changes in network training performance.

9.3 ANN Performance – Noise, Targets and Validation

In general ANN performance, the advantages of noise tolerance, with regards to target setting have been well demonstrated. These merit additional investigation. ANNs have been extensively researched and there is considerable work by other authors in this area; however, subject to a literature review, a close look at pattern-target relationships with regards to noise tolerance and overtraining would be in order. This would be evaluated against validation trained networks.

9.4 Displaced Equilibrium – Memory in Connectionist Systems

As part of the project work in the thesis, a system was investigated where an ANN was evolved to achieve a partial success in a problem. The particular problem was to associate different walking performance with input parameters. This evolutionary training was equivalent to the hard-wired biological component referred to as “genetic memory”, which allows organisms to survive in their initial environment.

The next phase was to introduce a learning algorithm that would minimise the error in the environment to achieve efficiency in walking. This differed from most learning algorithms in that the ANN learned while it performed.

The algorithm made use of several components that were evolved variables. A learning rate amplified any changes. A Hebbian (and anti-Hebbian) variable affected any active

connections. A synchronous variable affected any active connection when other connections were also active while a mediated contribution affected a connection (active or not) if associated connections were active. A bio-chemical feedback acted as an error to the entire ANN.

When this was implemented, the ANN was able to function on introduction to the environment and adapt to its maximum efficiency.

Once the input parameters were altered, to simulate a different walking environment, the learning algorithm allowed the ANN to adapt to the new environment as well, through altering its connection values. If the input change was too great (such as a radically different environment) then the ANN was unable to adapt.

Once the ANN was returned to its original environment, the learning algorithm adapted the ANN once more. This returned the connection values to the same as before, hence the term “displaced equilibrium”.

There were two observations that prevented this work from being included in the thesis. Firstly, a type of artificial amnesia developed. As the connection values shifted from one environment to another, some were permitted to break (a feature of the algorithm). These broken connections were never reformed and the ANN adapted by finding alternative solutions to a retuned environment. Eventually, too many connections were lost and the ANN could no longer adapt to changes. This is not an endpoint for this research, as adaptations to the original algorithm or connection formation could be used. Although it had been decided at this point that this was of no direct benefit to the project, future work in the area was considered worthwhile.

Secondly, when compared to biological systems there remains a problem in that the entire ANN is involved in all its activities. That a fully connected system is limited in development was observed and supported by the ongoing work of the research group in modular networks. It was therefore decided that this research should be later assigned to a modular development.

9.5 ABN Design

As the ABNs were introduced as a new concept, various opportunities for explorative research presented themselves during their construction; while not an exhaustive list, those that showed potential are included here.

9.5.1 Topology

To allow comparisons, the topologies of the ABNs were constructed in a similar manner to ANNs. These ABN topologies are fixed structures and do not permit adaptable changes in topology whereas biological systems do, both during initial development and through their life span. The thesis introduced the ABN concept as an alternative AI technique and accepted some initial restrictions to do so. It is intended that nodes may develop peripatetic behaviour and a project examining a hybrid Swarm-ABN system is proposed to this effect.

9.5.2 Pulse Time

The following method is suggested for future work and integrates the pulse information Sum *S* into t_{on} and incorporates the previous t_{on} value and a leaky integration *LI* factor alpha α.

$$t_{on} += (\beta \cdot \sigma(S_t) \cdot PW) - \alpha(LI) \qquad \text{equation 9.1}$$

Previous nodes have not altered the pulse parameters after a pulse has commenced, this is in keeping with biological neuron pulses. As biological neurons are pulse-frequency and not pulse-width there is little to alter. This is not the case with biochemical signalling, where the protein parameters are part of ongoing processes that may dynamically change. Allowing the pulse-width to vary after it has commenced a cycle would result in a variation in pulse-duration and introduce non-synchronisation in the ABN_w.

9.5.3 Amplitude Modulation

Biological neuron signals are not amplitude modulated but are time modulated, while artificial neurons such as McCulloch-Pitts are amplitude modulated but not time modulated. Biochemical signal-pathways can, in addition to time modulated signals, produce an amplitude factor by increasing the quantities of protein in the system. Current implementations of ABNs seek to model the time modulated signals but do not incorporate amplitude modulation. This has been due to the problem domain, where only one degree of freedom is incorporated into the input data and hence one type of signal modulation is performed by the ABN. Amplitude modulation may be a property of the ABN rather than the node, as adding such functionality contributes greater degrees of freedom to the output than are supplied by the input. Changes in the problem domain may be required to accommodate this.

Amplitude over time was a consideration when attempting to produce walking gaits. The gait was achieved through combining pulse-width and pulse-frequency modulation. It may be that gait transition, such as moving from a walk to a run, would benefit from such work. However, it is suspected that it will be more important in allowing speed variable gaits. For example, bipedal gaits are more similar in profile than those of quadrupeds; but for each there must be some method of signalling the power behind the limb as creatures can walk, trot, canter, pace and gallop at various speeds. This is the only way that creatures of differing physical size keep pace with each other.

Different amplitudes allow the possibility of using a “0” as the “no signal” state, the “+1” as the active move state - which lifts a limb forward and “-1” as the recovery state – which returns a limb to the initial position. Although biological systems signals do not operate this way, it may be conducive to artificial movement.

9.5.4 Improvements to ABN_w Backpropagation

Although the method used proved successful for the ABN_w there are some improvements possible, especially with regard to the use of the relevant hidden pulses.

The pulse from the hidden layer that was used to alter the weights for the output $pulse_n$ was hidden $pulse_{n-1}$. This is correct, but of limited accuracy.

The output $pulse_n$ is produced from Sum_n at the output node, which has been accumulated by the arriving of $pulse_0$ to $pulse_{n-1}$ from the hidden layer, the effect of each accumulated value being diminished by the factor α at every tick.

$$Sum_{tick=2} = \alpha \cdot Sum_{tick=1} + PulseAmplitude \cdot PathwayStrength \qquad \text{equation 9.2}$$

- When $pulse_{n-1}$ has fully arrived the effect of the leading edge has diminished by α^{10}. Given an α of 0.9, this is approximately 0.35.
- The effect on the trailing edge of the previous hidden pulse is diminished by α^{11} and the leading edge by α^{20}.
- This effect continues until the first pulse is computed.

As can be seen the more recent a pulse is the more significant its effect. A cumulative term CT could therefore be used instead of the hidden $pulse_{n-1}$.

Given this, it must be considered why Backpropagation works using hidden $pulse_{n-1}$ instead of CT. This may be due to the error being moved in the correct direction, using a reduced value, akin to implementing a learning rate < 1.0 (assuming CT is greater than $pulse_{n-1}$). There is also the consideration that the hidden $pulse_{n-1}$ may actually approximate CT. This is due to the leading edge of the pulse having an amplitude of "1" and the trailing edge having the amplitude "0".

9.6 Taylor-Series Functionality with ABNs

In this thesis there were two different approaches to unit functionality. Both were successful, one in the static-domain and one in the time-domain. As the TS neuron operated on the summation function and the ABN node operated on the output function, there is the potential to combine both of these and assess how they work across both domains. Although this is beyond the scope of this thesis, it is a logical next step.

Chapter 10
Conclusions

10.1 Introduction to the Chapter

This chapter presents the conclusions of the project. The original objectives, as described in Chapter 1, are revisited with reference to the work presented in this thesis. Then a discussion of the original contributions follows. A summary of the main findings and further work is made. Finally, some concluding remarks concerning the success of the project are made.

10.2 Project Objectives Revisited

The objectives as stated in Chapter 1 were:

1. To review the literature on the subject of generalised Artificial Neural Networks
2. To review the biological relationship of the work
3. To develop an appropriate generalised neural model
4. To extend the function of the above to time domain behaviour
5. To compare these results with published and standard data
6. To integrate these models into a complete neural system
7. To apply this system to a standard problem
8. To compare these results with previously published material

These can now be considered in terms of what was achieved.

10.2.1 To Review the Literature

The initial background reading and study, that was necessary to understand the purpose of the project, was undertaken at the beginning of the research. The main examination began with the work which was later used in "Evolution and Devolved Action" [MacLeod et al., 2002], (Appendix B), which this author contributed to. Study then centred on various

recommended AI textbooks, included in the bibliography, and continued with a review of the work of McMinn [2002]. As the project developed a continuous review of appropriate literature, including Muthuraman [2005], contributed to the body of knowledge.

10.2.2 To Review the Biological Relationship of the Work

The author examined the biological basis of Artificial Intelligence and centred on genetics and brain function. As the project developed away from traditional connectionism approaches, a greater emphasis was placed on biochemistry and the intelligence expressed by single-celled organisms.

10.2.3 To Develop an Appropriate Generalised Neural Model

An appropriate generalised neural model was developed, described in Chapters 4 and 5, using a Taylor Series expansion. The generalisation capability of the TS neuron was explored and an associated investigation on its universality was completed. These both gave favourable results for the new model. Once the neuron had been fully investigated, the model was integrated into network topologies and trained with a Genetic Algorithm. Additionally, beyond the requirements, it was shown that the model could be used with a standard learning algorithm, (see Appendix C). The functionality of the neuron was then explored in the networks it was added to, which again resulted in a favourable performance. This work led to a publication [Capanni et al., 2003], shown in Appendix A.

10.2.4 To extend the Function to Time-Domain Behaviour

The TS models were implemented as neural oscillators, which produced some interesting results. Their limitations were noted and although they may merit further work, they did not show sufficiently promising results to include a specific section. Instead, this investigation inspired the later successful research into the alternative connectionist system ABN.

10.2.5 Compare Results with Published and Standard Data

A comparison of the results with standard data and ANNs was made. This included a single neuron solution to the parity-bit problem [Minsky and Papert, 1969], which demonstrated the improved functionality of the TS neuron over the McCulloch-Pitts neuron. The generalisation and universality of the TS networks, single and multi-layer, were compared with that of the Single-Layer Perceptron and the Multi-Layer Perceptron. In these comparisons the new model showed several advantages, including training time, network size and noise tolerance. This was presented in Chapter 5.

10.2.6 Integrate Models into a Complete Neural System

Due to the investigation into the time-domain behaviour, the integration instead led to the investigation into alternative connectionist approaches. This resulted in the proposal of an Artificial BioChemical Network, (Chapter 6), after extensive research into biological intelligence.

This model was developed as a complete connectionist system for time-domain problems and a series of experiments and comparisons were set out, (Chapter 7).

The functionality of this model was examined and compared against standard ANN types. The same capabilities were examined in the spatial-domain as before and the new model performed successfully. Then the same ABN models were tasked with the production of time-domain behaviour that their ANN competitors found difficult or impossible to produce, and once more produced successful results, (Chapter 8).

10.2.7 Apply this System to a Standard Problem

This new connectionist model was applied to the task of producing locomotion gaits for robots, as had the previous models by McMinn [2002] and Muthuraman [2005]. It was not necessary to compare against these previous approaches as the project had been developed using the lessons learned from them as a direction for study. The model was able to produce the required locomotion gaits and complete the objective requirements. This was achieved with a single unit type (ABN) rather than designed neurons, (Chapter 8).

10.2.8 Compare Results with Previously Published Material

The development and comparison of the new model's capabilities was compared at the appropriate stages throughout the thesis. This showed the various advantages of the new model in its ability to interpret both spatial-domain data and time-domain data. This work was completed by the model's ability to process data across both domains, without the requirement for a translation system. The results from the tests on the model's functionality and comparisons of data (sections 10.2.6 to 10.2.8) contributed to a publication by Capanni et al., [2005].

10.3 Novel Aspects of this Research

The new contributions of this research are as follows:

- A new approach to connectionism based on the biochemistry of single celled organisms.

  This approach, Artificial BioChemical Networks, is the primary contribution in this thesis. It has produced new time-domain units and network paradigms. These have performed well when compared against standard, thoroughly researched and developed, ANN models. The new models are in their infancy and they have tremendous potential for further development. This work is presented in Chapters 6, 7 and 8.

- A highly functional advance to the neuron model based on the Taylor Series approach, (Chapters 4 and 5).

  A new model was introduced, based on mathematical theory, which was then shown to be highly, but controllably, functional as a neuron when compared to the previous model. This was demonstrated by a solution to the parity-bit problem which the McCulloch-Pitts neuron is incapable of producing.

- A comprehensive theoretical and experimental consideration of the mapping abilities of neurons in the spatial-domain, (Chapter 5).

The integration of the new highly functional model into networks showed that the neuron's capabilities could be implemented using traditional learning algorithms, with the potential for further improvement.

- Demonstrations of these models in modular connectionist networks. As described in Chapter 8, the capabilities of the new models have been shown in modular units which can produce the relevant outputs to communicate and build into a modular connectionist system, processing both spatial and time-domain information.

- A consideration and investigation of neural functionality in the context of robotic systems, presented in Chapters 7 and 8. As shown, the capabilities of the new models have been shown to be of real value in practical implementations.

- A basis for further research into learning, modular networks and time-domain connectionism, presented as part of the further work section, (Chapter 9).

10.4 Summary of Suggested Further Work

- Investigation of higher order problems solvable by the Taylor Series SLT and MLT.
- Production of order specific learning algorithms for the SLT and MLT.
- Investigation of specific target selection towards improvement in ANN noise tolerance.
- An investigation into adaptable memory in connectionist systems to produce AIs that can adapt to changes in their environments.
- Further investigate memory in connectionist systems as a modular component
- Construct hybrid Swarm-ABN system.
- Investigate alternative methods of pulse timing in ABNs.
- Encode additional information through the inclusion of amplitude modulation in ABN systems.
- Develop and improve the ABN_w Backpropagation Algorithm.
- Combine the advanced in spatial-domain and time-domain functionality through the development of a Taylor Series ABN.

10.5 Concluding Remarks

The project has successfully incorporated and extended the findings of its own and concurrent research, the management of which has introduced the author to a greater understanding of research.

Although there have been difficulties to overcome in developing and complicating these objectives, mainly in turning away from dead ends instead of forcing an ineffectual path through, and in allowing new areas to be fully investigated by colleagues, the project has found its own purpose and has contributed new knowledge to the field.

In particular it has provided a viable foundation for a new type of universal unit for use in connectionist AI.

The author believes that the work in the areas of neural functionality and Artificial BioChemical Networks are useful contributions to connectionist research.

This thesis joins a body of work which furthers the implementation of Evolutionary Artificial Intelligence. It is hoped that the contributions of this research may be integrated with those of associated researchers to provide innovative and exciting intelligence capabilities in modular and diversely functional systems.

References

Chapter 1

MacLeod, C., McMinn, D., Reddipogu, A., and Capanni, N., 2002. Evolution by Devolved Action: Towards the Evolution of Systems. In Appendix B of McMinn, D., Using Evolutionary Artificial Neural Networks to Design Hierarchical Animat Nervous Systems, PhD thesis, The Robert Gordon University.

McMinn, D., 2002. *Using Evolutionary Artificial Neural Networks to Design Hierarchical Animat Nervous Systems.* PhD thesis, The Robert Gordon University.

Minsky, M. L. and Papert, S. A., 1969. *Perceptrons.* expanded ed., 1990. Cambridge, MA: MIT Press.

Muthuraman, S., 2005. *The Evolution of Modular Artificial Neural Networks.* PhD thesis, The Robert Gordon University.

Chapter 2

Azam, F., 2000. *Biologically Inspired Modular Neural Networks*, PhD thesis, Virginia Polytechnic Institute and State University.

Barron, A,. 1993. Universal Approximation Bounds for Superposition of a Sigmoidal Function. *IEEE Transactions of Information Theory.* Vol. 39, No. 3, pp. 930-945.

Bishop, C. M., 1995. *Neural Networks for Pattern Recognition*, 1st edition. Oxford University Press. pp 302-304.

Edelman, G. M., 1987. *Neural Darwinism: The Theory of Neuronal Group Selection*, 1st edition. New York: Basic Books. pp. 213.

Ewart, J. P., 1987. Neuroethology of Releasing Mechanisms: Prey Catching in Toads. *Behavioral and Brain Sciences*. Vol. 10, No. 3, pp. 337-367.

Grossberg, S., 1976. Adaptive Pattern Classification and Universal Recoding, I: Parallel Development and Coding of Neural Feature Detectors, *Biological Cybernetics*, Vol. 23, pp. 121-134.

Hornik, K., 1989. Multilayer Feedforward Networks are Universal Approximators, *Neural Networks*, Vol. 2 pp 359-366.

Lansner A., Kotaleski J. H. and Grillner S., 1998. Modeling the spinal neuronal circuitry underlying locomotion in a lower vertebrate. Annals of the New York Academy of Sciences : Neuronal Mechanisms for Generating Locomotor Activity, Vol. 860 pp 239-249.

Maas, W. and Bishop, C. M., 1999. *Pulsed Neural Networks*, 1st edition. Cambridge, MA: MIT Press.

MacLean, P.D., 1990. *The Triune Brain in Evolution : Role in Paleocerebral Functions*, 1st edition. Springer.

MacLeod, C., Maxwell, G.M., McMinn, D., 1998. A Framework for Evolution of an Animat Nervous System. In *Proc. of EUREL European Advanced Robotics Systems Development: Mobile Robots*. 1-10 September 1998. Leiria, Portugal. VolumePart Paper 18.

MacLeod, C., 1999. *The Synthesis of Artificial neural Networks using Single String Evolutionary Techniques.* PhD thesis, The Robert Gordon University.

MacLeod, C., McMinn, D., Reddipogu, A., and Capanni, N., 2002. Evolution by Devolved Action: Towards the Evolution of Systems. In Appendix B of McMinn, D., Using Evolutionary Artificial Neural Networks to Design Hierarchical Animat Nervous Systems, PhD thesis, The Robert Gordon University.

Martin, E., ed., 1976. *The Penguin Book of the Natural World*. Middlesex UK. Penguin: Harmondsworth.

McMinn, D., 2002. *Using Evolutionary Artificial Neural Networks to Design Hierarchical Animat Nervous Systems.* PhD thesis, The Robert Gordon University.

Muthuraman, S., 2005. *The Evolution of Modular Artificial Neural Networks.* PhD thesis, The Robert Gordon University.

Potter, M. A., De Jong, K. A., and Grefenstette, J., J., 1995. A coevolutionary approach to learning sequential decision rules. In *Proc. of the Sixth International Conference on Genetic Algorithms,* July 1995, San Mateo,CA. pp. 366-372.

Reddipogu, A., Maxwell, G. and MacLeod, C., 2002. An Innovative Neural Network Based on The Toad's Visual System. In: *Proc. of ACTIVS, Advanced Concepts for Intelligent Vision Systems.* 9-11 September 2002. Ghent, Belgium: Ghent University. pp. 144-149.

Reid, M. B., Spirkovska, L., Ochoa, E., 1989. Rapid Training of Higher-Order Neural Networks for Invariant Pattern Recognition. In *Proc. of IJCNN International Joint Conference on Neural Networks,* 4-9 May 1998, Anchorage, Alaska. Vol. 1 pp. 689–692.

Sima, J., Orponen, P., 2003. A taxonomy of neural network models - General Purpose Computation with Neural Networks A Survey of Complexity Theoretic Results. *Neural Computation.* Vol. 15, pp. 2727–2778

Thompson, A., 1996. Silicon evolution. In *Proc. of Genetic Programming,* 28-31 July 1996, Palo Alto. pp. 444-452.

Wasserman, P. D., 1989. *Neural computing: Theory and Practice*. New York: van Nostrand Reinhold. pp. 127.

Wilson, S. W., 1991. The Animat Path to AI. *Conferences in from animals to animats*.

Chapter 3

Bishop, C. M., 1995a. *Neural Networks for Pattern Recognition*, 1st edition. Oxford University Press.

Bishop, C. M., 1995b. *Neural Networks for Pattern Recognition*, 1st edition. Oxford University Press. pp. 11.

Blum, E. K. and Li, L. K., 1991. Approximation theory and feedforward networks. *Neural Networks*, Vol. 4, No. 4, pp. 511–515.

Briggs, F., 2005. *Universal Meta Optimization* [online] Available from: http://www.generation5.org/content/2004/UniversalMetaOptimization.asp, [Accessed 22 November 2005]

Capanni, N. F., MacLeod, C., Maxwell, G., 2003. An Approach to Evolvable Neural Functionality. In *Proc. of ICANN/ICONIP Joint International Conference on Artificial Neural Networks and International Conference on Neural Information Processing*. 26-29 June 2003. Istanbul, Turkey. Vol. 2, pp. 220-223.

Chang, C. and Cheung, J.Y., 1992. Backpropagation algorithm for higher order neural network. In *Proc. International Joint Conference Neural Networks*. 7-11 June 1992. Baltimore, MD. pp. 164–166.

Chen, M., 1991. *Analyses and Design of Multi-Layer Perceptron Using Polynomial Basis Functions*. PhD thesis, The University of Texas at Arlington.

Cybenko, G., 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems.* Vol. 2, No. 4, pp. 303-314.

De Figueiredo, R. J. P., 1980. Implications and applications of Kolmogorov's superposition theorem. *IEEE Transactions . Automation and Control.* pp 1227–1230.

Duch, W. and Jankowski, N., 1997. New neural transfer functions. *Applied Mathematics and Computer Science.* Vol. 7, pp. 639-658.

Elder IV, J. F., Brown, D. E., 1992. *Induction and Polynomial Networks*, IPC-TR-92-009, Institute for Parallel Computation and Department of Systems Engineering, University of Virginia, Charlottesville, VA. pp. 29.

Gurney, K., 1997a. *An Introduction to Neural Networks.* 1st edition. UCL Press.

Gurney, K., 1997b. *An Introduction to Neural Networks.* 1st edition. UCL Press. pp. 81.

Gurney, K., 1997c. *An Introduction to Neural Networks.* 1st edition. UCL Press. pp. 80.

Gurney, K., 1997d. *An Introduction to Neural Networks.* 1st edition. UCL Press. pp. 83-84.

Gurney, K., 1997e. *An Introduction to Neural Networks.* 1st edition. UCL Press. pp 84.

Hecht-Nielsen, R., 1987. Kolmogorov's mapping neural network existence theorem. *In Proc. of the International Conference on Neural Networks*. 1987. New York, Vol. III, pp. 11-14.

Heywood, M. and Noakes, P., 1996. A framework for improved training of Sigma-Pi networks. *IEEE Trans. Neural Networks*. Vol. 6, pp. 893–903.

Hornik, K., Stinchcombe, M. and White, H., 1989. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*. Vol. 2, pp. 359-366.

Kolmogorov, A.N., 1957. On the Representation of Continuous Functions of Several Variables by Superpositions of Continuous Functions of One Variable and Addition. *Dokladi*. Vol. 114. pp. 679-681.

Kurkov´a, V., 1992. Kolmogorov's theorem and multilayer neural networks. *Neural Networks*. Vol. 5, pp. 501-506.

Lorentz, G. G., 1966. *Approximation of Functions*. New York : Holt, Rinehart, and Winston.

Minsky, M. L. and Papert, S. A., 1969. *Perceptrons.* expanded ed., 1990. Cambridge, MA: MIT Press.

Nikolaev, N. Y., 2003. Learning Polynomial Feedforward Neural Networks by Genetic Programming and Backpropagation. *IEEE Transactions on Neural Networks.* Vol. 14, No.2, pp. 337-350.

Parker, D. B., 1985. *Learning logic*. Technical Report TR-47, Cambridge, MA: MIT Center for Research in Computational Economics and Management Science.

Pednault, E., 2004. *Transform Regression and the Kolmogorov Superposition Theorem,* IBM T. J. Watson Research Center, New York.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J., 1986. *Learning Internal Representations by Error Propagation.* Vol. 1 of Computational models of cognition and perception. Cambridge, MA: MIT Press. Chap. 8, pp. 319-362.

Steffensen, J. F., 1950. *Interpolation*. New York: Chelsea Publishing Company.
Blum, E. K. and Li, L. K., 1991. Approximation theory and feedforward networks. *Neural Networks.* Vol. 4(4), pp. 511–515.

Tikk, D., Kóczy, L. T. and Gedeon, T. D., 2001. *A survey on the universal approximation and its limits in soft computing techniques*. Research Working Paper RWP-IT-02-2001, School of Information Technology, Murdoch University, Perth, W.A. pp. 14.

Werbos, P. J., 1974. *Beyond regression: New tools for prediction and analysis in the behavioral sciences*. PhD thesis, Harvard University, Cambridge, MA.

Chapter 4

Bellman, R., 1961. *Adaptive Control Processes: A Guided Tour*. New Jersey: Princeton University Press.

Bishop, C. M., 1995. *Neural Networks for Pattern Recognition*, 1st edition. Oxford University Press. pp 333-338, 373-374.

Capanni, N. F., MacLeod, C., Maxwell, G., 2003. An Approach to Evolvable Neural Functionality. In *Proc. of ICANN/ICONIP Joint International Conference on Artificial Neural Networks and International Conference on Neural Information Processing*. 26-29 June 2003. Istanbul, Turkey. Vol. 2, pp. 220-223.

Ivakhnenko A.G., 1968. The Group Method of Data Handling – A rival of the Method of Stochastic Approximation. *Soviet Automatic Control*. Vol. 13, No. 3.

Ivakhnenko, A.G., 1971. Polynomial Theory of Complex Systems. *IEEE Transactions on Systems, Man, Cybernetics*. Vol. 1, No.4, pp. 364-378.

Levitan, I. B. and Kaczmarek, L. K., 2001. *The Neuron: Cell and Molecular Biology*, 2nd edition. Oxford University Press Inc, USA. Chapter 2, pp. 23-41.

MacLeod, C., McMinn, D., Reddipogu, A., and Capanni, N., 2002. Evolution by Devolved Action: Towards the Evolution of Systems. In Appendix B of McMinn, D., Using Evolutionary Artificial Neural Networks to Design Hierarchical Animat Nervous Systems, PhD thesis, The Robert Gordon University.

Thomas, G. B., Finney, R. L., 1996a. *Calculus and Analytic Geometry*, 9th edition. Addison Wesley Publishing Company. Chapter 8, pp. 663.

Thomas, G. B., Finney, R. L., 1996b. *Calculus and Analytic Geometry*, 9th edition. Addison Wesley Publishing Company. Chapter 8, pp. 672-673.

Thomas, G. B., Finney, R. L., 1996c. *Calculus and Analytic Geometry*, 9th edition. Addison Wesley Publishing Company. Chapter 8, pp. 672-673, 687.

Chapter 5

Barron, A. R., 1993. Universal approximation bounds for superpositions of a sigmoid function. *IEEE Transactions on Information Theory.* Vol. 39, pp. 930–945.

Barron, 2005. *pioneering and advancement of polynomial neural networks* [online] Available from: http://www.barron-associates.com, [Accessed 11 November 2005]

Bellman, R., 1961. *Adaptive Control Processes: A Guided Tour*. New Jersey: Princeton University Press.

Bishop, C. M., 1995a. *Neural Networks for Pattern Recognition*, 1st edition. Oxford University Press. pp. 9-14.

Bishop, C. M., 1995b. *Neural Networks for Pattern Recognition*, 1st edition. Oxford University Press. pp. 134.

Bishop, C. M., 1996. *Neural Networks: A Pattern Recognition Perspective.* Technical Report. NCRG. [online] Available from: http://www.ncrg.aston.ac.uk/ [Accessed 11 November 2005]

Capanni, N. F., MacLeod, C., Maxwell, G., 2003. An Approach to Evolvable Neural Functionality. In *Proc. of ICANN/ICONIP Joint International Conference on Artificial Neural Networks and International Conference on Neural Information Processing*. 26-29 June 2003. Istanbul, Turkey. Vol. 2, pp. 220-223.

Chen, M. S., 1991. *Analyses and Design of Multi-Layer Perceptron Using Polynomial Basis Functions.* PhD thesis, The University of Texas at Arlington.

Cotter, N. E., 1990. The Stone-Weierstrass theorem and its application to neural networks. *IEEE Transactions on Neural Networks.* Vol. 1, pp. 290-295.

Pao, Y. H., 1989. *Adaptive Pattern Recognition and Neural Networks*, 1st edition. Reading, MA: Addison-Wesley.

Crabbe, F., Dyer, M., 2001. *Goal Directed Adaptive Behaviour in Second-Order Neural Networks The MAXSON family of architectures*. Artificial Intelligence Lab, Computer Science Department, University of California, Los Angeles.

Duch, W. and Jankowski, N., 1999. Survey of Neural Output Functions. *Neural Computing Surveys.* Vol.2, pp. 163-212.

Durbin, R. and Rumelhart, D., 1989. Product Units: A Computationally Powerful and Biologically Plausible Extension to Backpropagation Networks. *Neural Computation*. Vol. 1, pp. 133–142.

Engelbrecht, A. P. and Ismail, A., 1999. Training product unit neural networks. *Stability and Control: Theory and Applications.* Vol. 2, No. 1-2, pp. 59–74.

Giles, C. L., Maxwell, T., 1987. Learning, Invariance, and Generalization in High Order Neural Networks. *Applied Optics.* Vol. 26, No. 23, pp. 4972.

Ivakhnenko A.G., 1968. The Group Method of Data Handling – A rival of the Method of Stochastic Approximation. *Soviet Automatic Control*. Vol. 13, No. 3.

Ivakhnenko, A.G., 1971. Polynomial Theory of Complex Systems. *IEEE Transactions on Systems, Man, Cybernetics*. Vol. 1, No.4, pp. 364-378.

Milenkovic, S., Obradovic, Z. and Litovski, V., 1996. *Annealing Based Dynamic Learning in Second-Order Neural Networks*, Technical Report, Department of Electrical Engineering, University of Nis, Yugoslavia.

Minsky, M. L. and Papert, S. A., 1969. *Perceptrons.* expanded ed., 1990. Cambridge, MA: MIT Press.

Nikolaev, N Y., 2003. Learning Polynomial Feedforward Neural Networks by Genetic Programming and Backpropagation. *IEEE Transactions on Neural Networks*. Vol. 14 No. 2, pp.337-350

Qian, S., Lee, Y. C., Jones, R. D., Barnes, C. W., and Lee, K., 1990. Function approximation with an orthogonal basis net. In *Proc. of IJCNN International Joint Conference on Neural Networks.* Vol. 3, pp. 605–619.

Rumelhart, D. E., McClelland, J. L., et al., 1986. *Parallel distributed processing : explorations in the microstructure of cognition*. Cambridge, MA. MIT Press.

Widrow, B. and Hoff, M., 1960. Adaptive switching circuits. In Proc. *1960 IRE WESCON Convention Record.* Vol. 4, pp. 96 - 104.

Chapter 6

Alberts, B., Bray, D., Lewis, J., Raff, M., Roberts, K., Watson, J. D., 1994a. *Molecular Biology of the Cell*, 3rd edition. New York: Garland Publishing Inc. pp 24–25,. 111-135, 721-782.

Alberts, B., et al, 1994b. *Molecular Biology of the Cell*, 3rd edition. New York: Garland Publishing Inc. pp 24.

Alberts, B., et al, 1994c. *Molecular Biology of the Cell*, 3rd edition. New York: Garland Publishing Inc. pp 25.

Alberts, B., et al, 1994d. *Molecular Biology of the Cell*, 3rd edition. New York: Garland Publishing Inc. pp 128-135, 211, 564-565.

Alberts, B., et al, 1994e. *Molecular Biology of the Cell*, 3rd edition. New York: Garland Publishing Inc. pp 557-558.

Almogy, G., Stone, L., Ben-Tal, N., 2001. Multi-Stage Regulation, a Key to Reliable Adaptive Biochemical Pathways, *Biophysical Journal*, Vol. 81, pp. 3016–3028.

Capanni, N.F., Macleod, C., Maxwell, G., Clayton, W., 2005, Artificial BioChemical Networks, CIMCA-IAWTIC, *Joint International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies & Internet Commerce*, 28-30 November 2005. Vienna, Austria. In Proc. IEEE special issue, Vol. 2, pp 98-102.

Chau, H. F., Yan, K. K., Wan, K. Y., and Siu, L. W. 1998. Classifying rational densities using two one-dimensional cellular automata. *The American Physical Society, Physics Review*. Vol. 57, pp. 1367–1369.

Chua, L. O., and Yang, L., 1988. Cellular neural networks: Theory and Applications. *IEEE Trans. Circuits and Systems.* Vol. 35, pp. 1257-1290.

Chua, L. O., Hasler, M., Moschytz, G. S., Neirynck, J., 1995. Autonomous cellular neural networks: A unified paradigm for pattern formation and active wave propagation. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*. Vol. 42, No. 10, pp. 559-577.

Curtis, H., 1968. *The Marvelous Animals: An Introduction to the Protozoa.* Garden City, N.Y., Published for the American Museum of Natural History [by] the Natural History Press.

Dogaru, R., 2003. Universality and Emergent Computation in Cellular Neural Networks. *Cellular paradigms theory and simulation*, chapter 2, World Scientific.

Elowitz, M. B., and Leibler, S., 2000. A synthetic oscillatory network of transcriptional regulators. *Letters, Nature*. Vol. 403, pp. 335-338.

Gerstner, W. and Van Hemmen, J. L., 1994. How to describe neural activity - spikes, rates, or assemblies? *Advances in Neural Information Processing Systems 6.* San Francisco, CA: Morgan Kaufmann. pp. 463-470.

Gerstner, W., 1995. Time structure of the activity in neural network models. *Physics Review.* Vol. 51, pp. 738-758.

Gerstner, W. and Kistler, W. M., 2002a. *Spiking Neuron Models. Single Neurons, Populations, Plasticity.* Cambridge University Press. Chapter 2.2.

Gerstner, W. and Kistler, W. M., 2002b. *Spiking Neuron Models. Single Neurons, Populations, Plasticity.* Cambridge University Press. Chapter 2.2.1.

Gerstner, W. and Kistler, W. M., 2002c. *Spiking Neuron Models. Single Neurons, Populations, Plasticity.* Cambridge University Press. Chapter 2.2.2.1.

Gontar, V., 2004. The dynamics of living and thinking systems, biological networks, and the laws of physics. *Discrete Dynamics in Nature and Society.* Vol. 1, pp. 101–111.

Gurney, K., 1997. *An Introduction to Neural Networks.* 1st edition, UCL Press. pp. 116.

Hameroff, S., Kaszniak, A. and Scott, A., 1998. *Toward a Science of Consciousness II: The 1996 Tucson Discussions and Debates*. Cambridge, MA: MIT Press. pp.421-437.

Hodgkin, A. L. and Huxley, A. F., 1952. A Quantitative Description of Membrane Current and its Application to Conduction and Excitation in Nerve. *Journal of Physiology*. Vol. 117 pp 500-544.

Hodgson, B.J., Taylor, C.N., Ushio, M., Leigh, J.R., Kalganova, T., Baganz, F., 2004. Intelligent modelling of bioprocesses: a comparison of structured and unstructured approaches. *Bioprocess Biosystems Engineering*. Vol. 26, pp. 353-359.

Khiel, T. R., and Bonissone, P. P., 2003. Evolving Artificial Biochemical Reaction Networks First Steps. In *Proc. International Conference on Systems Biology.* St Louis MO, November 2003.

Maass, W., 1997. Networks of Spiking Neurons : The Third Generation of Neural Network Models. *Neural Networks.* Vol. 10, pp. 1659-1671.

MacLeod, C, 2004. *Technical notes on spiking neurons*, Technical report. The Robert Gordon University.

MacLeod, C. and Maxwell, G., 1999. Intelligent Signal Processing. *Electronics World.* Vol. 105, No. 1764, December 1999, pp. 978-981.

MacLeod, C. and Maxwell, G., 2003. Practical Neural Networks, part 4: Applications and large Neural Nets, *Elektor Electronics.* Vol. 29, No. 320, April 2003, pp. 28-31.

MacLeod, C., McMinn, D., Reddipogu, A., and Capanni, N., 2002. Evolution by Devolved Action: Towards the Evolution of Systems. In Appendix B of McMinn, D., Using Evolutionary Artificial Neural Networks to Design Hierarchical Animat Nervous Systems, PhD thesis, The Robert Gordon University.

McCulloch, W. S. and Pitts, W. H., 1943. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics.* Vol. 5, pp. 115-133.

Sleigh, M. A., 1989. *Protozoa and Other Protists*. 1st edition, London: Edward Arnold.

Stein, R. B., 1967. Some models of neuronal variability. *Biophysics. Journal.* Vol. 7, pp. 37-68.

Thattai, M. and van Oudenaarden, A., 2002. Attenuation of Noise in Ultrasensitive Signaling Cascades, *Biophysics Journal*. Vol. 82, No. 6, June 2002, pp. 2943-2950.

Vandenbunder, B., 2001. Genomics in the understanding of the mechanisms of transcriptional regulation, *Bulletin du cancer.* Vol. 88, No. 3, pp. 253-60. Translated from French.

Von Neumann, J. and Burks, A. ed., 1966. *Theory of Self-Reproduction Automata*, University of Illinois Press, 1997 Translated from French.

Vreeken, J., 2003. *Spiking neural networks, an introduction*, Technical report. Institute of Information and Computing Sciences, Utrecht University.

Chapter 7

McMinn, D., Maxwell, G. and MacLeod, C., 2002. Evolutionary Artificial Neural Networks for Quadruped Locomotion. In *Proc. of ICANN the International Conference on Neural Networks.* 27-30 August 2002. Madrid, Spain, pp. 789 – 794.

Muthuraman, S., 2005. *The Evolution of Modular Artificial Neural Networks*, PhD Thesis, The Robert Gordon University, 2004.

Palmer, A.R., Shackleton, T.M. and McAlpine D., 2002. Neural mechanisms of binaural hearing. *Tutorial, Acoustic Science & Technology*. Vol. 23, No. 2.

Chapter 8

Capanni, N. F., MacLeod, C., Maxwell, G., 2003. An Approach to Evolvable Neural Functionality. In *Proc. of ICANN/ICONIP Joint International Conference on Artificial Neural Networks and International Conference on Neural Information Processing*. 26-29 June 2003. Istanbul, Turkey. Vol. 2, pp. 220-223.

McMinn, D., Maxwell, G. and MacLeod, C., 2002. Evolutionary Artificial Neural Networks for Quadruped Locomotion. In *Proc. of ICANN the International Conference on Neural Networks.* 27-30 August 2002. Madrid, Spain, pp. 789 – 794.

Minsky, M., 1961. Steps toward artificial intelligence. *Proceedings Institute of Radio Engineers.* Vol. 49, pp. 8–30.

Minsky, M. L. and Papert, S. A., 1969. *Perceptrons.* expanded ed., 1990. Cambridge, MA: MIT Press.

Muthuraman, S., Maxwell, G. and MacLeod, C., 2003. The Evolution of Modular Artificial Neural Networks for Legged Robot Control. *Artificial Neural Networks and Neural Information Processing.* Berlin: Springer. pp. 488-495.

Muthuraman, S., 2005. *The Evolution of Modular Artificial Neural Networks.* PhD thesis, The Robert Gordon University.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J., 1986. *Learning Internal Representations by Error Propagation*, Vol. 1 of Computational models of cognition and perception. Cambridge, MA: MIT Press. Chapter 8, pp. 319-362.

Samuel, A., 1957. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*. Vol. 3, pp. 210–229.

Chapter 10

Capanni, N. F., MacLeod, C., Maxwell, G., 2003. An Approach to Evolvable Neural Functionality. In *Proc. of ICANN/ICONIP Joint International Conference on Artificial Neural Networks and International Conference on Neural Information Processing*. 26-29 June 2003. Istanbul, Turkey. Vol. 2, pp. 220-223.

Capanni, N.F., Macleod, C., Maxwell, G., Clayton, W., 2005, Artificial BioChemical Networks, CIMCA-IAWTIC, *Joint International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies & Internet Commerce*, 28-30 November 2005. Vienna, Austria. In Proc. IEEE special issue, Vol. 2, pp 98-102.

MacLeod, C., McMinn, D., Reddipogu, A., and Capanni, N., 2002. Evolution by Devolved Action: Towards the Evolution of Systems. In Appendix B of McMinn, D., *Using Evolutionary Artificial Neural Networks to Design Hierarchical Animat Nervous Systems*, -PhD thesis, The Robert Gordon University.

McMinn, D., 2002. *Using Evolutionary Artificial Neural Networks to Design Hierarchical Animat Nervous Systems.* PhD thesis, The Robert Gordon University.

Minsky, M. L. and Papert, S. A., 1969. *Perceptrons.* expanded ed., 1990. Cambridge, MA: MIT Press.

Muthuraman, S., 2005. *The Evolution of Modular Artificial Neural Networks.* PhD thesis, The Robert Gordon University.

Appendices

Bishop, C. M., 1995. *Neural Networks for Pattern Recognition*, 1st edition. Oxford University Press. pp 13-14.

Capanni, N. F., MacLeod, C., Maxwell, G., 2003. An Approach to Evolvable Neural Functionality. In *Proc. of ICANN/ICONIP Joint International Conference on Artificial Neural Networks and International Conference on Neural Information Processing*. 26-29 June 2003. Istanbul, Turkey. Vol. 2, pp. 220-223.

Chang, C. and Cheung, J. Y., 1992. Backpropagation algorithm for higher order neural network. In *Proc. International Joint Conference Neural Networks*. Baltimore, MD. pp. 164–166.

Duch, W., and Jankowski, N., 1999. Survey of Neural Transfer Functions. *Neural Computing Surveys*. Vol. 2, pp. 163-212.

Giles, C.L., Maxwell, T., 1987. Learning, Invariance, and Generalization in High Order Neural Networks, *Applied Optics*, Vol. 26, No. 23, pp. 4972.

Haykin, S., 1999. *Neural Networks, A Comprehensive Foundation*, 2nd edition. Prentice Hall. pp. 139.

Kim, D. W. and Park, G. T., 2003. *A Design of EA-based Self-Organizing Polynomial Neural Networks using Evolutionary Algorithm for Nonlinear System Modeling*, Technical report. Department of Electrical Engineering, Korea University.

Minsky, M., 1961. Steps toward artificial intelligence. *Proceedings Institute of Radio Engineers.* Vol. 49, pp. 8–30.

Nikolaev, N. Y., 2003. Learning Polynomial Feedforward Neural Networks by Genetic Programming and Backpropagation. *IEEE Transactions on Neural Networks*. Vol. 14, No. 2, pp. 337-350.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J., 1986. *Learning Internal Representations by Error Propagation*, Vol. 1 of Computational models of cognition and perception. Cambridge, MA: MIT Press. Chapter 8, pp. 319-362.

Appendix A
Papers Produced During the Research

A.1 Introduction to the Appendix

The following contains two papers:

Capanni, N.F., Macleod, C., Maxwell, G., 2003, An Approach to Evolvable Neural Functionality, ICANN-ICONIP, Joint International Conference on Artificial Neural Networks and International Conference on Neural Information Processing, Istanbul, Turkey, Proc. supplementary volume for short papers, pp 220-223.

Capanni, N.F., Macleod, C., Maxwell, G., Clayton, W., 2005, Artificial BioChemical Networks, CIMCA-IAWTIC, Joint International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies & Internet Commerce, Vienna, Austria, Proc. IEEE special issue, vol. 2, pp 98-102.

An Approach to Evolvable Neural Functionality

Niccolo Capanni, Christopher MacLeod, Grant Maxwell
The Robert Gordon University, Aberdeen AB10 1FR, U.K.

Abstract. This paper outlines a neural model, which has been designed to be flexible enough to assume most mathematical functions. This is particularly useful in evolutionary networks as it allows the network complexity to increase without adding neurons. Theory and results are presented, showing the development of both time series and non-time dependent applications.

Introduction

Research into Artificial Neural Networks (ANNs) has resulted in a diverse range of neuron models that have improved network functionality and expanded applications. Improvements in training methodologies have further increased the possibilities and this has spurred extensive work on the intricacies of improving training time and network robustness.

Examples of the resulting innovative neuron models include Radial Basis, Leaky Integration, Non-linear and Spiking types [1]. Despite this, most widely used ANNs operate on a variation of the classical neural (Continuous Perceptron) model.

Our research team has produced a new neural model based on the idea that a neural unit should be flexible enough to fulfil any differentiable mathematical function required of it [2]. This model is a logical extension of the Perceptron and is particularly useful in evolutionary and control applications.

Basic Power Series Neuron

The most common artificial neural models in current use are those developed from the original McCulloch-Pitts neuron. Ignoring the squashing or activation function, which normalises the output, the activity of this neuron is given by:

$$O = \sum_{i=1}^{n} x_i w_i$$

Where n is the number of inputs, x_i is an input and w_i is the corresponding weight.

For a two input neuron, with input x associated with weight b and input y associated with weight c (as illustrated in figure 1), the activity could be written as:

$$O = bx + cy$$

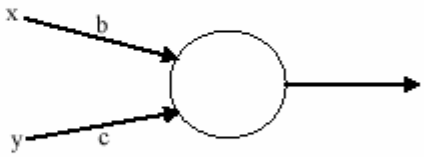


Figure 1. A simple neuron

This, of course, corresponds to a linear separator [3].

We can model any continuous function using an infinite Power Series [4] (for example a Taylor series):

$$f(x) = \alpha x + \beta x^2 + ... + \delta_n x^{n-1}$$

This is the basic series, which is given in most references. However, it can be extended to any number of variables (and hence any number of dimensions). For example, in two dimensions, the series is:

$$O = b_1 x + c_1 y + b_2 x^2 + c_2 y^2 + b_3 x^3 + ..$$

Notice that the first two terms are the same as in the first equation. This could correspond to a two-input neuron with inputs x, and y and weights b_n, c_n. A three input version of this neuron is shown in figure 2.

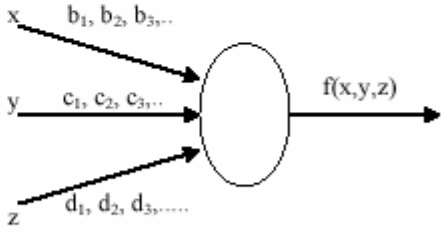


Figure 2. A polynomial neuron

More generally, for γ variables and an χ order series:

$$f(\overline{\sigma}) = \sum_{n=1}^{\chi} \sum_{m=1}^{\gamma} \alpha_{n,m} \sigma_m^{n-1}$$

Which is a non-linear separator. This can also be expanded to include input product terms (e.g. **wxy**) [5], which can enclose areas as discussed below.

Separators in the Second Order Case

To illustrate some of the attributes of higher order separators, let us first consider the second order case of a two input neuron with inputs labelled i_n and weights ω_n.

$$S = \omega_0.i_0 + \omega_1.i_1 + [\omega_2.i_0^2 + \omega_3.i_1^2]$$

Figure 3 shows how a single second order neuron can separate areas requiring many linear separators.

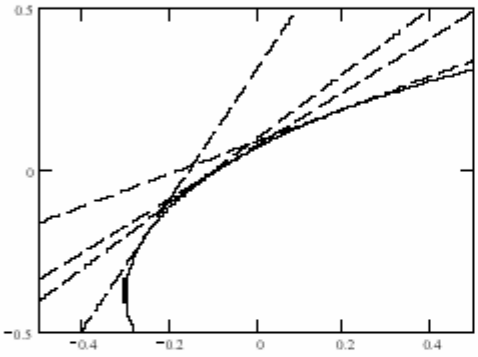


Figure 3. Second order separator

Figure 4 shows in a three-dimensional plot how the neuron can form a separator which can enclose (or exclude) a particular area.

Note that to reduce sensitivity in the neuron, it is often necessary to divide the higher power terms by their factorial.

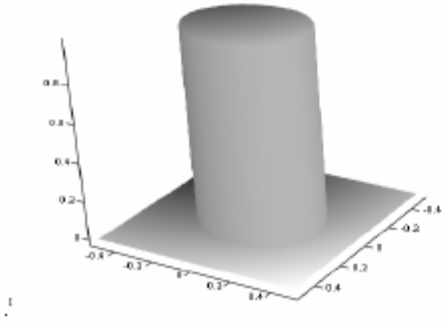


Figure 4. More complex second order separator

In all of these models, the weights of the second powers can evolve to "0", and the Perceptron emerges.

Separators in the Third Order Case

The decision surface can be further complicated through expansion of the power series. The symmetrical limitation of the 2nd order case can be removed by adding a 3rd order as shown in figure 5.

$$S = \omega_0.i_0 + \omega_1.i_1 + \omega_2.i_0^2 + \omega_3.i_1^2 + [\omega_4.i_0^3 + \omega_5.i_1^3]$$

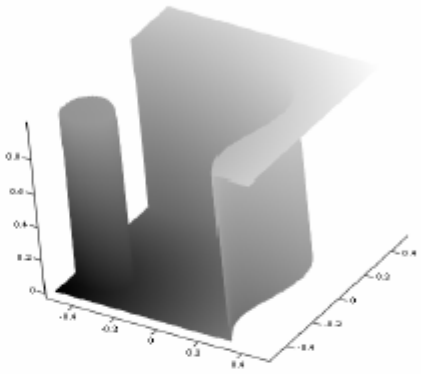


Figure 5. 3D Decision surface of 3rd order neuron

Addition of higher powers increases the complexity of the decision surface and allows greater separation and isolation of decisions.

More Complex Cases

The addition of higher orders of the power series is not the only method of improving the unit functionality. The previous neurons had no interaction between the inputs. However, if interaction is allowed, even more complex behaviour exists (as in sigma-pi units).

$$S = \omega_0.i_0 + \omega_1.i_1 + \omega_2.i_0^2 + \omega_3.i_1^2 + [\omega_4.(i_0^2.i_1) + \omega_5.(i_0.i_1^2) + \omega_6.(i_0.i_1)^2]$$

A 2nd order expansion is shown, with the complex expansions enclosed within brackets.

The addition of these product terms allows greater flexibility in the decision surface without continually expanding the power series. It also introduces a greater element of asymmetrical separation and isolation of distinct regions through the interaction of the inputs. A 3rd order complex neuron produces a separator as shown in figure 6.

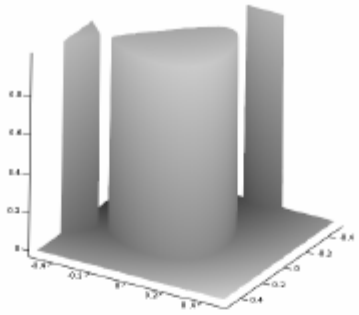


Figure 6. 3D decision surface 3rd order incomplete complex neuron

Applications

Allowing neurons to use higher powers of their inputs allows smooth separators as shown in figure 3. This improves generalisation in the network, although for higher orders generalisation decreases again [5].

These neurons are particularly useful in control systems in a similar way to radial basis units as they can take complex continuous forms without the need for large networks.

In evolutionary networks they allow the complexity of the network to increase by adding extra orders without adding new units to the network structure.

Finally, they can be used with Taguchi Method training [6] by training the first order initially and adding and training each additional order thereafter for a more accurate response.

Some Typical Results

When used with evolutionary training methods, the neurons allow us to reduce the number of epochs required to reach a solution as shown in figure 7.

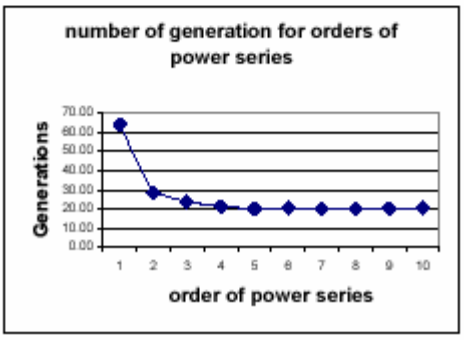


Figure 7. Reduction of training epochs

This network is three layered, consisting of 25 inputs and 60 neurons configured for character recognition. One can see that there is little point in introducing orders above the third. Although the training epochs decrease, the computational power required for training increases - in the case of the three order neuron, by three times. However, there is still a net improvement in training time.

As mentioned above we can also train one order of the neuron at a time using an evolutionary algorithm.

When used in a standard pattern recognition system (of the same type mentioned in relation to figure 7), the use of the higher order neurons allows the system to operate with fewer units, as shown in figure 8.

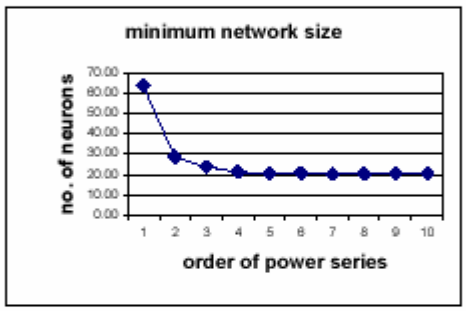


Figure 8. Number of units required in character recognition system

One can see in both cases, that above the fifth order, performance shows little improvement. Indeed there may be disadvantages in using too many orders [5]. In this case, the reduction in number of neurons offsets the increase in multiply and accumulate instructions required for a more complex network.

Future Work - Time Series Models

A time response can also be modelled with another Power Series

$$f(t) = a + bt + ct^2 + ...etc$$

Where t is the time variable. There is no evidence that the temporal properties of neurons are this complex. As a result, it is often easier to simply make an evolvable (or trainable) time decay.

$$f(t) = ae^{bt}$$

A squashing function may be applied to this if necessary, as can another time series representing a delay (refractory period) of the neuron's output.

This response can be multiplied with the power series described earlier to provide a neuron that is capable of mimicking any differentiable function of time.

Conclusions

The neuron described above may prove useful in several application areas including control systems and evolutionary systems. Its principal asset is that it allows the network to evolve with a wide variety of behaviours from a small number of neurons.

References

1. Arbib, M.: The Handbook of Brain Theory and Neural Networks, The MIT Press (1998)
2. MacLeod, C., McMinn, D., et al.: Evolution by devolved action: towards the evolution of systems. In appendix B of: McMinn, D.: Using Evolutionary Artificial Neural Networks to Design Hierarchical Animat Nervous Systems, PhD Thesis, The Robert Gordon University, Aberdeen, UK (2002)
3. Khanna, T.: Foundations of Neural Networks, Addison Wesley (1990)
4. Croft, A., Davison, R., Hargreaves, M.: Engineering Mathematics, Addison-Wesley (1996) 418-440
5. Bishop, C. M.: Neural Networks for Pattern Recognition, Oxford (1995) 9-14.
6. MacLeod, C., Maxwell, G. M.: Using Taguchi Methods to Train Artificial Neural Networks," AI Review, **13,** 3, Kluwer (1999) 177-184

Artificial BioChemical Networks

[1,3]Niccolo Capanni, [2,3]Christopher MacLeod, [2,3]Grant Maxwell, [4]William Clayton
[1]*The Robert Gordon University, School of Computing, St Andrew Street, AB25 1HG*
[2]*The Robert Gordon University, School of Engineering, Schoolhill, AB10 1FR*
[3]*Aberdeen, Scotland UK*
[4]*Olin University, Olin Way, Needham, MA 02492-1200, USA*
E-mail nc@comp.rgu.ac.uk; chris.macleod@rgu.ac.uk; grant.maxwell@rgu.ac.uk; William.Clayton@students.olin.edu

Abstract

Connectionist approaches to Artificial Intelligence are almost always based on Artificial Neural Networks. However, there is another route towards Parallel Distributed Processing, taking as its inspiration the intelligence displayed by single celled creatures called Protoctists (Protists). This is based on networks of interacting proteins. Such networks may be used in Pattern Recognition and Control tasks and are more flexible than most neuron models. In this paper they are demonstrated in Image Recognition applications and in Legged Robot control. They are trained using a Genetic Algorithm and Back Propagation.

1. Introduction

Protoctists, also called Protists, are singled celled organisms which live in a variety of different environments [1]. Those which display animal-like behaviour are usually called Protozoa and make up a large part of the fauna often disparagingly known as "pond life".

However, despite their primitive reputation, they display remarkable abilities and behaviours [2]. Some have stinging darts with which they disable their prey; others have sensory hairs to feel their way about and sense the vibration of prey approaching and a few even have leg-like appendages for locomotion. They can avoid light with their sensitive eyespots and actively hunt for their food. The variety they display is enormous, with a range of relative sizes greater than that between a rabbit and a blue whale. Some even build shelters - shells with which to protect themselves from predators and the environment. They display many of the traits of intelligence.

2. Natural Biochemical Networks

Protozoa display the behaviours described above by means of interactions between proteins in their cytoplasm. Proteins are the chemical workhorses of the cell [2]. It is the cell proteins which the DNA genetic code specifies, as shown in Figure 1. This scheme is so fundamental that it is sometimes referred to as the "Central Dogma" of biology.

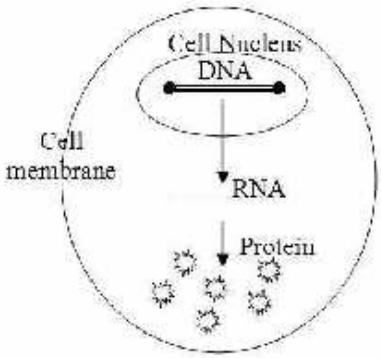


Figure 1. Central Dogma

Proteins perform all the important operations of the cell - making new material, destroying old and sensing and signalling changes in the cell's environment. All proteins bind to other chemicals. Some synthesise new molecules by joining bound component parts together, others break them up - such proteins are called Enzymes. Yet others use their ability to bind by joining to other proteins, changing their behaviour and thereby forming signalling networks within the cell [2]. Such a

signalling network is best illustrated by example - see Figure 2.

Figure 2. A simplified signalling pathway

Figure 2 is a hypothetical example of a signalling network. Molecules in the cell's external environment A bind to receptor proteins B. This changes the shape of the receptor and causes a protein C, which was bound to the receptor to disassociate from it. This protein then floats freely in the cell's cytoplasm and eventually binds with the protein D (chemicals in the cytoplasm are buffeted around by thermo-dynamic forces which act to mix the constituents). When C and D are bound as shown in E, they can bind further to a motor protein F (a protein which can change its shape by a large amount, allowing it to move large objects). The motor protein is attached to the cell's outer membrane and this causes the cell to move towards or away from the molecules A by changing its shape.

Obviously such a system may be represented by a network in a similar way to a Neural Net [2]. In this case, the nodes would represent the proteins, the connections their interactions and layers represent sequential/hierarchical protein interaction. One can also see that the system allows for intricate control over these functions - for example, by using other proteins generated as a result of other internal or external cellular stimuli, which can stimulate or suppress those shown [4]. An appropriate name for such a network might be an Artificial Biochemical Network (ABN).

3. Artificial Biochemical Networks

Given that in the simplest implementation, the basic network topology can be constructed to be no different in appearance from other connectionist networks, the difference is mainly in the unit functionality and information flow. A typical output is shown in Figure 3. The lag time until the presence of the protein is felt is A; this is set using the Genetic Algorithm which can also train the network weights. Time B is proportional to unit activity, the constant of proportionality being defined by the Genetic Algorithm. Unit activity is calculated using a standard Leaky Integrator [5].

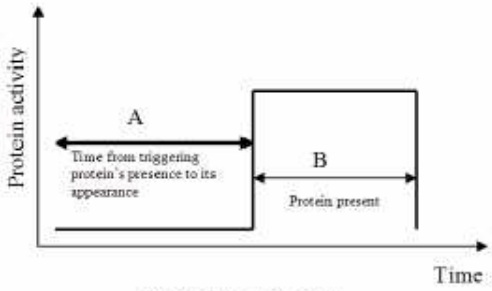


Figure 3. Unit cycle

The Genetic Algorithm has also been implemented to choose which of the time periods A or B is proportional (or inversely proportional) to the unit activity and which is fixed [6]. This additional evolvable parameter [7] has lead to pulse width or frequency modulated units as shown in Figure 4.

This allows for the production of more universal units from this basic type. It has been suggested that such dynamics may lead to new perspectives on intelligence [8].

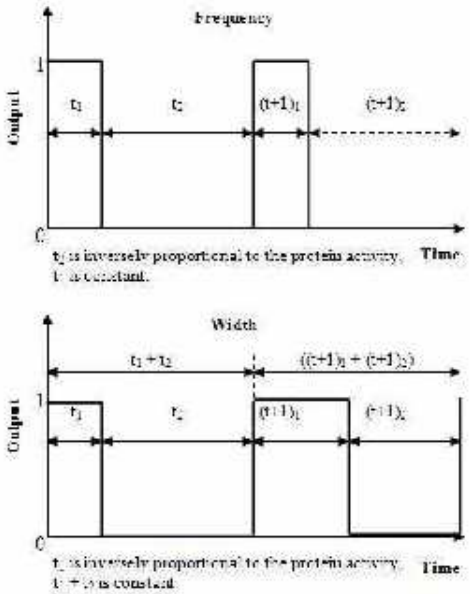


Figure 4. Pulse modulated units

4. Examples in Pattern Recognition

A network based on the units described above (in this case the Pulse Width Modulated variety) was compared with a standard Multilayer Perceptron (MLP) in pattern recognition problems. A 5 by 5 pixel grid was set up with standard roman characters and alternative identifiers as shown in Figure 5.

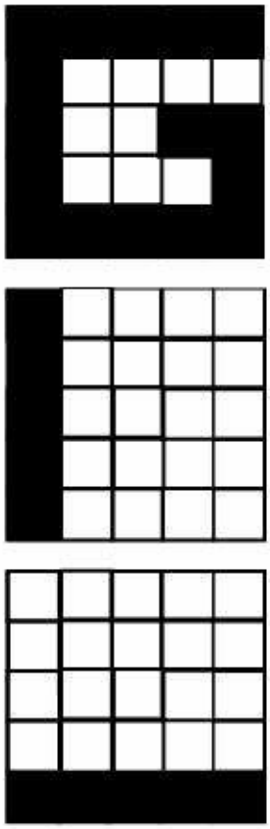


Figure 5. A 5 by 5 grid of letter "G", simple predator and prey identifiers

It was tested first whether the network had the same memory capacity as an equivalent MLP network (one with the same number of units). The networks used had 25 inputs (corresponding to the input pattern pixels) and the same number of output units as patterns. The number of hidden layer units was then increased and the network trained, using a [200, 200] Genetic Algorithm, with one pattern at a time (starting with character .A.) until failure. The results are shown in Figure 6. The solid line shows the MLP and quantised ABN results, the dashed line the initial (un-quantised) ABN.

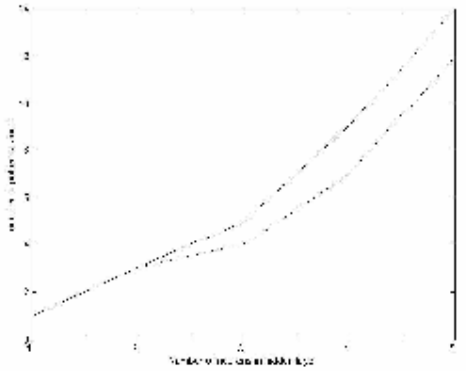


Figure 6. Memory performance of ABN vs. MLP

It may be seen from the figure that the two networks hold a similar number of patterns slight differences at first attributed to different initial values used in the training algorithm were found to be characteristics of the time-domain quantisation of ABNs.

Next the systems were tested to establish their generalisation abilities. Noise was progressively added to the data and the performance measured. Figure 7 shows the results of this (lines are represented as previously). The noise addition procedure is that used in the MATLAB Neural Networks toolbox [9].

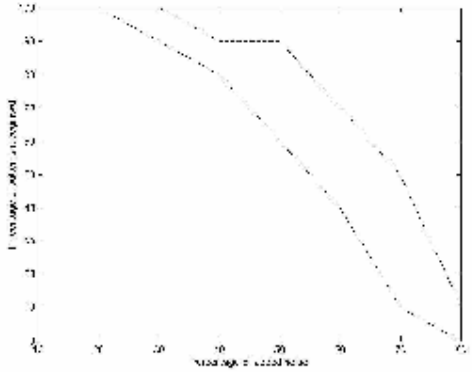


Figure 7. Noise tolerance of ABN vs. MLP

Again, it may be seen that the networks are comparable in performance. Similar results were also obtained for the Frequency Modulated and full versions of the network. The ABN showed better generalisation and an investigation on this is reported [6].

It was also shown that the network could be trained using standard Back Propagation. In these cases, the

scalar inputs (in the case of the characters used above, a black pixel was a 1 and a white 0) where coded as pulses using a sigmoidal transfer function for normalisation as shown in Figure 8. The outputs were similarly normalised.

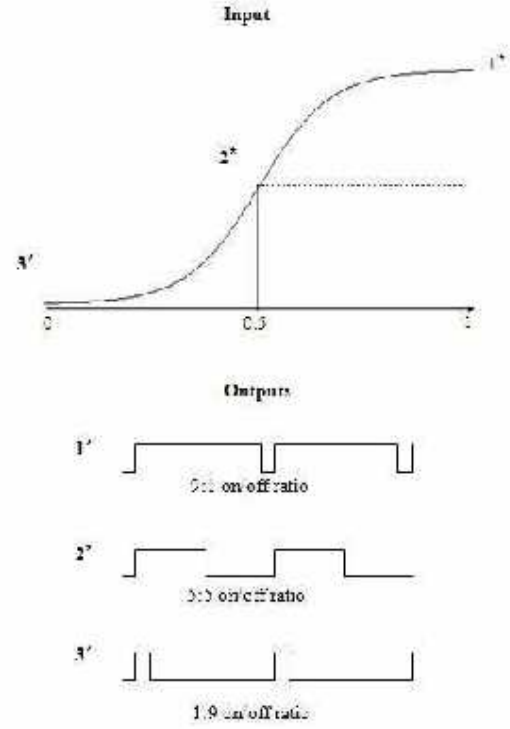


Figure 8. Transfer function

A method of Back Propagation was synthesised for the time domain specifics of the ABN. This resulted in an improved training time over standard Back Propagation from the same initial parameters but as expected no functional improvement in memory or generalisation [6].

5. Examples in control

The results above show that, in pattern recognition problems, the network is similar in performance to a standard MLP type network. There is scarcely any advantage in this in terms of time independent pattern recognition as it is more complex to program (having to keep internal clocks to account for where the units are in their cycles). There are functional advantages which are discussed [6] in the conclusions.

However, MLP networks have difficulty producing suitable outputs to control time domain tasks (for example PWM motor control). The ABN network is inherently time domain and does not have this disadvantage. It was therefore also trained to control the gaits of a simulated bipedal legged robot. The robot is based on a physical robot, the legs of which are controlled by servo motors as shown in Figure 9. The legs have one active and one passive degree of freedom. The network has four units; two chosen by the GA are designated outputs.

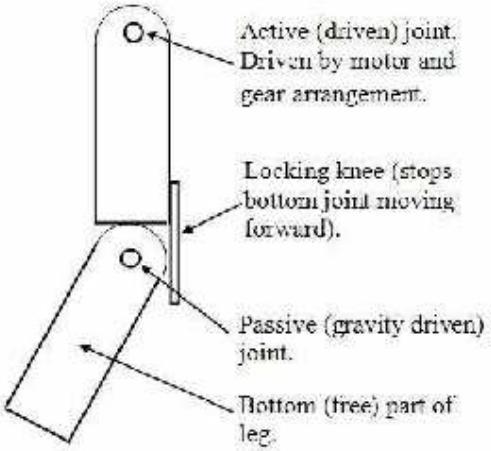


Figure 9. Robot leg layout

Space restrictions here stop us from exploring these dynamics in depth; however, this simulation has been used and reported many times previously and the dynamics of the legs and the robot are fully reported in other papers [10, 11]. Figure 10 shows the leg movements generated when the network was evolved to walk. The result corresponds well with the perfect pattern (a perfect pattern would have a repeat time of 60 time steps and a movement from position 80 to position 100).

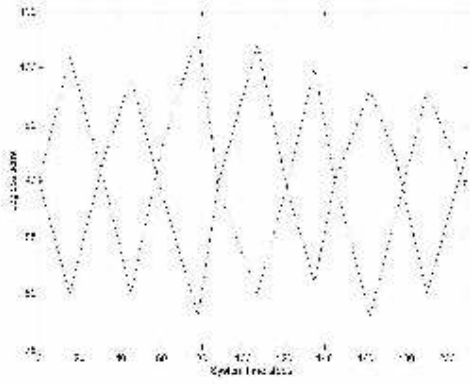


Figure 10. Movement pattern of legs

6. Conclusions

The system discussed in this paper is a new and different approach to connectionist AI. Instead of being based on neural networks, it models the chemical signalling within cells. Of course, such signalling lies at the root of neuron functionality also, as the neuron is itself a cell.

The retention of both generalisation and universality [12] affects the ABN performance in pattern and control, allowing for graceful decay as noise increases. Such "fuzzy" uncertainty is far more stable than a system that performs longer with higher accuracy then undergoes critical failure with little warning.

With regards to mobile robot operation there is a functional advantage of ABN pattern recognition. Most pattern recognition is achieved "in vitro" where time is not a constraining factor, here "snapshot" pattern recognition can be utilised. In an artificial organism that has to adapt to its environment "in vivo" then an ABN information flow pattern system can assimilate information as it appears.

The implementation of ABNs allows a single type of intelligent units to perform all the operations of a modular AI used in robot control and can be encoded as part of the evolutionary algorithm with an operator placing specific units.[6, 13] .

The approach has several advantages. It simplifies the design of time dependant outputs which, in turn, allows the straightforward implementation of Central Pattern Generator networks in robots, Pulse Width Modulation for Motor Control and other similar systems. However, the networks are equally at home in traditional Pattern Recognition tasks. They also allow systems to be developed which behave in many respects like Spiking Neuron models, but without the associated complexity. Finally, they may be trained using traditional methods.

References

[1] H. Curtis, *The marvellous animals: an introduction to the Protozoa*, Heinemann education, 1969.

[2] B. Alberts et al, *Molecular Biology of the Cell*, Garland Publishing Inc, New York, 1994 (3rd edition). pp24 - 25. 111 - 135. 721 - 782.

[3] A.R. Barron, R.L. Barron and E.J. Wegman, "Statistical learning networks: A unifying view", *in Computer Science and Statistics: Proceedings of the 20th Symposium on the Interface*, edited by E.J. Wegman, 192-203, 1992.

[4] C. MacLeod and G. Maxwell, "Evolutionary Electronics". *Practical Electronics*, August 1999.

[5] K. Gurney, *An Introduction to Neural Networks*, UCL Press, 1997.

[6] N.F. Capanni, *Evolvable Neural Functionality and Artificial BioChemical Networks*, PhD Thesis, The Robert Gordon University, 2006.

[7] C. MacLeod and G. Maxwell, "Practical Neural Networks, part 4: Applications and large Neural Nets", *Elektor Electronics*, vol 29, no 320, April 2003, 28-31.

[8] C. MacLeod and G. Maxwell, "Intelligent Signal Processing", *Electronics World*, vol 105, no 1764, December 1999. 978-981.

[9] H. Demuth and M. Beale, *MATLAB Neural Network toolbox manual*, The MATH WORKS inc, 11.40 -11.46, 1994.

[10] S. Muthuraman, G. Maxwell and C. MacLeod, "The Evolution of Modular Artificial Neural Networks for Legged Robot Control", *Artificial Neural Networks and Neural Information Processing*, Springer, Berlin, 2003, 488-495.

[11] D. McMinn, G. Maxwell and C. MacLeod, "Evolutionary Artificial Neural Networks for Quadruped Locomotion", *Proceedings of the International Conference on Neural Networks ICANN 2002*, Madrid, Spain, 2002, 789.794.

[12] N.F. Capanni, C. MacLeod, G. Maxwell, "An Approach to Evolvable Neural Functionality", *Proceedings of ICANN/ICONIP*, pp 220-223, 2003.

[13] S. Muthuraman, *The Evolution of Modular Artificial Neural Networks*, PhD Thesis, The Robert Gordon University, 2004.

Appendix B
Evolution and Devolved Action

B.1 Introduction to the Appendix

"Evolution and Devolved Action" examines the limitation of current Artificial Intelligence, concentrating on connectionist models such as Artificial Neural Networks which are created through Evolutionary Algorithms. The paper presents ideas on how these limitations may be overcome and was the initial information source for this research.

Evolution and Devolved Action: towards the evolution of systems.

C. MacLeod, D. McMinn, A. B. Reddipogu, N. F. Capanni, G. M. Maxwell.
School of Electronic and Electrical Engineering,
The Robert Gordon University, Aberdeen.

Introduction.

The Artificial Neural Networks group at the Robert Gordon University has, over the last six years, built up considerable knowledge and practical experience in Evolutionary Artificial Neural Networks. This experience started with the PhD project by C MacLeod [1] and continued with the work of D McMinn [2] which is due for submission in June 2001. These are being followed up by the work of research students A B Reddipogu and N F Capanni.

Initial work concentrated on Neural Networks that could grow to fulfil their function. C MacLeod, in his thesis on this topic, proposed a model of a robotic control system to be used as a vehicle for further research. This model formed the basis of D McMinn's project. The robotic control system [2, 3, 4] has a defined modular structure that enables the researcher to create networks for particular tasks and so allow the robot to function. McMinn has used this structure successfully as a basis to develop Evolutionary ANNs implementing Central Pattern Generators and Reflexes for robot locomotion.

It has become apparent, over the course of these projects, that a network that can evolve into a modular structure without the need for designed partitioning would be the next step forward for the group's research. This should allow the network to develop naturally and in an open-ended way without the need to artificially constrain it. Such an approach needs an evolutionary algorithm that can automatically and naturally evolve a "system": that is, a modular network rather than a fully interconnected homogenous structure. No acceptable Genetic or Evolutionary Techniques are currently available to do this. The group therefore needed to look to nature and discover the reasons why natural systems allowed such modularity to evolve and how it might be exploited in the course of future work.

It was felt that this work would be a culmination of the group's research to date, both furthering the work on robotic control systems and also including ideas from previous practical work, by MacLeod and McMinn, in terms of Evolutionary Networks and Incremental Evolution (Embryological Algorithms) [1]. This is because such a network will need to evolve and to "grow" from a simple to complex structure. The research and ideas that lay the foundations for this work are contained in this report. The report also considers two related aspects (neural learning and neural function), but as explained above, it is the layout topology or wiring which is felt to be the most important topic. After all, complex systems may be built up from simple elements, such as transistors or gates (or indeed, at the most basic level, atoms), providing that they are interconnected in the correct manner.

Therefore, to summarise: We are concerned with discovering an evolutionary method capable of evolving *systems*. To establish a method capable of this, we must look

again at the action of biological evolution and compare it with current artificial evolutionary methods.

PART A – A critical review of biological evolution.

A.1 Artificial Evolutionary Methods.

ANNs are usually directly coded into a GA [5], if such is to be used for topology evolution, each node or connection being part of the chromosome. However, the entire human genome does not contain enough space to directly code even a small part of an actual biological brain (let alone the rest of the body). So we must look again at the action of real genetics, as this must hold a clue to understanding the complexity of the biological system.

A.2 Biological genetic action.

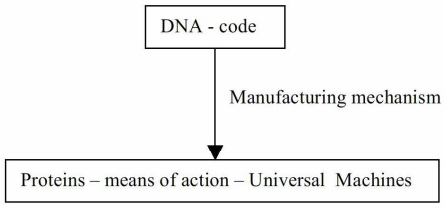
DNA is a code arranged in the form of a long string. Each position in the string has one of four possible values; these correspond to the four nucleotide bases. This code is transcribed by a rather complex machinery to amino acid. Three bases code for one amino acid (the three bases are known as a codon). This would allow a total of $4^3 = 64$ possible amino acids to be specified. However the code has redundancy built in (which reduces errors) and so only 20 amino acids are actually coded.

The proteins are made of these amino acids, joined together. Thus the DNA, by coding a string of amino acids, codes proteins (essentially, one protein = one gene). All that the genetic material does is code proteins and nothing else. The proteins are made by the amino acids joining together via peptide bonds (the joining is done automatically by the machinery which transcribes DNA to Protein). There is no defining line between a polypeptide and a protein. The genome of a simple organism such as e. coli can code about 4000 proteins. A human genome can code some 30,000 and 60,000 is considered the practical limit (due to problems with mutation) [6].

A.3 Proteins are the really clever part of genetics.

Every action in biology is mediated by proteins; they are the *Universal Machines* which make all biological processes possible. If other chemicals are required, they must be made with the aid of proteins. This vital point is summarised in figure 1.

Fig 1, Relationship between DNA and proteins.



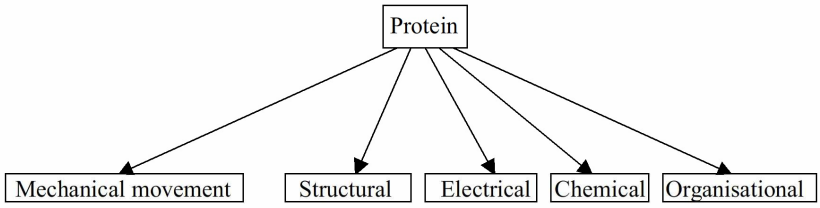
Because DNA cannot directly control the organism, we may say that the system exhibits a *Devolved Action*. All control proceeds via the proteins (the contrast with the artificial version is obvious).

Therefore, the system has two functional components. Firstly a code which can be mutated and exchanged through breeding. Secondly the machines (ie the proteins) which the code specifies (perhaps *part* is a better description than machine. This is because proteins can self assemble into more complex structures. However, it is worth remembering that there is much more to proteins than just self-assembly).

Examples of proteins:
Actin - Protein which operates mechanically to produce muscle contraction.
Haemoglobin – Carries oxygen supply around the body.
Enzymes – Catalyse chemical reactions.

So, these substances can perform almost any task. Figure 2 shows a tentative classification.

Fig 2, Proteins as Universal Machines.



It is astonishing that nature has one group of substances which can perform all these functions. The classification also shows up some of the limitations of organisms; there are some things which proteins cannot achieve.

One could, in fact, imagine a great map of all possible genotypes (DNA codes) to all possible phenotypes (resulting protein group structures, some of which would be organisms). Once such a process is understood, it opens up the possibility of "biological engines"- artificial biological structures engineered via DNA manipulation to become, for example, motors or other mechanisms. Imagine an engine running on grass in the same way a cow does. Such bio-engines might more resemble bagpipes than internal combustion engines.

All this throws up a problem for the designer of artificial systems: *The biological system is not directly coded as in the current Artificial Evolutionary Algorithms - what is coded are the universal machines which can assemble themselves or other parts into a system.* There is presently no way of synthesising such universal machines. Therefore, any system we design will be less adaptable than nature's machines (although it is arguable that a similar effect might be achieved by a code being supplied to a *Universal Manufacturing Machine*; such a system would not be able to interlock as discussed below). In the sections that follow we will consider how some of the most important of these machines operate.

Of course, it might be argued that even the process described above is not truly mimicking nature. After all, why not start with the most basic components, which occur naturally in the system - Electrons, Protons and Neutrons? These self assemble under the control of thermodynamic forces into molecules. One can argue, indeed,

from this point of view, that the current search in physics for a "theory of everything", is no more that the search for the initial rules governing the way fundamental automata assemble. However, it should be borne in mind that, apart from life, evidence of organised complex dynamical structures in nature is hard to find. Most structures form into "lumps" like familiar rocks. What complex structures do exist - crystal and dendritic mineral structures, for example, are the produce of simple automata rather than complex processes like those present in life. DNA and life can therefore be seen as having rather unique properties (perhaps because of their complexity). A point further underlined by the absence of life (or even evolution) based on any other chemical system, produced either in nature or the laboratory.

A.4 The importance of interlocking.

The proteins themselves can effect the DNA by binding to it and stopping it manufacturing other proteins. They can also effect each other and work together to create much more complexity in the system (see descriptions of signal peptides and adhesion molecules, below) [7]. Such behaviour may be termed *interlocking*, in that the different parts of the whole have evolved to function as a system.

Interlocking is a vital part of the organisational ability inherent in biological organisms and allows them to exhibit complexity which would be impossible through separately functional components.

Interlocking is puzzling because it is difficult to understand how one machine can evolve separately from another and still show inter-dependence on it. The explanation lies in the fact that proteins are all made from combinations of the same 20 amino acids. So eventually natural mutation will produce one protein which effects another in a beneficial way. In other words, the evolutionary search distance between proteins, which can effect each other, is not too great. This is dramatically illustrated in the genetic disease called sickle cell anaemia. In this disorder, there is a mutation in the coding for haemoglobin. This mutation causes the protein to form long thread-like structures within the cell and this causes problems with its function. However, it shows that a small mutation can change a protein from one machine (which carries oxygen) to another (a self assembled structural protein). So the crux of the matter is: *Because proteins are built from the same limited number of units, they can interact and relatively small variations can completely change their function* (and occasionally into something useful) [8].

A.5 The power of the universal machine – Protozoa.

Such is the power of this system that it can design 'intelligence' without the need for neural circuitry. For example, consider the Protozoa; these are single celled animals which display almost unbelievable complexity and apparent intelligence (defined loosely). All this is achieved by protein chemistry. They show that with a powerful organisational system, perhaps the individual components need not be complex 'processing units' such as neurons [9].

A.6 The perfect artificial evolutionary system is not achievable with current technology.

We can now see that the most general Artificial Evolutionary System would consist of a mutatable code, which specifies general universal machines (and the process which turns code into machine).

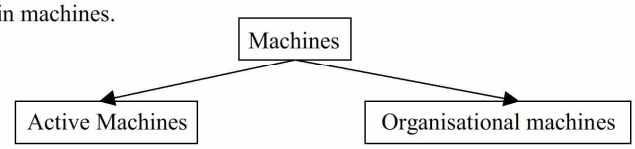
The general universal machine itself would be a machine capable of forming any mechanism or structure, capable of fully interlocking, with a small evolutionary distance between itself and others, and finally, capable of forming associations which can direct their own organisation.

If we apply selection (natural or artificial) to such a system, we could eventually evolve any mechanism. Although this is currently impossible, a method will be suggested in which a more restricted set will prove useful.

A.7 Active Machines and Organisational Machines.

First, let us make an important distinction between two types of protein. Those which form the active processing machinery and those which direct the construction of such, figure 3.

Fig 3, Protein machines.



If we can tackle the organisational part of the system, perhaps its power may be used (as demonstrated by the protozoa) to build complex structures without the need for a completely universal (active) machine.

A.8 Organisational proteins.

Presently, we do not understand all the mechanisms of organisation within an organism. However, what is known is powerful enough to solve many of the organisational problems we are concerned with. There are several different types of organisational protein identified. These operate in quite different ways [10].

One example, signal peptides, act like keys, allowing proteins which have the signal peptide chain attached to enter different organelles of the cell. Generally, once the protein has entered, the signal peptide (which forms only a short part of the total protein chain), is discarded. Proteins get distributed around the cell by the normal thermodynamic forces which act on all chemicals within the cell [11].

A.9 Cellular structure.

Why do all known, large organisms maintain a cellular structure? The answer to this question lies in the fact that the mechanism of chemical distribution in the cell is driven by thermodynamic forces as just described. The molecules are buffeted by other constituents of the cell and eventually, because of the small physical volume

involved, will find their way to the area where they are required (for example, be admitted by a gate to their signal peptide into another section of the cell)[12].

For billions of years organisms existed as single cells before multicellular structure developed. With the organisation of earth's biochemical system (which may be only one of many possible), multicellular organisation is necessary to achieve sizes greater than about 1mm.

A.10 Gradients.

The chemical signals which allowed signalling inside cells were adapted to allow signals to propagate from cell to cell. In fact, all multicellular organisms have extensive intercellular signalling built into their chemistry.

Probably the most important of these intercellular signals are *gradients*. If one cell emits a chemical signal, it reduces in concentration as it travels out from the cell which emitted it. Cells have a built-in threshold to these chemical gradients and can switch off or on the expression of proteins according to whether they are receiving enough of the signal or not (once a protein is switched on, it can maintain itself by positive feedback, that is it supports its own production. It can also block other proteins by binding to their DNA sites). This mechanism plays a pivotal role in the differentiation of cells in the foetus and in their eventual position in the body - and so the formation of a patterned organism. For example, cells migrate along gradients (so they control the eventual position of cells) and gradients can cause the cell to switch on the expression of proteins which might change it into (say) a muscle cell (so gradients control cell function). The symmetrical nature of gradients is the reason why animal bodies display radial or axial symmetry. It also explains the fractal nature of some biological systems (because, as each chemical gradient triggers, it produces divisions at successively smaller levels, so producing a self-similar pattern at successive dimensions) [13, 14, 15].

A.11 Neurons.

The neuron developed to allow long distance signalling. This is important because, for the reasons listed above, there are limits to intercellular signalling, particularly in terms of speed (and gradients like other chemical signals, which are not re-triggered in each cell, only operate over a distance of less than a couple of millimetres).

So Neurons developed with long cellular processes which can stretch over a much larger distance. Once they had this property, they could also develop a more specialised signalling apparatus. This is important if any organism is to react to its environment quickly (chemical signals are slow), with the obvious evolutionary advantage this would give the organism. In fact, neurons are one of the most ancient of all cells. They also demonstrate another type organisational protein, those which control cellular adhesion. These cause some cells to be able to adhere to each other and not to others. This appears, along with gradients and cellular migration, to be a major factor in neural organisation [16]. Neurons also use gradients to direct their long processes to their destinations (axon growth factors), something which, if we could harness it, could direct the wiring of three dimensional integrated circuits.

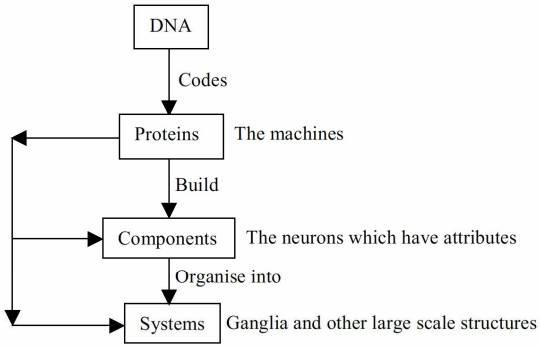
PART B – A New approach to ANNs.

B.1 An Evolutionary Artificial Neural Network.

Now, all this may have left the feeling that we know what goes into an evolutionary system but we couldn't replicate it because we can not replicate a universal machine (at least at present). However, while a full system is currently beyond our capabilities, we can identify, for a neuron, which parameters are important and so limit the case to something more manageable. However, to do this, we need a cunning scheme to replicate some of the diversity which nature provides.

We should recognise, firstly, that what the universal machines build are cells and what they give to these cells are a series of properties or attributes which aid self-organisation. For example, attributes include the ability to adhere to other cells or the ability to migrate along gradients. One way of looking at this, which helps us to simplify the problem, is the hierarchy shown in figure 4.

Fig 4, DNA and proteins as part of a systems structure.



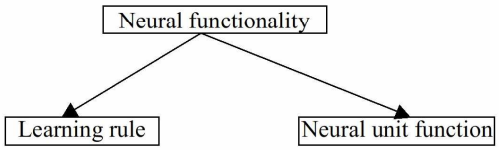
Obviously, it isn't quite as simple as this because proteins outside cells (in the cellular or developmental environment) help organise components and also play an important part in the functioning of the system. This is shown by the side arrows in the diagram. However, this idea does give us more to work with than the most basic systems outlined earlier.

The problem is best split into two different parts. Firstly, the function of the neuron and secondly its connections (this is where the modularity of the network arises). These two parts, in a way, are roughly equivalent to the distinction between active and organisational machines. It is to the second of these that we will mainly apply our evolutionary ideas.

B.2 Components necessary to make a general neural system.

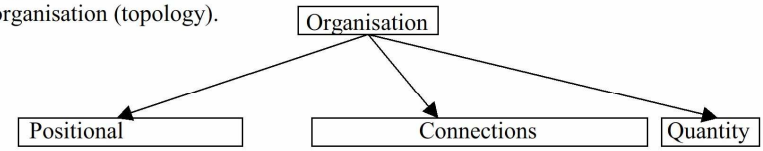
First let us consider the functionality of the neuron, what is required here? Well, there are really just two components, figure 5.

Fig 5.



Secondly, we need to consider the organisational mechanism of the network. We can identify the most important aspects of this from nature; these are shown in figure 6.

Fig 6, Network organisation (topology).



The importance of these will be become obvious as we proceed.

B.3 Functionality.

Let us consider the two components of the function of the neuron.

B.3.1 Unit function.

Contrary to popular belief, all neurons do not have the same unit functionality. Actually, some operate in quite different ways from each other. In fact we do not fully understand the operation of some of the more exotic types. What is needed is an evolutionary system which can evolve any reasonable neural function. This would mimic the universal machines within the cell which can arrange themselves to produce a wide range of electrical behaviours.

Although we could simply choose a combination of operators genetically [17], we can also model any function using an infinite Power Series (for example a Taylor Series):

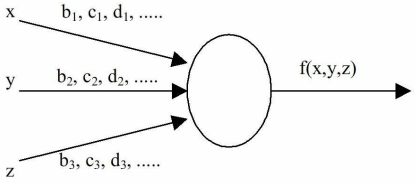
$$y = a + bx + cx^2 + ... + \delta_n x^{n-1}$$

This is the basic series which is given in most text books, However, it is extendable to any number of variables (and hence any number of dimensions)- for example in 3 dimensions:

$$f(x, y, z) = a + b_1 x + b_2 y + b_3 z + c_1 x^2 + ...etc$$

If this were an ANN x, y and z would be the three inputs and b_n, c_n and d_n would be the respective weights as shown in figure 7.

Fig 7, a polynomial ANN.

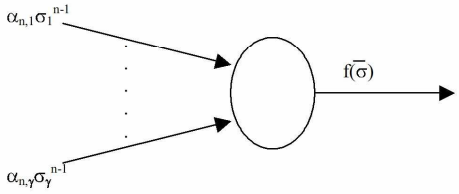


Or, more generally, for γ variables and a χ order series.

$$f(\bar{\sigma}) = \sum_{n=1}^{\chi}\sum_{m=1}^{\gamma}\alpha_{n,m}\sigma_m^{n-1}$$

Shown in figure 8.

Fig 8, a more generalised neuron.



As an example, we could approximate a circle of radius $\sqrt{a}$ with a two variable second order Power Series (coefficients $\bar{b} = 0$).

$$f(x, y) = x^2 + y^2 - a$$

We could also generalise the series more by adding fractional powers $\sigma_m^{1/n-1}$ or products of inputs $\sigma_m^n\sigma_j^i$, etc. Actually, the normal McCulloch-Pitts neuron is nothing more than a first order series. If the term *a* is included, it corresponds to bias in the network. It should be obvious when one considers that the separator in a normal neuron is a straight line that a first order power series is simply a straight line approximation to a function. A first order network may approximate any function provided there are enough neurons in the system. The coefficient vectors a, b, c, etc are weights which must be trained. So we have a system which can emulate almost any function of the neuron (which has a continuous response). Now let us consider time response.

We could model any time response with another Power Series:

$$f(t) = a + bt + ct^2 + ...etc$$

but there is no evidence that the temporal properties of neurons are this complex. So it may be easier to simply make a evolvable (or trainable) time decay:

$$f(t) = ae^{bt} \qquad (a, b \in \mathbb{R})$$

By applying a threshold function to this it is easy to produce a good approximation to "spiky" neurons. Usually a would be 1 and b negative, and so our general neuron has this function:

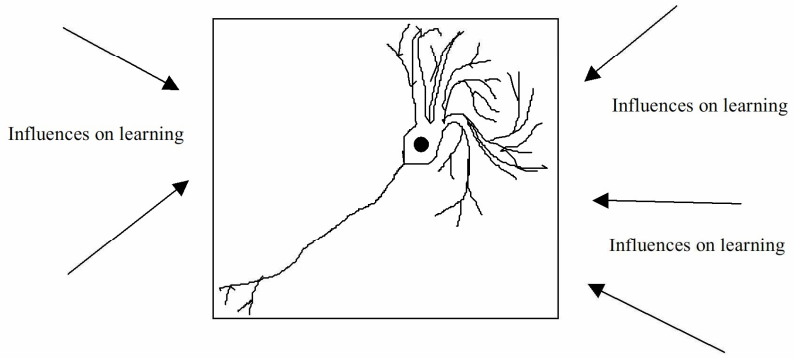
$$f(neuron) = T(\bar{i})\, f(t)$$

Where T($\bar{i}$) is a nth order Power Series of the input vector $\bar{i}$. A squashing function may be applied to this if necessary and we could also add a refractory (rest) period [18].

B.3.2 Learning.

How could such a system learn? Consider a biological neuron; what are its possible learning mechanisms? We can find the answer to this problem by drawing the isolated neuron - a "neuron in a box", figure 9, and writing down all the *possible* parameters which could allow it to learn (After all, the neuron is only connected to other neurons. So the only possible mechanisms involve either signals transmitted from other units or chemical signals from the surrounding medium).

Fig 9 Isolated "neuron in a box"



Some examples might be:

a) Biochemical. The neuron is bathed in a 'soup' of intercellular fluid. Hormonal and other stimuli can affect this soup. For example, its constituents might change in the presence of hunger, pain, fright, the urge to mate, etc (or, in a simple system, simply a good / bad signal). The signal would effect whole regions, not individual synapses. Let us call this component B.
b) Hebbian (or Anti-Hebbian). A synapse gets strengthened through use. The more activity it has, the stronger it becomes. This contribution effects each synapse individually and is therefore a matrix. Let us call this component $\overline{H}$.

c) Synchronous (or Anti). A synapse gets strengthened when it is active at the same time as others (and weakened if it is not). Like the Hebbian contribution, this is therefore a matrix. Let us call this component $\overline{S}$.
d) Mediated. One synapse (or a group of them) controls the strength of an other.

$$M_{i,j} = ax + by + ...etc$$

Where a, b, etc are constants defined by evolution (or learned), x, y etc are other inputs to the neuron. This effect may be difficult to achieve in practical terms, and since there is no evidence for it from biology, it may be best left out of a model.

The mathematical forms of these learning mechanisms have not been expanded because they are reasonably obvious. Of course the programmer may choose to add one of the traditional learning methods to (or instead of) these.

The total learning contribution to the weight matrix could be expressed as.

$$\overline{W^+} = \overline{W} + \eta(aB(b\overline{H} + c\overline{S}))$$

W^+ = updated weight matrix. η = learning rate. a, b, c = sensitivity to individual learning types. W = old weight matrix.

It is fairly obvious that these mechanisms, which are biologically realistic, are highly dependent on network topology. After all, a synchronous or mediated learning strategy will only work with neurons which are placed in the correct positions within networks – it is easy to see that in other positions they could have no effect or cause the network to deviate away from the required response. They are therefore probably only suitable for use with networks whose topology is defined using evolutionary mechanisms. This topology dependence may be the reason why it has been so difficult to decide on biologically feasible learning mechanisms for ANNs.

B.4 Organisation

It has already been explained that, of the three elements of network specification, (learning, neural function and connection pattern), the connection pattern or topology of the network would appear to be particularly important. One can perhaps see this if one considers that all machinery, whether artificial or natural, is made from smaller elements, indeed, in the ultimate analysis from atoms themselves. It is merely a matter of selecting and placing (and in the case of neurons, connecting) these elements to produce the response required.

In the case of neural networks (and also electronic systems is general) an important aspect of the system is its modularity. In the case of the robotic system presented in D McMinn's thesis, the network has been artificially partitioned into Reflexes, CPG and Biological Oscillator. When one attempts to evolve this system as a fully interconnected homogeneous network, the result is failure - or at least considerable difficulty. Therefore, the standard genetic algorithm is not suitable and we must look to a new paradigm – one which has the evolution of a modular system integrated into it.

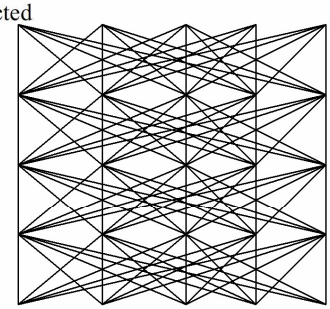
Before embarking on an exploration of the methods capable of doing this, let us first pause to consider *why* a fully connected network seems incapable of producing the results we require. Work needs to be done on the theoretical basis of this question as there seems to be a theorem missing from systems theory which adequately explains it. However three possible avenues of enquiry might be:

a) The fully connected networks currently used are too small.
Almost all networks are limited by orthodoxy to three layers. Kolmogorov's theorem is usually cited to support this view. However, this assumes that the problem is a single data domain solvable with straight line separators. In practice however, the problem may involve multiple data domains or multiple data transformations and so the simple three layer model may not be adequate.

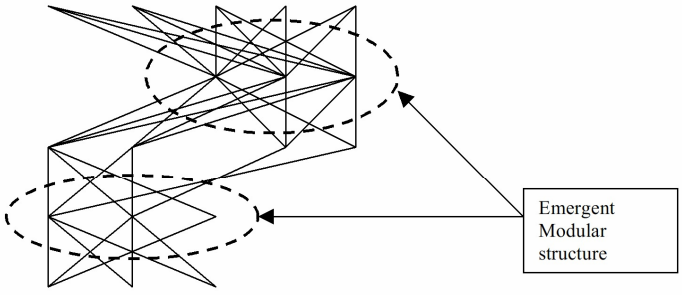
In theory a homogeneous "deep" network like that shown in figure 10 should, through its learning process, become a modular network as shown - the deleted weights becoming very small. That such networks are not often experimented with may be one reason why we don't observe this phenomena. Another could be that BP is not a suitable algorithm for training such a system (although a GA should do it).

Fig 10, a "deep" network.

i) Fully connected



ii) After it has learned (deleted connections have small weights)



b) The networks have to process separate datasets.

To illustrate this point let us take an example. Suppose that we have inputs from a complex visual system which requires several layers of processing (for example feature detection) and also inputs from an equally complex auditory system. If we tried to process both of these in the same network it would cause difficulty because the two independent datasets are not separated and can interfere with one another. the same argument might also apply to complex operations even within the same set. So separate discrete networks would certainly make this processing task easier.

c) A single large network represents too large a search space for the evolutionary algorithm.
Several smaller networks would represent a smaller search space, but in McMinn's networks the size was small and the single large network still did not converge well, which indicates that this explanation may be false.

B.4.1 Organisation - methods
Let us now turn our attention to some practical methods of introducing modularity into the network. At the simplest level, there are two basic approaches to this. One could adopt a method based on developmental biology, using computer models of cellular differentiation, migration and pattern formation to fashion a network. Alternatively, we could take an engineering standpoint and try and extract the essence of what is required to produce a modular network and code this from a purely pragmatic point of view. However, whichever path one chooses, there are certain obvious aspects that the algorithm will have to accommodate:

1. Position.	In biologically inspired algorithms, the neurons must be placed in the correct positions in the "evolution space" or body.
2. Number.	At each position the algorithm will have to decide the number of neurons to be present in each module and the number of modules.
3. Function.	What the units actually do in each position.
4. Connections.	Connections must be established both locally between neurons in the same module and globally between different modules.

Although these ideas may be implicit in all the solutions resulting from the different ideas described below, some may be easier to implement and more successful than others. Outlined are five methods for creating modular networks, these are:

i) Using models of biological neural development.
ii) Using a set of production rules.
iii) Using fractals or similar mathematical models of natural patterns.
iv) Adapting traditional evolutionary techniques such as the Genetic Algorithm.
v) Using Direct Modular Growth on a network.

Let us consider each of these in turn.

B.4.2 Modelling Biology
In biology, the formation of patterns in the organism is not particularly well understood. There are however certain generalisations we can make which are probably sufficient in themselves to allow us to develop an Artificial Neural Network that can evolve modularity [20]. The processes involved are differentiation,

proliferation, migration, and patterning in normal cells. On top of these, there is cell orientation and connection formation in the case of neurons. The order in which these occur may be different in different cell types. Let us consider these aspects separately.

In the new embryo, the cells have no set function. Differentiation is the process whereby these "stem cells" become specialised to be bone, muscle, neurons or one of the many other cell types in the body. When the fertilised egg cell attaches itself to the womb lining, it may be a chemical signal from the mother that starts development. It is thought that a chemical gradient starts cell differentiation. Cells further away from the point of attachment in the womb may be below the threshold level, whereas those closer are above it. This triggers a change in the cells closest to the womb lining that is the start of the road to development. Large aggregates of cells differentiate first, forming gross structures; these in turn start producing other gradients which allow differentiation to take place on a smaller scale and so on. The *birthdays* of the cells (when they differentiated) are important in this respect, as earlier cells tend to form the larger structures and later cells the finer. As an example, in the limb bud of an animal, developmental genes progressively get switched on and form finer and finer structure, starting with the form of the limb and working down eventually to the digits.

Proliferation is tied up with differentiation. Obviously there needs to be enough cells to build the structures of the organism and since development starts from a single cell, the embryo's cells are continuously dividing and so that the cell mass becomes larger.

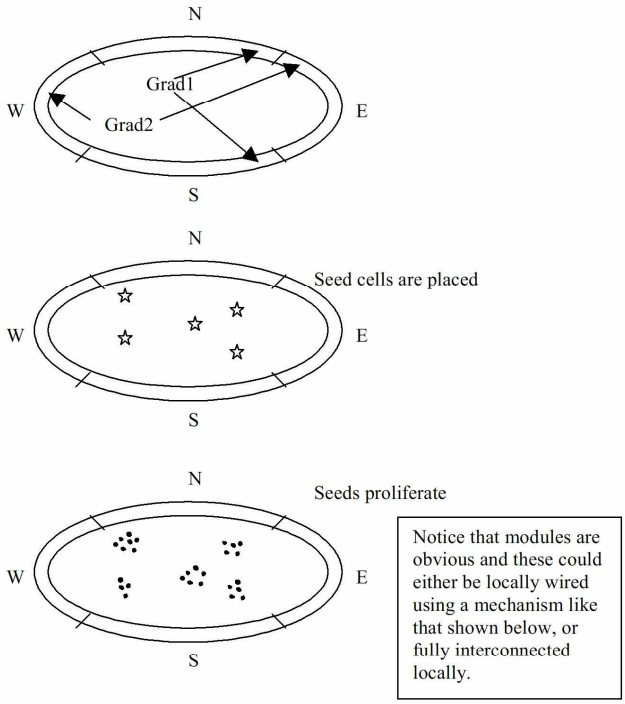
Migration is important in some cell types and in particular neurons. Basically the idea is that cells may proliferate and differentiate far away from the point where they are finally going to be used and so they travel or migrate to their final homes. Exactly why this evolved is not particularly clear, but it may be an accident of evolution. Neurons originate in the same layer as the skin (they may have originally been skin cells that became specialised during the course of evolution) and they have to move if other tissue is to become enervated. Migration is mediated by chemical attractors and cellular adhesion.

Patterning is the result of the previous processes. Cells can form patterns in situ by proliferation and differentiation – this is what happens with bone and muscle. As different cells form, they switch on new developmental genes, which form their own gradients, and spark the next level of differentiation and pattern detail. This mechanism results in symmetrical structures (because gradients are involved) and fractal structures (because larger patterns cause smaller ones to form in more and more detail). Whether migration is involved or not, these three mechanisms have the result that: *The right type of tissue is at the right place and in the right quantity.*

From the point of view of a computer model, we can simplify the picture to say that we need to place the correct quantity of cells in the correct place. This may be done if artificial gradients are set up which allow "seed cells" to be placed, figure 11. These may be assigned numbers which signify their attraction towards two perpendicular poles, therefore locating themselves in 2D space (3D space is obviously just as easy). The seeds then proliferate to form the correct number of neurons. One has got to make

sure that the genetic coding for this is such that when mutations or crossover occur they conserve positional and proliferation information.

Figure 11, Artificial Gradients can be set up in an evolution space.



Cell orientation and the formation of connections are particular to neurons as they have to form "wiring". Experiments show that axons form along chemical pathways ensuring that one part of the brain is connected, as appropriate, to another. The mechanism could be mimicked by allowing the neurons to grow connections from only one side, these would then progress outwards until they connected with another neuron which was compatible with the first (compatibility values could be attached to neurons). Or alternatively, all neurons (and the sensory inputs and motor outputs of the network) may be labelled with markers which specify attractors for the growth of connections from the placed neurons. Those which are closer or have stronger markers 'win' and connections are made towards them.

In schemes like this, numbers are attached to each neuron type which allow the units to find their own connection patterns. This idea is equivalent to proteins getting expressed in individual neuron types hence allowing them to self-organise. The organisational machinery is contained within the individual neurons and not in a universal genotype. Action is therefore devolved.

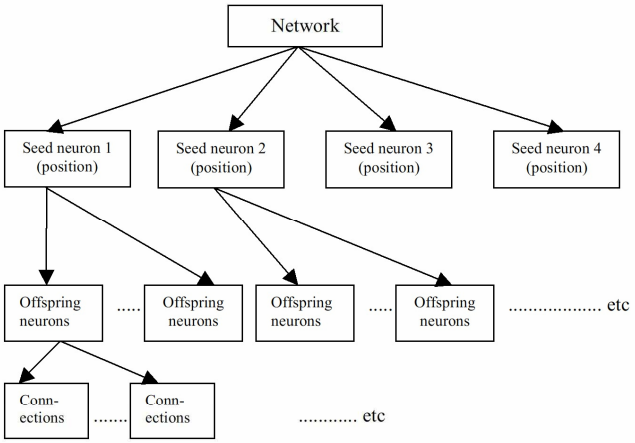
Of course this example is used simply to illustrate the issues involved in implementing a scheme based on nature. One could include much more complex

details from developmental biology, right up to creating a sophisticated model which simulated the effect of known genes (for example the hox genes), or axon growth factor. However, the basic idea remains the same and it is doubtful if this level of detail confers much advantage over a simpler algorithm that captures the essence of the system.

B.4.3 Production Rules and Trees

Another system which captures the biological approach, but at a more abstract level, is to use a set of Production Rules to configure the network. Either the rules, or the network that they produce, can be conveniently written down in the form of a tree - following the tree from top to bottom produces the network. A typical tree for encoding a network shown in figure 11 is shown in figure 12.

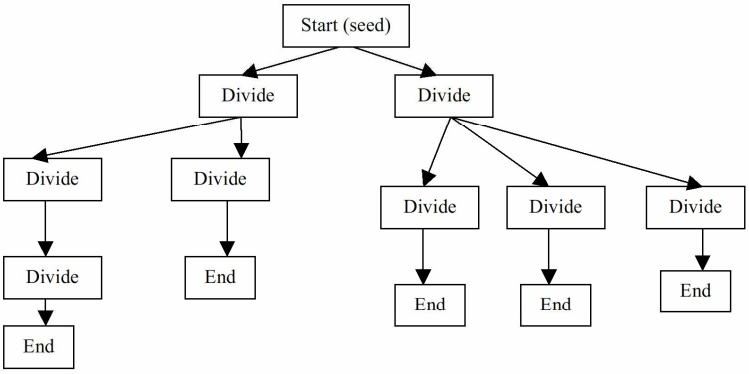
Figure 12, A tree encoding a network.



Of course this is really just an alternative way of representing the biological system which was illustrated in the previous example. The tree comes under the control of the Evolutionary Algorithm - which is basically the production rules for creating it. They might include, for example, neuron proliferation (turning one neuron into many), migration and so on. Arranging the system like this however has the advantage that one can allow mutation and recombination (crossover) in the G.P. fashion by swapping or mutating individual branches of the tree. In this way important sections of the network can be reused without having to be re-evolved.

This is just one of many possible trees which could be used to encode the network and many different production rules which could be used. The alternative to encoding the network is to encode the production rules themselves as the tree [19], figure 13, the rules in this case being produced by the Evolutionary Algorithm.

Figure 13, Production rules.



The connections may be part of the production rules or evolved through the hierarchy established by the rules themselves. The arguments about recombination and mutation outlined for the previous case apply here also.

B.4.4 Fractals

Many of the patterns produced by living organisms are self-similar at different scales. This phenomenon can be seen clearly in many plants - good examples are ferns. The study of such structures has become popular and important in recent years and some aspects of them (although not all) are tied up with non-linear systems and in particular so-called chaotic dynamics. Mappings of the stability of such systems produce the beautiful "Mandelbrot Set" and others. Other self-similar structures can be produced by the repeated application of a formation rule, by geometric transformations or by self-organisation in automata. However they are produced, such structures are often referred to as "Fractals" [21]. The word Fractal is widely misused and applies to many different types of system - here we are taking a loose meaning, understanding it to mean a mathematical model which forms approximately self-similar systems that resemble biological structures.

The importance of fractals is that they are often the mathematical form resulting from the action of genetics. They display symmetry (as does the biological nervous system) and are generally "lumpy" (modular). The idea that fractals could be used to define neural net topology has been suggested before [22], but researchers have yet to take it seriously enough to produce working systems. One of the main reasons for this is that it would require the design or discovery of a fractal system suited for the job (currently available fractals are not).

Not all types of fractal would be suitable for use in the definition of neural nets. There are three important attributes which the fractal should have. Firstly, it must be constructed from lines, because these will be the connections in the network. Secondly, it must be able to grow, from a simple to a complex structure, the later stages building on the earlier. Finally, it must have "nodes" at which neurons or modules can be placed.

There are three types of fractal (and perhaps several others) which almost immediately fit the bill. These are outlined below.

i) Dawkin's Biomorphs.
Biomorphs made their appearance in the book 'The Blind Watchmaker', by Richard Dawkins [23]. In the book, Dawkins explained the operation of evolution in animal populations. To aid his explanation of the evolutionary process, he developed a simple computer program which produced branching structures which he named Biomorphs. Each time the program ran, the Biomorph grew; that is, it added another layer to its structure, figure 14. Dawkins encoded a sequence of numbers which represented the structure of the Biomorph as a string and which can be mutated - hence the method is inherently genetic.

Fig 14, Typical Biomorph development.

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

The process has several important properties:
- It represented a structured search and produced a structured result
- It allowed growth from simple to complex structures
- It was based on sound biology (although the final algorithm is not a direct model of nature).
- It appeared to be adaptable to neural nets

The system, as presented by Dawkins, is not in itself suitable for use with neural networks. However, it may be possible to use a similar algorithm as the basis of a system suitable for use with ANNs.

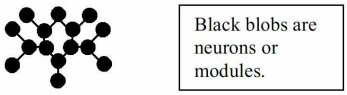
(ii) "L" (Lindenmayer) systems.
These are often seen in the context of producing beautiful plant-like patterns, particularly ferns and trees. They are generated through the repeated application of rules. Like the Biomorphs described above they can "grow" from simple to complex and can display line features. Other similar fractals include Pythagoris and Mandelbrot trees.

(iii) Cellular Automata.
Certain types of Cellular Automata can also be configured to produce similar structures to fractals, both symmetrical and non-symmetrical. They can form branch-like dendritic structures in the same way as the other fractals discussed. However these are built up from the self-assembling properties of the automata rather than from a mathematical formula, transformations or growth rules like the other structures. In the case of the automata, systems can be envisaged where the neurons are the

automata themselves, or alternatively, where the dendritic chains simply form the "wiring" of the network. Both "L" systems and automata are illustrated in Stewart's book [21].

The three examples given above are simply used to illustrate the possibility of the fractal use in ANN definition, very little real work has been done in this area. As mentioned, none of the systems, as they presently stand, are really suitable and a structure really needs to be designed which "fits the bill". However, once this is done, there are two way to use the fractal. Firstly, the nodes of the fractal could be used as placement points for neurons and the branches their connections; this is illustrated in figure 15 below. Or, alternately, the nodes could be placement points for network modules (probably fully interconnected). Looking at figure 15, one can see why the basic Biomorph is not suitable for use. However, other fractals do display fully interconnected modules with sparse interconnectivity between them.

Fig 15, Using a fractal to produce connectivity.



B.4.5 Altering existing Evolutionary Algorithms

One obvious area which we have not yet touched on is whether the standard evolutionary algorithms such as Genetic Algorithms or Evolutionary Strategies could be modified to produce a modular result. Some possible ways of achieving this are:

1) Define each module by a section of chromosome within the population of the GA. Each neuron appears as a sub-string within the module and for each might be coded: (1) The weights, (2) The function of the unit and (2) The wiring or topology. Some neurons are designated inputs and some outputs. These allow connections to other modules (for ease of coding the inputs and outputs can come at the beginning and end of each string and delineate it). As modules are added to the structure, the strings are allowed to grow.

2) An extension of this idea is to have a fixed string length for each module, but to allow certain parts of the string to turn on or off or define the network (akin to pruning). This is an attempt to get around the problem of strings having to grow if modules become bigger or alternatively, have an independent GA for each module.

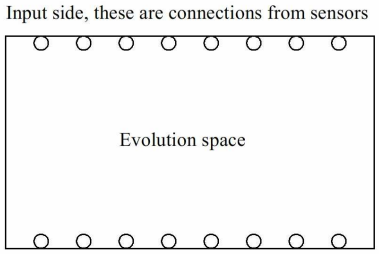
A new module could be created when the GA string reaches a certain fixed size - at which point it would automatically "bud off" a sub-network which would be wired independently of the first. Alternately this could be done once the network had fulfilled its function (once the fitness function was as high as possible).

B.4.6 Direct Growth

At the opposite end of the spectrum from the biological approach is a purely engineering point of view. From an engineering view we simply want to place groups

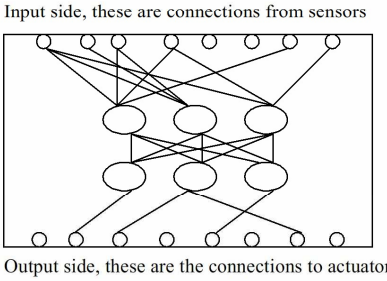
of neurons into a modular structure and connect them together to form a system under the control of evolution. There are a number of approaches we could adopt to achieve this. Consider the concept of an "evolution space" were the network will develop as shown in figure 16.

Fig 16, An "evolution space"



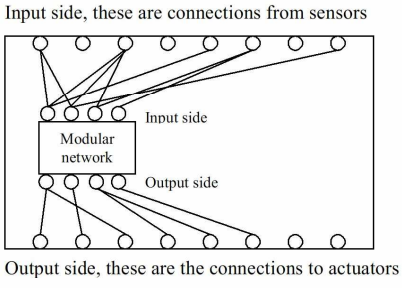
In the systems developed in McMinn's thesis and in previous projects a single network is caused to evolve in the space as shown below in figure 17

Figure 17, A network, as evolved in McMinn's thesis.



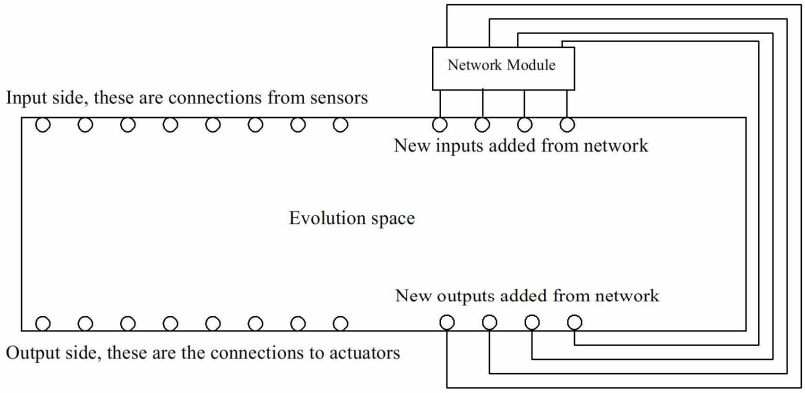
However, the concept can be easy adopted to serve modular networks in two obvious ways. Firstly, by replacing individual neurons in the diagram above by networks, figure 18.

Fig 18, Adapting the above idea to modular networks.



The internal wiring of the modular network would be determined by one evolutionary algorithm (the local algorithm), the wiring between networks and inputs/outputs by another (the global algorithm). The algorithm would start with few inputs and outputs and build up by added these and more modular networks (this is the incremental part of the system described in detail in the next section). How much of the previous network is conserved on each iteration could be varied in different algorithms. Another, possibly more flexible system is shown in figure 19.

Figure 19, Another system.



In this case only connections evolve in the evolution space (although one could also allow single neurons to be placed in this space). The operation other than this is the same as the previous system described.

Naturally, the best aspects of the systems described in the previous section may be "mixed and matched" to provide a better result.

B.4.7 Other issues in Modular Evolution

i) Limitations of Evolutionary Techniques

There are several problems with Evolutionary Algorithms. Some are fairly obvious and others are subtle. One of the more subtle problems has to do with the inter-reaction between different parts of the system and is well illustrated by Timetabling.

Time Tabling is extremely important in academic and some other environments and basically involves assigning lecture classes, lecturers, rooms and timeslots so that classes run smoothly. Because the resources are limited it is not as trivial a problem as it might, at first, appear. Generally it is done by a human carefully placing the correct classes and teachers into the correct time slots, but this is tedious and time consuming. Genetic Algorithms have been devised to try and solve the Timetabling Problem.

The Genetic Algorithm is actually very good at *helping* with the problem. It can often nearly complete the timetable - however, a human can always do better and usually has to finish the task. The reason for this turns out to be important: a human can see

intuitively that placing certain time slots in a particular order early in the structure allows other slots to fit more easily later. Of course, given enough time, a GA would stumble across the same combination, but it is just one in a huge number. This is the advantage of design or intuition over evolution.

The same argument also applies to modular networks. Evolution has many different combinations to search and in any large network it may be a next to impossible task for it to find a particularly important combination of parameters which satisfy a particular requirement.

The other problem of modular network design is anything but subtle - what do the modules actually do? If the weights in a module have to be trained then how is the fitness function determined. This is analogous to training the hidden layer of a normal neural net.

In the following discussion, a possible answer to these two questions will be given in terms of incremental evolution. It should be remembered that, whatever the researcher's doubts about the difficulty of implementing this, the two tenants of evolutionary connectionism should be borne in mind:

1. Can a machine be intelligent to the point of consciousness?
Yes. The biological brain shows this to be true and it is a machine.

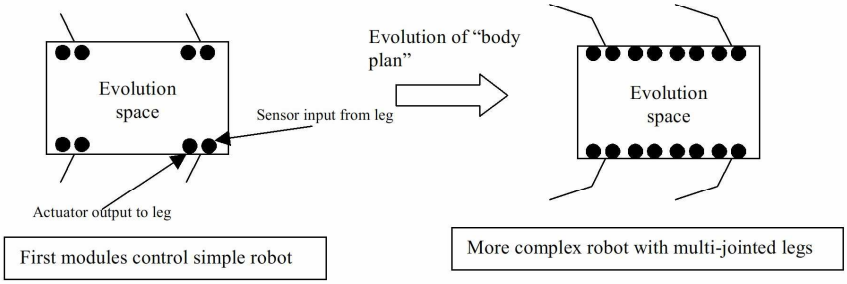
2. Is it possible to create such a machine?
Yes. Nature has designed such a machine through blind biological evolution and so therefore can artificial evolution.

ii) Possible solutions to these dilemmas.
The solution to the problems mentioned above lies in *Incremental Evolution* [1]. To understand this, consider the evolutionary development of the human brain. As the monkey brain developed into the ape brain and the ape into the human brain, did the whole structure re-wire itself each time? No, re-wiring tens of billions of neurons is simply not feasible. A much more reasonable explanation is that the brain *added* to its structure, building new modules on top of old. The scaffolding of the mammalian (and indeed the reptilian) nervous system is still present, buried deep within our own [24].

This is the process which can help us to evolve Modular Artificial Nervous Systems. Start with a few simple modules which allow the system to function at a primitive level in a constricted environment and build on these as the system develops.

Such an approach requires that sensor inputs and actuator outputs from the network must also evolve in complexity along with network - in effect evolving the body plan of the robot. For example, a legged robot might start of with simple, single degree of freedom legs, each with a single control input and a single sensory output as shown in figure 20a.

Fig 20, Robot evolution.



a) b)

Once the system can control the simple legs, extra degrees of freedom can be added and the network allowed to evolve (incrementally), figure 20b. The control system for a prosthetic might also proceed along similar lines, starting with gross movements and working down finally to digits. Likewise, a sensory system like vision would start perhaps with a single eye spot - only able to perceive light and dark and evolve complexity from there.

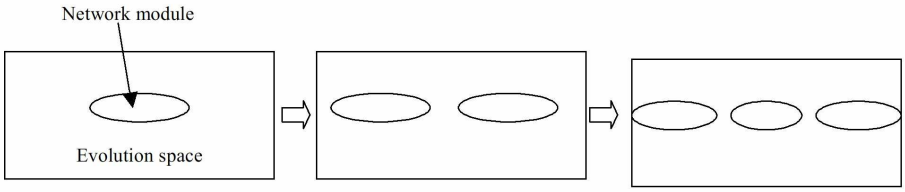
Obviously, any complex artificial organism would start life (as in both evolutionary and developmental biology) as a simple group of cells and develop.

One can perhaps also see some limitations in this structure. Although it is possible to see intuitively that it might work for robots, it probably would not work (or at least to work so well) for systems with a high degree of serialism in their design. One might postulate from this that there are some systems which nature would find it very difficult to design but which humankind would find easy. This is an aspect which has yet to be investigated.

iii) Operation of a practical algorithm

Let us consider how such a system might operate in a practical sense. This is best illustrated by an example. Suppose that we have a direct growth system as described in section B.4.6. As described above, the algorithm will start with a simple evolution space with few modules (one in this case), figure 21a.

Figure 21, Evolutionary algorithm operates on direct growth.



a) b) c)

This one module can be configured using two nested Evolutionary Algorithms. The first EA wires the module to the inputs and outputs of the system (the Global EA), the second wires the module internally (the local EA).

To illustrate this further: the global EA first selects an external connection topology, then the local EA configures the module's internal wiring. Once the internal wiring has reached its peak fitness this configuration is stored and the global EA runs onto its next set-up. In this way, the combination of best local and global connections can be found.

Having wired the simple system up, the algorithm next adds another module and more input/outputs to the outside world. The process is then repeated except that the previously wired modules are retained and the new modules wired on top (global connections to previous modules may change but local connections are retained), figures 21b and c.

The exact formation of the algorithm has yet to be developed. For example, it may be that operation is better if some changes to previously wired modules are allowed, particularly to modules which have been recently placed. This could be implemented using a simulated annealing algorithm running alongside the EAs. Multiple levels of modular networks (networks of networks) could be dealt with in the same way – using nested EAs.

Another important aspect of this type of system is the genetic operators which are used in the EA. Conventional EAs use recombination and mutation, but in nature large sections of DNA can be accidentally deleted, inserted, copied and inverted. This potentially allows chunks of network to be duplicated. Such duplication is a huge advantage. It could allow hard-to-evolve sections of network or whole modules to be reused. One can see the advantage of this given the problem described in time tabling above. In fact large parts of the human brain consists of millions of repeated modules, consisting of a few hundred neurons. These may be "learning units", configured to expedite learning – which may be heavily intertwined with topology as described in section B.3.2.

Conclusions

This paper has attempted to explain the lack of success of traditional Evolutionary Algorithms by outlining the differences between them and Biological Evolution. In particular the point that DNA codes self-organising chemicals which can produce repeating (modular) structures. It is pointed out that a artificial equivalent of this is difficult to implement.

As an alternative, several new methods of considering ANNs are suggested including several new methods of organising the ANN topology.

Finally some suggestions about the practical running of the algorithm are made.

References:

[1] The synthesis of Artificial Neural Networks using Single String Evolutionary Techniques, C MacLeod, PhD Thesis, The Robert Gordon University, 1999.
[2] D McMinn, PhD Thesis, The Robert Gordon University, 2001.
[3] A Framework for evolution of an animat nervous system, C MacLeod et al, EUREL Advanced Robotics Systems Development (conf), 1998. Paper 18.
[4] An evolutionary artificial nervous system for Animat locomotion, D McMinn et al, ICEANN -Engineering applications of neural networks (conf), 2000. p170-6.
[5] Combinations of Genetic Algorithms and Neural Networks: A survey of the state of the Art, J D Schaffer D Whitley L J Eshelman, COGANN -92 (conf), IEEE comp soc press, 1992.
[6] Molecular biology of the cell (3rd edition), B Alberts, D Bray et al, Garland Publishing Inc, 1994. p104 - 10.
[7] As reference 6, but p204-5, 404 -5, 573-4.
[8] As reference 6, but p104 - 10, 56 - 7, 111 - 35.
[9] As reference 6, but p24 -5.
[10] As reference 6, but p556 -60, also ch12.
[11] As reference 6, but p95 - 7, 111 - 35.
[12] As reference 6, but p556 - 560, also ch12.
[13] As reference 6, but p721 – 27, 1050 – 66, 1080 – 92, ch 21.
[14] Open University video – course s202, Biology, form and function course, video title : Patterns in developing gradients.
[15] Cellular Development, D R Garrod, Chapman-Hall, 1973.
[16] As reference 6, but p1119 – 30.
[17] As reference 1, but pA78 - A98.
[18] Investigation of an advanced neural network model, S I Monsen, MSc Thesis, The Robert Gordon University, 2000.
[19] Genetic synthesis of Boolean neural networks with a cell rewriting development process, F Gruau, *In* Combinations of Genetic Algorithms and Neural Networks, IEEE Computer Soc Press, 1992.
[20] A new approach to neural network topology, J Murray, MSc Thesis, The Robert Gordon University, 2000.
[21] Life's other secret, Ian Stewart, Penguin books, 1989.
[22] As reference 1, but p70-76, A3-A5.
[23] The blind watchmaker, R Dawkins, Penguin books, 1991.
[24] The brain: The last frontier, R M Restak, Warner Books, 1979. p 50 - 52.

Appendix C

Backpropagation Algorithm, SLT Delta Rule and Pulse-Width Backpropagation

C.1 Introduction to the Appendix

This appendix presents the Backpropagation Algorithm as used for static-domain training of Multi-Layer Perceptrons in this thesis. A component of this algorithm is the Delta Rule, which is used for training Single-Layer Perceptrons. A specifically derived Delta Rule is presented; it was created and used for training Single-Layer Taylor Series networks. The last algorithm presented is a derived version of Backpropagation, which suits the requirements for the time-domain training of the pulse-width modulated Artificial BioChemical Network.

C.2 The Backpropagation Algorithm

Backpropagation was introduced by Rumelhart, Hinton and Williams, [1986]. This remains the most prevalent training method used in feed-forward networks. There have been many improvements made to the initial method since its introduction, however the algorithm was used in this thesis to compare the performance of different artificial units and did not examine the assorted alternative versions of the algorithm.

Backpropagation is a gradient-following error-minimising algorithm. This means that as the algorithm progresses the parameter alterations cause the error to follow the downwards slope until a minimum level is reached. This error minimum is usually not encountered as the stop criterion (a higher target error) should be found first.

$$e_{epoch} = \tfrac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(t_{ij} - o_{ij} \right)^2 \qquad \text{equation C.1}$$

The epoch error (e_{epoch}) is defined as in equation C.1 where there are *m* data-patterns. This is called the Least Mean Square (LMS) error. Each pattern error (e_p) is the sum of its

neuron errors where there are *n* output neurons. The neuron error is designated (e_n) where *n* may be replaced by the specific neuron identifier.

- The epoch error is calculated each epoch and compared against the stop criterion.
- Only the neuron errors are used in the training algorithm.

Summary of the Backpropagation Algorithm operation:

Once a pattern has completed its forward pass of the MLP training begins. It is customary to encompass all of the training from this point onwards as Backpropagation; however the initial training on the output layer follows Perceptron training delta rules and was used before Backpropagation was introduced.

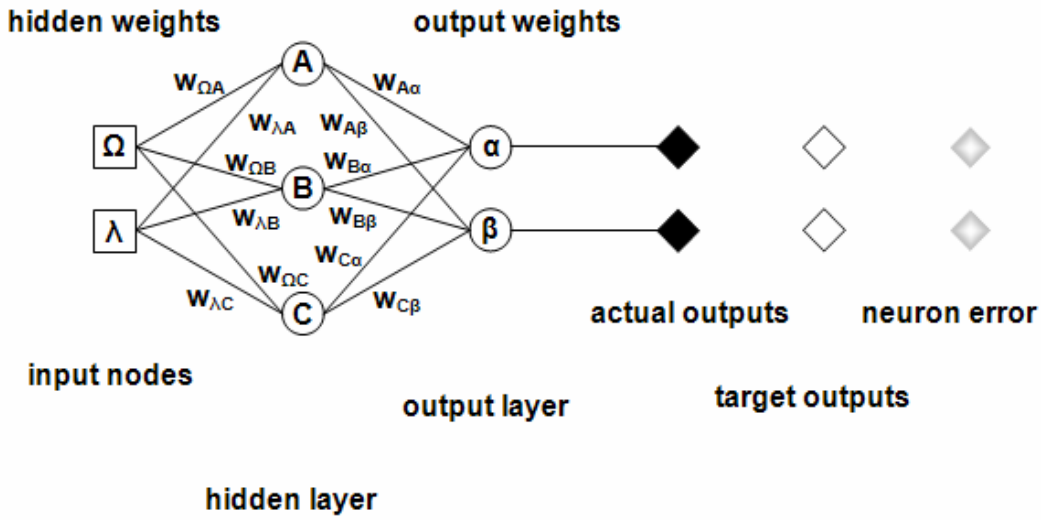


Figure C.1 – Backpropagation of an MLP

Figure C.1 shows an MLP arrangement of nodes and neurons. Training proceeds as follows:

- For each pattern in turn, the output neurons errors e_n are calculated. For the output neurons;

$$e_\alpha = \left(t\arg et_\alpha - out_\alpha\right) \qquad \text{equation C.2a}$$

$$e_\beta = \left(t\arg et_\beta - out_\beta\right) \qquad \text{equation C.2b}$$

- The errors e_n are used to calculate an associated parameter for each output neuron. This assesses the change its connecting weights should undergo and incorporates the first derivative of the output function used. In this case the logistic sigmoid. The parameter is denoted by the lower case Greek letter delta (δ) - for the output neurons this is;

$$\delta_\alpha = out_\alpha \cdot \left(1 - out_\alpha\right) \cdot e_\alpha \qquad \text{equation C.3a}$$

$$\delta_\beta = out_\beta \cdot \left(1 - out_\beta\right) \cdot e_\beta \qquad \text{equation C.3b}$$

- The deltas are used to calculate the new values of each connecting weight using the Delta Rule. Weight values are designated w_{ij} where i is the connecting unit in the previous layer and j is the current neuron. The new weight is indicated as w^+. For the above network the calculations are as follows;

$$w^+_{A\alpha} = w_{A\alpha} + \eta \cdot \delta_\alpha \cdot out_A \qquad \text{equation C.4a}$$

$$w^+_{B\alpha} = w_{B\alpha} + \eta \cdot \delta_\alpha \cdot out_{\mathbf{B}} \qquad \text{equation C.4b}$$

$$w^+_{C\alpha} = w_{C\alpha} + \eta \cdot \delta_\alpha \cdot out_C \qquad \text{equation C.4c}$$

$$w^+_{A\beta} = w_{A\beta} + \eta \cdot \delta_\beta \cdot out_A \qquad \text{equation C.4d}$$

$$w^+_{B\beta} = w_{B\beta} + \eta \cdot \delta_\beta \cdot out_{\mathbf{B}} \qquad \text{equation C.4e}$$

$$w^+_{C\beta} = w_{C\beta} + \eta \cdot \delta_\beta \cdot out_C \qquad \text{equation C.4f}$$

In equations C.4 the term η is the Greek lower case eta and is a constant called the learning rate. Usually a $\eta \leq 1$. At this stage the weights for the output layer have been altered; however, as indicated, Backpropagation has not been implemented.

Backpropagation refers to the method by which the error at each output node is back-propagated to the hidden nodes so that the correct share of responsibility for this error can

be allocated and the hidden weights adjusted. This is the "credit assignment" problem defined by Minsky [1961].

- As with the output layer the first stage in any hidden layer is to calculate the delta values for each neuron, as follows;

$$\delta_A = out_A \cdot (1 - out_A) \cdot (\delta_\alpha \cdot w_{A\alpha} + \delta_\beta \cdot w_{A\beta}) \qquad \text{equation C.5a}$$

$$\delta_B = out_B \cdot (1 - out_B) \cdot (\delta_\alpha \cdot w_{B\alpha} + \delta_\beta \cdot w_{B\beta}) \qquad \text{equation C.5b}$$

$$\delta_C = out_C \cdot (1 - out_C) \cdot (\delta_\alpha \cdot w_{C\alpha} + \delta_\beta \cdot w_{C\beta}) \qquad \text{equation C.5c}$$

In equations C.5 the neuron error e_n of ($target_n$ – $output_n$) is replaced by the credit assignment term. This is the delta value of each neuron in the subsequent layer multiplied by the strength of the connection. Using their delta terms, the new weights for each hidden neuron are found in exactly the same method as for the output neurons.

$$w^+_{\Omega A} = w_{\Omega A} + \eta \cdot \delta_A \cdot out_\Omega \qquad \text{equation C.6a}$$

$$w^+_{\Omega B} = w_{\Omega B} + \eta \cdot \delta_B \cdot out_\Omega \qquad \text{equation C.6b}$$

$$w^+_{\Omega C} = w_{\Omega C} + \eta \cdot \delta_C \cdot out_\Omega \qquad \text{equation C.6c}$$

$$w^+_{\lambda A} = w_{\lambda A} + \eta \cdot \delta_A \cdot out_\lambda \qquad \text{equation C.6d}$$

$$w^+_{\lambda B} = w_{\lambda B} + \eta \cdot \delta_B \cdot out_\lambda \qquad \text{equation C.6e}$$

$$w^+_{\lambda C} = w_{\lambda C} + \eta \cdot \delta_C \cdot out_\lambda \qquad \text{equation C.6f}$$

Once these calculations are completed, exactly one pattern has been passed forward through the network to find its outputs, and using the error calculation this pattern error has been passed back through the network to calculate the training. After this has been done for all patterns an epoch has occurred.

C.3 SLT Delta Rule

The delta rule for a Single-Layer Taylor Series network must take into account the multiple weights for each connection and their associated factorial and power terms.

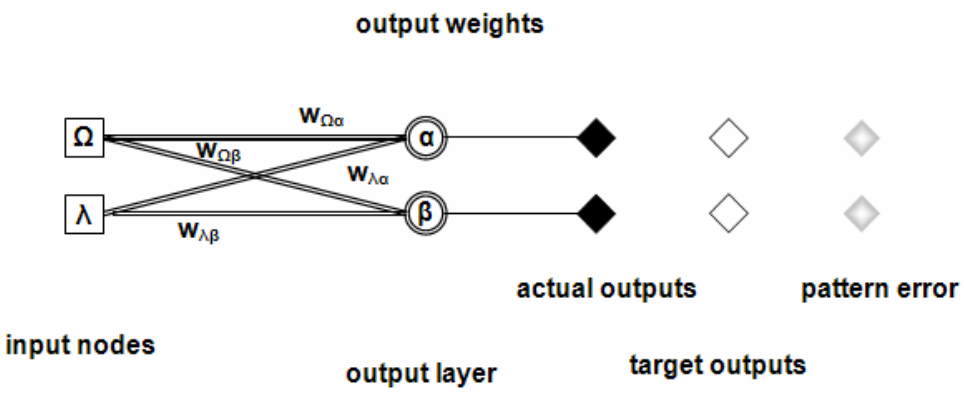


Figure C.2 – Delta Rule of an SLT

The Delta Rule is only concerned with training an output layer. The rule used is based on the version implemented in Backpropagation and proceeds as follows;

- For each pattern in turn, the output neurons' errors e_n are calculated. For the output neurons;

$$e_\alpha = \left(t\arg et_\alpha - out_\alpha\right) \qquad \text{equation C.7a}$$

$$e_\beta = \left(t\arg et_\beta - out_\beta\right) \qquad \text{equation C.7b}$$

This is the same as for a McCulloch-Pitts neuron. This is a property of the neuron, related to its output, and as such it is not affected by the connections to the neuron.

- The errors e_n are used to calculate the deltas δ_n for the output neurons - this is;

$$\delta_\alpha = out_\alpha \cdot \left(1 - out_\alpha\right) \cdot e_\alpha \qquad \text{equation C.8a}$$

$$\delta_\beta = out_\beta \cdot \left(1 - out_\beta\right) \cdot e_\beta \qquad \text{equation C.8b}$$

Once more as a property of the neuron this is the same as in the MP neuron.

The deltas are used to calculate the new values of each connecting weight using the Delta Rule. This is different to the MP neuron as there are now multiple weights connecting each neuron and node. Weight values are designated w_{ijp} where i is the connecting unit in the previous layer, j is the current neuron and p is the order of power the weight is connected via.

- For the above network, assuming a 2rd order implementation, the calculations are as follows;

$$w^{+}_{\Omega\alpha 1} = w_{\Omega\alpha 1} + \eta \cdot \delta_{\alpha} \cdot \frac{out_{\Omega}{}^{1}}{1!} \qquad \text{equation C.9a}$$

$$w^{+}_{\Omega\alpha 2} = w_{\Omega\alpha 2} + \eta \cdot \delta_{\alpha} \cdot \frac{out_{\Omega}{}^{2}}{2!} \qquad \text{equation C.9b}$$

$$w^{+}_{\Omega\beta 1} = w_{\Omega\beta 1} + \eta \cdot \delta_{\beta} \cdot \frac{out_{\Omega}{}^{1}}{1!} \qquad \text{equation C.9c}$$

$$w^{+}_{\Omega\beta 2} = w_{\Omega\beta 2} + \eta \cdot \delta_{\beta} \cdot \frac{out_{\Omega}{}^{2}}{2!} \qquad \text{equation C.9d}$$

$$w^{+}_{\lambda\alpha 1} = w_{\lambda\alpha 1} + \eta \cdot \delta_{\alpha} \cdot \frac{out_{\lambda}{}^{1}}{1!} \qquad \text{equation C.9e}$$

$$w^{+}_{\lambda\alpha 2} = w_{\lambda\alpha 2} + \eta \cdot \delta_{\alpha} \cdot \frac{out_{\lambda}{}^{2}}{2!} \qquad \text{equation C.9f}$$

$$w^{+}_{\lambda\beta 1} = w_{\lambda\beta 1} + \eta \cdot \delta_{\beta} \cdot \frac{out_{\lambda}{}^{1}}{1!} \qquad \text{equation C.9g}$$

$$w^{+}_{\lambda\beta 2} = w_{\lambda\beta 2} + \eta \cdot \delta_{\beta} \cdot \frac{out_{\lambda}{}^{2}}{2!} \qquad \text{equation C.9h}$$

- For any weight designated w_{ijp}, the new weight can be calculated as follows;

$$w^{+}_{ijp} = w_{ijp} + \eta \cdot \delta_{j} \cdot \frac{out_{i}{}^{p}}{p!} \qquad \text{equation C.10b}$$

Once completed for every pattern, an epoch has occurred.

C.4 Pulse-Width Backpropagation

For an ABN_w the Backpropagation Algorithm must consider the time-domain parameters that accumulate the Sum value for each node. The algorithm and its derivation was presented in Chapter 8 with future considerations presented in Chapter 9. Given an ABN of equivalent topology to the ANN shown in figure C.1, the algorithm is summarised for nodes α and *A*, all other nodes in the same layers having equivalent operation.

- The output node error e_n is calculated for time t_i;

$$e_{\alpha(i)} = \left(t\arg et_{\alpha} - out_{\alpha(i)}\right) \qquad \text{equation C.11a}$$

This is the same calculation as used for an Artificial Neuron once output at t_i is evaluated.

- The associated delta term is calculated;

$$\delta_{\alpha} = out_{\alpha} \cdot \left(1 - out_{\alpha(i)}\right) \cdot e_{\alpha(i)} \qquad \text{equation C.12a}$$

This is also the same as for an AN.

- The delta values are used to calculated the new signal-pathway strengths s^+.

$$s^{+}_{A\alpha} = s_{A\alpha} + \eta \cdot \delta_{\alpha} \cdot out_{A(i-1)} \qquad \text{equation C.13a}$$

$$s^{+}_{B\alpha} = s_{B\alpha} + \eta \cdot \delta_{\alpha} \cdot out_{\mathbf{B}(i-1)} \qquad \text{equation C.13b}$$

$$s^{+}_{C\alpha} = s_{C\alpha} + \eta \cdot \delta_{\alpha} \cdot out_{C(i-1)} \qquad \text{equation C.13c}$$

These calculations take into consideration that the most recent output from the previous layer is not the correct one to use, as it was in an AN. Instead the output at the previous time $t_{(i-1)}$ must be used.

At this point the output layer has been trained and the credit assignment must be made for the hidden nodes.

- The delta for the hidden layer node *A* is;

$$\delta_A = out_{A(n-1)} \cdot \left(1 - out_{A(n-1)}\right) \cdot \left(\delta_\alpha \cdot w_{A\alpha} + \delta_\beta \cdot w_{A\beta}\right) \quad \text{equation C.14a}$$

This delta value is used to calculate the strength of the hidden signal-pathways. In this example the outputs from the input nodes remain constant while the data pattern is presented. Therefore there is no need to calculate a pulse at $t_{(i-2)}$.

- The new hidden signal-pathway strengths s^+ are;

$$s^+_{\Omega A} = s_{\Omega A} + \eta \cdot \delta_A \cdot out_\Omega \quad \text{equation C.15a}$$

$$s^+_{\lambda A} = s_{\lambda A} + \eta \cdot \delta_A \cdot out_\lambda \quad \text{equation C.15b}$$

If the previous layer was also a layer of hidden node rather than input nodes then time-domain again becomes important.

- In this case the calculations are;

$$s^+_{\Omega A} = s_{\Omega A} + \eta \cdot \delta_A \cdot out_{\Omega(i-2)} \quad \text{equation C.16a}$$

$$s^+_{\lambda A} = s_{\lambda A} + \eta \cdot \delta_A \cdot out_{\lambda(i-2)} \quad \text{equation C.16b}$$

Once completed, the ABN has relaxed for one pattern and had its signal-pathways adjusted. After this has been done for all patterns an epoch has occurred.

Appendix D
Polynomial Over-Fitting

D.1 Introduction to the Appendix

Networks composed of polynomial-type neurons have remarkable pattern recognition abilities. These abilities may be due to their capacity to follow a decision boundary contour far more accurately than McCulloch-Pitts neurons Duch and Jankowski, [1999]. This is attributed by Giles and Maxwell, [1987] to their modelling of the high-order structure of the environment in which they operate. These authors also give a useful and detailed definition of generalisation in neural networks.

D.2 Polynomial Over-Fitting

A network composed of linear separators requires an infinite number of them to exactly map any smooth curve, whereas a polynomial may be able to follow it perfectly. Herein lies both a functional advantage and a potential weakness.

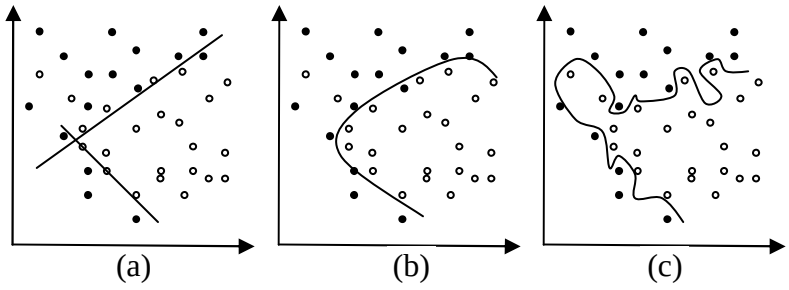


Figure D.1 – Polynomial over-fitting
modified from [Bishop 1995]

Figure D.1(a) shows an attempt by a singe linear separator to distinguish between the classes of open and closed circles. This results in many errors in the training-set. Figure D.1(b) shows a single polynomial separator. This approximates the decision boundary much better and accounts for fewer errors. If the single polynomial separator is allowed to increase its order until it reaches a zero error, (see Figure D.1(c)), it can classify all the training-set correctly. This shows the ability of the polynomial compared to the linear-

separator. It may be more apparent if the linear separator is viewed as a polynomial of first order terms Capanni et al., [2003].

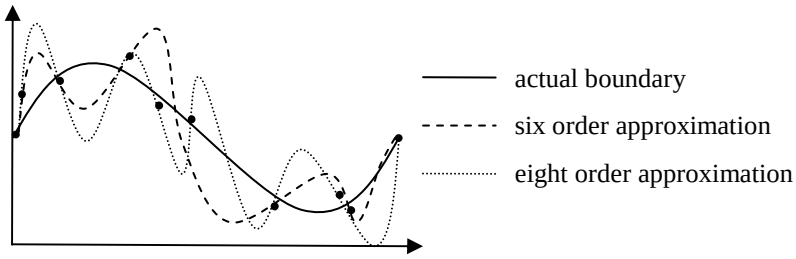


Figure D.2 – Polynomial over-fitting of smooth curve

A single polynomial is used to approximate a decision boundary such as shown in figure D.2, and allowed to increase in order to improve its fit. This may then become prone to over-fitting as it attempts to intersect with every training-data point and makes abrupt changes to do so. In the example shown, the decision boundary is closely approximated by a 2^{nd} order polynomial. Then the order of the polynomial is increased, shown as 6^{th} order, until there is an exact mapping, 8^{th} order. The neuron does exactly what it is asked to do by the training algorithm and finds an exact solution to the training-set. The extreme differences in the decision boundary show the inherent danger of over-fitting through the pattern matching abilities of polynomial neurons.

There are many approaches to avoid over-fitting with higher-order networks. One method is to evolve the order of the polynomial with an Evolutionary Algorithm while training the EA's parameters with another approach, such as a separate EA or a derived gradient descent algorithm. If the fitness takes account of over-fitting then a fitness function with a weighting factor can be used [Kim and Park, 2003]. Methods can be derived from linear separation training to utilise error feedback, such as Backpropagation Nikolaev [2003]. In a known problem, where there is no set formula for determining the size of the hidden layer in a MLP, these methods can be extended to polynomial networks. If the network size is incorrectly set and the algorithm does not allow the network to alter its size, then over-fitting or under-fitting will occur Chang and Cheung, [1992].

It is a prime requirement of networks to have good generalisation, that is to be tolerant of noise in the training patterns. Therefore the network must either be very problem-specific or be flexible enough to include generalisation within its training algorithm.

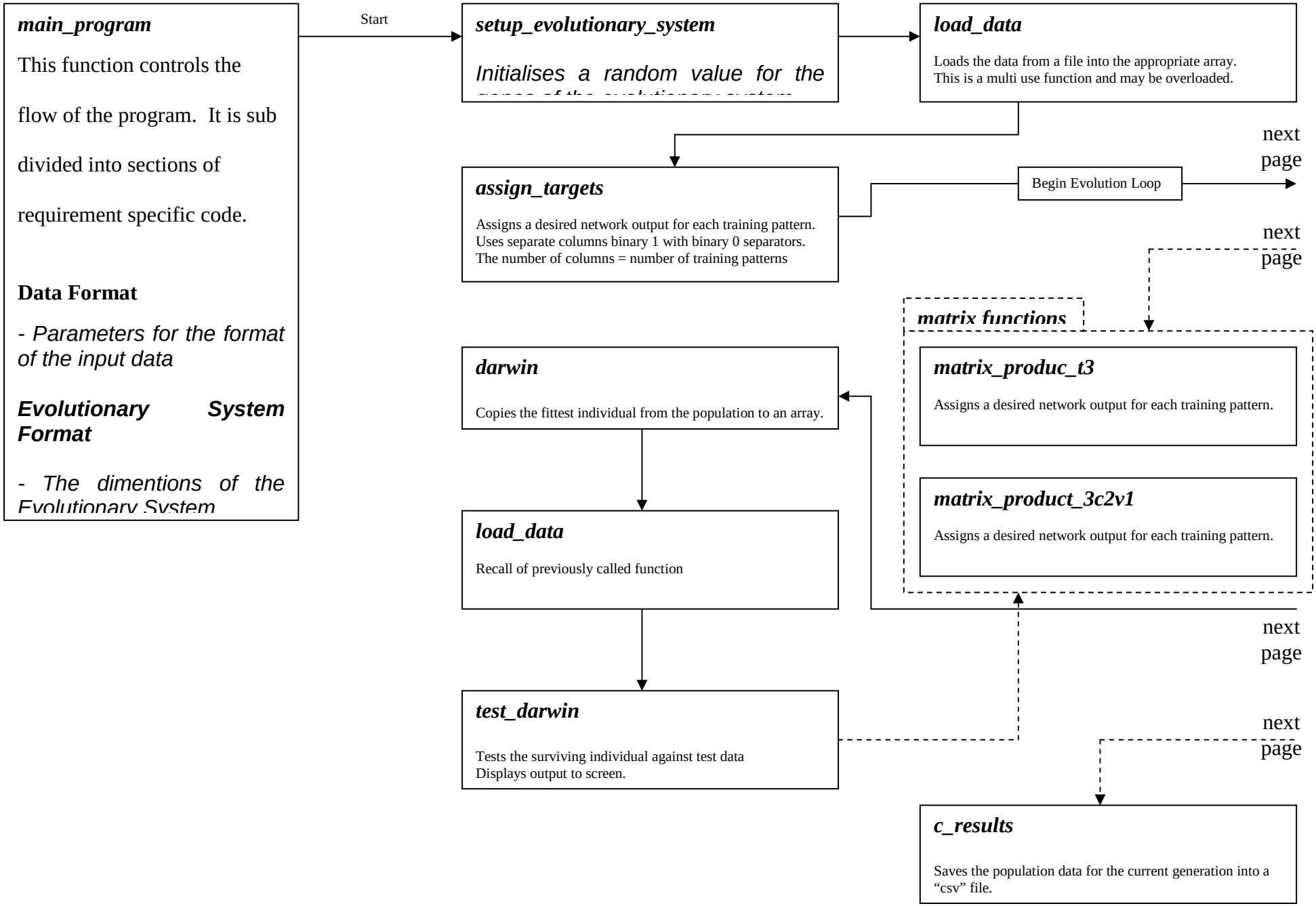
Appendix E
Methods

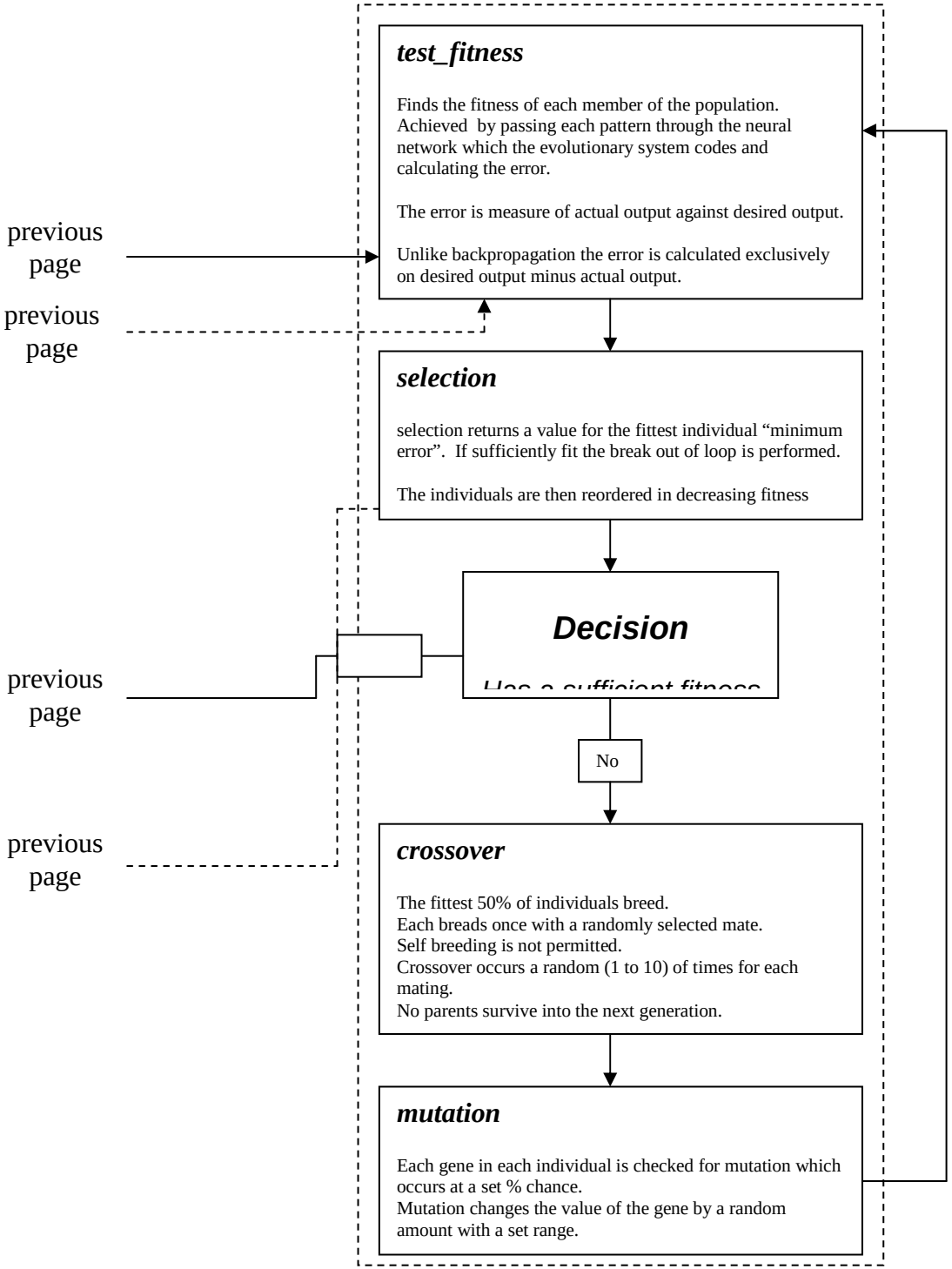
E.1 Introduction to the Appendix

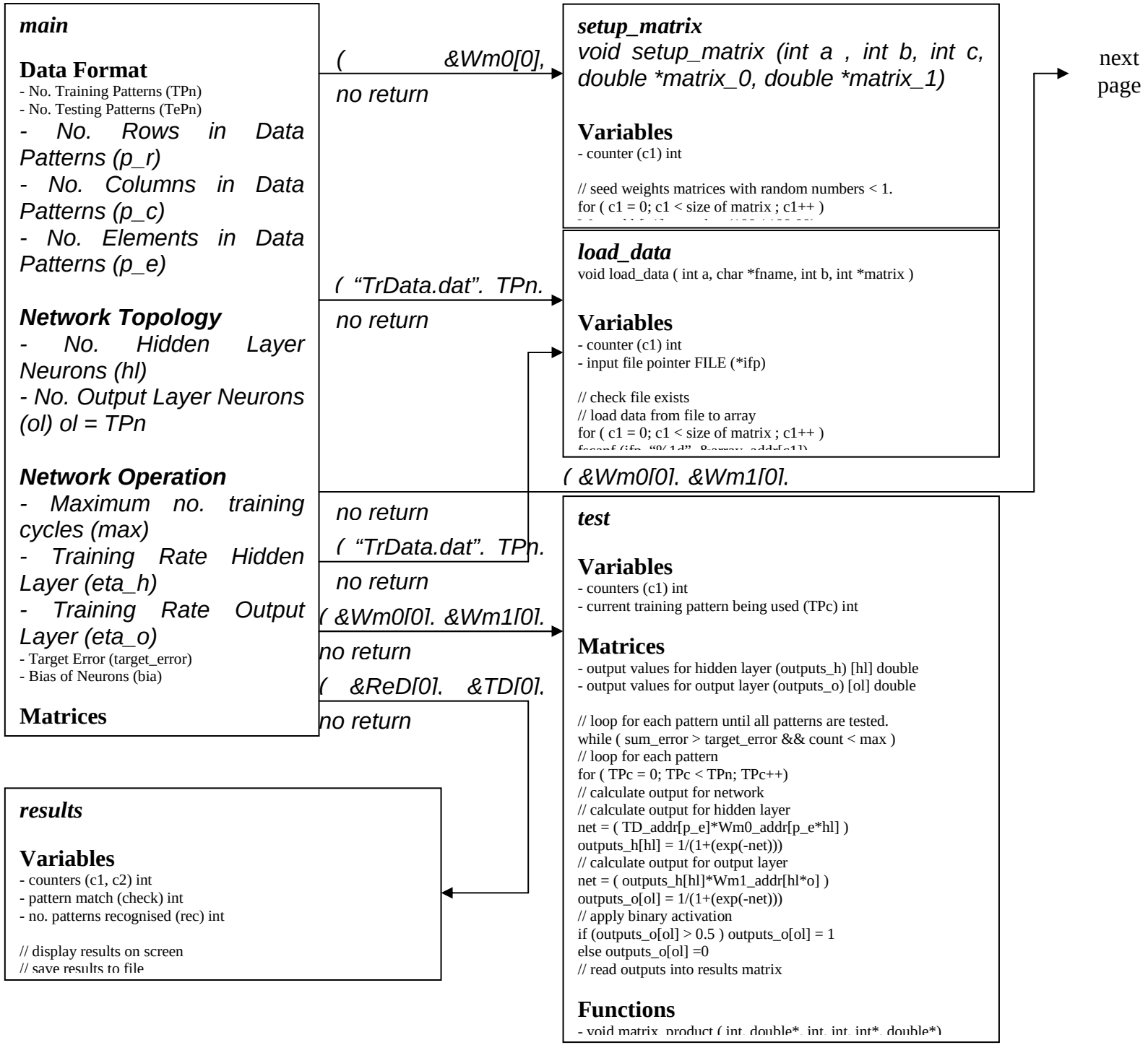
The experiments in this thesis used a variety of software. The main software used was Borland C++ version 5.02. Minor simulations were assessed through Visual Basic (VB) within Microsoft Excel 2003. Additional visual functionality was achieved with Microsoft HTA interfaces which used HTML, CSS, JavaScript and VB. Visualisation was achieved through Mathcad 11 Enterprise Edition and user- constructed Graphical Interfaces in DHTML.

The algorithms used or derived have been supplied appropriately in Chapters 5 and 8 and in Appendix C.

All the different systems required individual programmes, however the inclusion of all of these would not benefit the thesis. As an example, supplied below are the flow charts for the programme design of an Evolutionary Algorithm used to evolve the connectionist networks.


main_program
This function controls the flow of the program. It is sub divided into sections of requirement specific code.
Data Format
- Parameters for the format of the input data
Evolutionary System Format
- The dimentions of the Evolutionary System
Start
setup_evolutionary_system
Initialises a random value for the
load_data
Loads the data from a file into the appropriate array.
This is a multi use function and may be overloaded.
next page
assign_targets
Assigns a desired network output for each training pattern.
Uses separate columns binary 1 with binary 0 separators.
The number of columns = number of training patterns
Begin Evolution Loop
next page
matrix functions
darwin
Copies the fittest individual from the population to an array.
matrix_produc_t3
Assigns a desired network output for each training pattern.
matrix_product_3c2v1
Assigns a desired network output for each training pattern.
load_data
Recall of previously called function
next page
test_darwin
Tests the surviving individual against test data
Displays output to screen.
next page
c_results
Saves the population data for the current generation into a "csv" file.


previous page
previous page
previous page
previous page
test_fitness
Finds the fitness of each member of the population. Achieved by passing each pattern through the neural network which the evolutionary system codes and calculating the error.
The error is measure of actual output against desired output.
Unlike backpropagation the error is calculated exclusively on desired output minus actual output.
selection
selection returns a value for the fittest individual “minimum error”. If sufficiently fit the break out of loop is performed.
The individuals are then reordered in decreasing fitness
Decision
No
crossover
The fittest 50% of individuals breed.
Each breads once with a randomly selected mate.
Self breeding is not permitted.
Crossover occurs a random (1 to 10) of times for each mating.
No parents survive into the next generation.
mutation
Each gene in each individual is checked for mutation which occurs at a set % chance.
Mutation changes the value of the gene by a random amount with a set range.


main
Data Format
- No. Training Patterns (TPn)
- No. Testing Patterns (TePn)
- No. Rows in Data Patterns (p_r)
- No. Columns in Data Patterns (p_c)
- No. Elements in Data Patterns (p_e)
Network Topology
- No. Hidden Layer Neurons (hl)
- No. Output Layer Neurons (ol) ol = TPn
Network Operation
- Maximum no. training cycles (max)
- Training Rate Hidden Layer (eta_h)
- Training Rate Output Layer (eta_o)
- Target Error (target_error)
- Bias of Neurons (bia)
Matrices
(&Wm0[0],
no return
setup_matrix
void setup_matrix (int a , int b, int c, double *matrix_0, double *matrix_1)
Variables
- counter (c1) int
// seed weights matrices with random numbers < 1.
for (c1 = 0; c1 < size of matrix ; c1++)
next page
("TrData.dat". TPn.
no return
load_data
void load_data (int a, char *fname, int b, int *matrix)
Variables
- counter (c1) int
- input file pointer FILE (*ifp)
// check file exists
// load data from file to array
for (c1 = 0; c1 < size of matrix ; c1++)
(&Wm0[0]. &Wm1[0].
no return
("TrData.dat". TPn.
no return
(&Wm0[0]. &Wm1[0].
no return
(&ReD[0]. &TD[0].
no return
test
Variables
- counters (c1) int
- current training pattern being used (TPc) int
Matrices
- output values for hidden layer (outputs_h) [hl] double
- output values for output layer (outputs_o) [ol] double
// loop for each pattern until all patterns are tested.
while (sum_error > target_error && count < max)
// loop for each pattern
for (TPc = 0; TPc < TPn; TPc++)
// calculate output for network
// calculate output for hidden layer
net = (TD_addr[p_e]*Wm0_addr[p_e*hl])
outputs_h[hl] = 1/(1+(exp(-net)))
// calculate output for output layer
net = (outputs_h[hl]*Wm1_addr[hl*o])
outputs_o[ol] = 1/(1+(exp(-net)))
// apply binary activation
if (outputs_o[ol] > 0.5) outputs_o[ol] = 1
else outputs_o[ol] =0
// read outputs into results matrix
Functions
results
Variables
- counters (c1, c2) int
- pattern match (check) int
- no. patterns recognised (rec) int
// display results on screen
// save results to file

matrix_product_3c2v1

void matrix_product (double *Wm0_addr, double *Wm1_addr, int *TD_addr)

Variables

- counters (c1, c2) int
- counter, no. training cycles (count_cycles) int
- current training pattern being used (TPc) int
- sum of errors for all patterns (sum_error = 1.0) double
- combined error for propagating error to hidden layer (c_error = 0.0) double
- sum of weights*inputs for each neuron (net) double

Matrices

- output values for hidden layer (outputs_h) [hl] double
- output values for output layer (outputs_o) [ol] double
- error values for hidden layer (delta_h) [hl] double
- error values for output layer (delta_o) [ol] double
- change in weights for hidden layer (DELTA_h) [p_e*hl] double
- change in weights for output layer (DELTA_o) [hl*ol] double

previous page

matrix_product_3

void matrix_product (double *Wm0_addr, double *Wm1_addr, int *TD_addr)

Variables

- counters (c1, c2) int
- counter, no. training cycles (count_cycles) int
- current training pattern being used (TPc) int
- sum of errors for all patterns (sum_error = 1.0) double
- combined error for propagating error to hidden layer (c_error = 0.0) double
- sum of weights*inputs for each neuron (net) double

Matrices

- output values for hidden layer (outputs_h) [hl] double
- output values for output layer (outputs_o) [ol] double
- error values for hidden layer (delta_h) [hl] double
- error values for output layer (delta_o) [ol] double
- change in weights for hidden layer (DELTA_h) [p_e*hl] double

train

void train (double *Wm0_addr, double *Wm1_addr, int *TD_addr)

Variables

- counters (c1, c2) int
- counter, no. training cycles (count_cycles) int
- current training pattern being used (TPc) int
- sum of errors for all patterns (sum_error = 1.0) double
- combined error for propagating error to hidden layer (c_error = 0.0) double

Matrices

- output values for hidden layer (outputs_h) [hl] double
- output values for output layer (outputs_o) [ol] double
- error values for hidden layer (delta_h) [hl] double
- error values for output layer (delta_o) [ol] double
- change in weights for hidden layer (DELTA_h) [p_e*hl] double
- change in weights for output layer (DELTA_o) [hl*ol] double

```
// loop for each pattern until error has reached target or maximum cycles have occurred.
while ( sum_error > target_error && count < max )
// loop for each pattern
for ( TPc = 0; TPc < TPn; TPc++)
// calculate output for network
// calculate output for hidden layer
matrix_product_3c2v1( 1, p_e, hl, TPc, &outputs_h[0], &TD_addr[0], &Wm0_addr[0] )
outputs_h[hl] = 1/(1+(exp(-outputs_h[hl])))
// calculate output for output layer
matrix_product_3( 1, hl, ol, &outputs_o[0], &outputs_h[0], &Wm1_addr[0] )
outputs_o[ol] = 1/(1+(exp(-outputs_o[ol])))
// calculate error for output layer
delta_o[o1] = outputs_o[o1]*(1-outputs_o[o1])*(Target – Output)
// calculate sum_error for network
sum_error += delta_o[o1]
// calculate change in weights for output layer
DELTA_o[hl*ol] = eta_o*delta_o[ol]*outputs_h[hl]
// calculate new weights for output layer
Wm1_addr[hl*ol] += DELTA_o[hl*ol]
// calculate error for hidden layer
c_error =+ ( delta_o[ol]*Wm1_addr[hl*ol] )
delta_h[hl] = outputs_h[hl]*(1-outputs_h[hl]*c_error)
// calculate change in weights for hidden layer
DELTA_h[p_e*hl] = eta_h*delta_h[hl]*TD_addr[p_e]
// calculate new weights for hidden layer
Wm0_addr[p_e*hl] += DELTA_h[p_e*hl]
```

Functions

- void matrix_product_3 (int, double*, int, int, int*, double*)
- void matrix_product_3c2v1(int, double*, int, int, double*, double*)

Appendix F

Taylor Series Neuron Results

F.1 Introduction to the Appendix

Additional results and enlarged figures from chapter 5 are included in this appendix for fullness and clarification. Each is placed under the title of the section which they relate too.

From 5.2.1 Output Functions

The linear and hyperbolic tangent output functions are illustrated below.

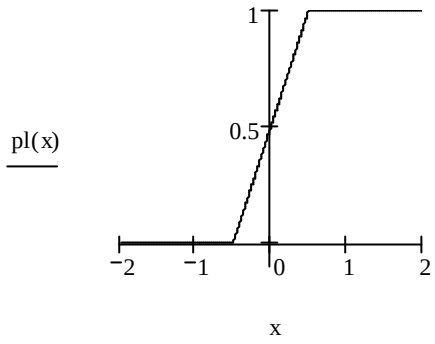


Figure F.1 – Piecewise linear function *pl*(x)

$$pl(x) = \begin{cases} 1 & \text{if } x \geq 0.5 \\ x & \text{if } -0.5 < x < 0.5 \\ 0 & \text{if } x \leq -0.5 \end{cases}$$

equation F.1

A linear function is termed “piecewise linear” when the output is constrained to linear operation within a region. In the example shown in figure F.1 and equation F.1, the function is linear in the region (-0.5,0.5). Outside this region, it operates a threshold function. Without the linear region, the function collapses to a threshold function.

This function is useful for directly reflecting the input values while preventing saturation when very large values occur.

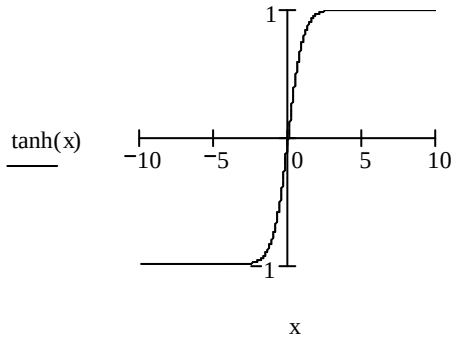


Figure F.2 – Hyperbolic tangent function *tanh*(x)

$$\tanh(x) = \frac{e^{(x)} - e^{-(x)}}{e^{(x)} + e^{-(x)}} \qquad \text{equation F.2}$$

The hyperbolic tangent is probably the second most common function. It is similar to the logistic function with the range anti-symmetrical about the origin. The function, shown in figure F.2 and equation F.2, has a range [-1,1]. There are some advantages to this which are associated with training parameters. The specifics of these are reasonably well known, [Haykin, 1999], and are not discussed further here.

The last two functions to consider are the step-logistic and step-hyperbolic tangent functions. These take a threshold on the functions shown in figure 5.2 (logistic sigmoid function *l*(x)) and figure F.2 (hyperbolic tangent function *tanh*(x)). The output becomes that of figure 5.1, threshold *t*(x), for the sets {0,1} and {-1,1} respectively.

These functions are not usually considered as part of the training process of any network. They are only of use if a decision output is required from a network and decimal values provide more information than needed. These are rounded to the nearest integer output from the relevant set.

From 5.3.1 Taylor Series Neuron Output Functions

The equations for the logistic hyperbolic tangent output function shown below, equation F3. The range of the output is [-1,+1]. This gives output values in equation F.4 for MP and equation F.5 for TS.

$$Output = \frac{e^{(Sum)} - e^{-(Sum)}}{e^{(Sum)} + e^{-(Sum)}}$$ equation F.3

$$O = f(S) = \frac{e^{\left(\theta + \sum_{i=1}^{2} x_i w_i\right)} - e^{-\left(\theta + \sum_{i=1}^{2} x_i w_i\right)}}{e^{\left(\theta + \sum_{i=1}^{2} x_i w_i\right)} + e^{-\left(\theta + \sum_{i=1}^{2} x_i w_i\right)}}$$ equation F.4a

$$O = \frac{e^{(\theta + x_1 w_1 + x_2 w_2)} - e^{-(\theta + x_1 w_1 + x_2 w_2)}}{e^{(\theta + x_1 w_1 + x_2 w_2)} + e^{-(\theta + x_1 w_1 + x_2 w_2)}}$$ equation F.4b

$$O = f(S) = \frac{e^{\left(\theta + \sum_{p=1}^{m} \sum_{i=1}^{2} x_i^p \frac{w_{i,p}}{p!}\right)} - e^{-\left(\theta + \sum_{p=1}^{m} \sum_{i=1}^{2} x_i^p \frac{w_{i,p}}{p!}\right)}}{e^{\left(\theta + \sum_{p=1}^{m} \sum_{i=1}^{2} x_i^p \frac{w_{i,p}}{p!}\right)} + e^{-\left(\theta + \sum_{p=1}^{m} \sum_{i=1}^{2} x_i^p \frac{w_{i,p}}{p!}\right)}}$$ equation F.5a

$$O = \frac{e^{\left(\theta + \sum_{p=1}^{m} \left(x_1^p \frac{w_{1,p}}{p!} + x_2^p \frac{w_{2,p}}{p!} \right)\right)} - e^{-\left(\theta + \sum_{p=1}^{m} \left(x_1^p \frac{w_{1,p}}{p!} + x_2^p \frac{w_{2,p}}{p!} \right)\right)}}{e^{\left(\theta + \sum_{p=1}^{m} \left(x_1^p \frac{w_{1,p}}{p!} + x_2^p \frac{w_{2,p}}{p!} \right)\right)} + e^{-\left(\theta + \sum_{p=1}^{m} \left(x_1^p \frac{w_{1,p}}{p!} + x_2^p \frac{w_{2,p}}{p!} \right)\right)}}$$ equation F.5b

From 5.4 Testing : Single Neuron Functionality

This section visualises the various separator functions in three dimensions. The x-axis and y-axis represent the inputs (x_1,x_2) while the z-axis shows the output value.

From 5.4.1 McCulloch-Pitts Functions

The first simple operator is the piecewise linear separator as shown in equation F.1. The output range is constrained in [0,1] as shown in equation F.6 and limits any values from the input range of sum ≥ 1 or sum ≤ 0. The obvious drawback of this strategy is that a sum can greatly exceed these values, resulting in an output value that does not directly equate to the sum. This may result in saturation of network parameters with no direct proportional effect on the output.

$$O = \begin{cases} 1 & \text{if } S \geq 1.0 \\ S & \text{if } S > 0.0 \\ 0 & \text{if } S \leq 0.0 \end{cases}$$

equation F.6

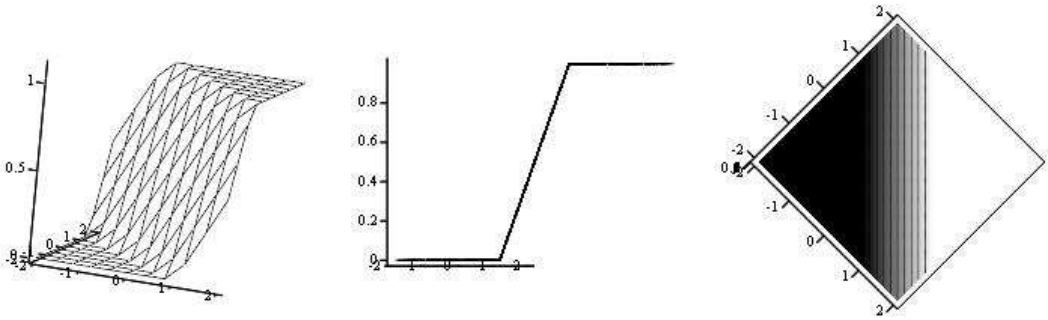


Figure F.3(a,b,c) – Piecewise linear output functions

A comparison of figure F.3a with figure 5.4a shows a similar profile with a maximum and minimum output. Figure 5.4b shows that the output separator remains linear within its range. Figure F.3c now shows the two extreme values the outputs can take with a uniform gradient connecting them.

The values assigned to the piecewise linear separator are critical and unexpected operation can result in discontinuities. Such an example is shown in equation F.7.

$$O = \begin{cases} 1 & \text{if } S \geq 0.5 \\ S & \text{if } S > -0.5 \\ 0 & \text{if } S \leq -0.5 \end{cases}$$

equation F.7

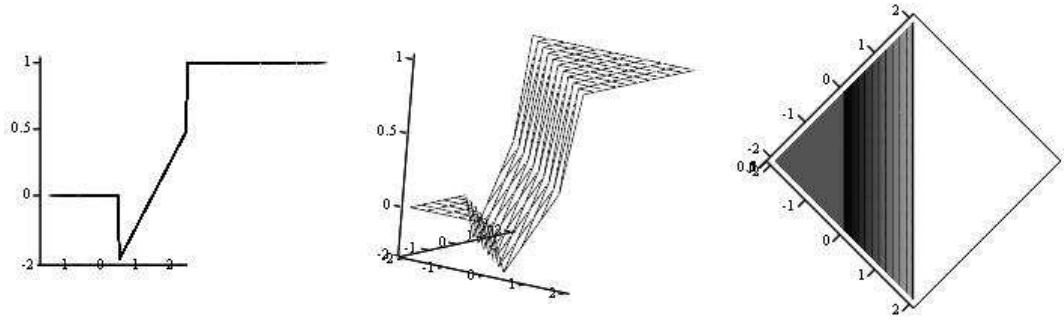


Figure F.4(a,b,c) – Piecewise linear output functions

Figures F.4 show the effect of incorrect assignment of parameters. This can be utilised if there is a purpose to it; however, the drop below the minimum value and the sudden step to the maximum value can make behaviour erratic.

When the hyperbolic tangent sigmoid is examined, it can be seen that it is very similar to the logistic sigmoid except that the output-plane is squashed in (-1,1).

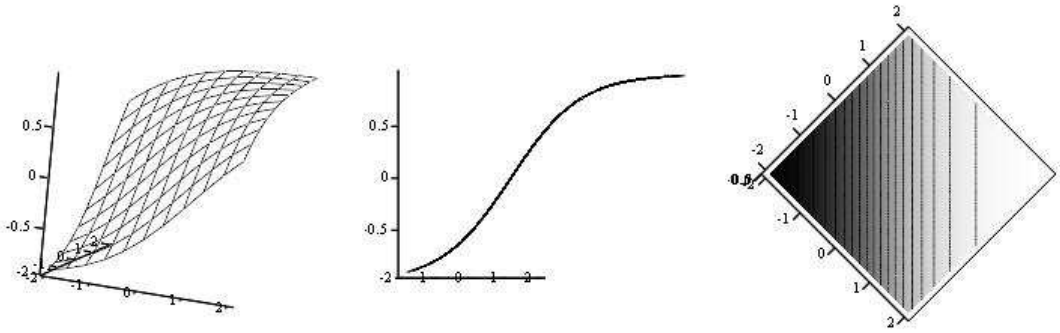


Figure F.5(a,b,c) – Hyperbolic tangent output function

The hyperbolic tangent plane in figure F.5 is visually very similar to the logistic sigmoid. Comparing equations 5.5 and F.3 shows similar derivation. In fact, the logistic sigmoid's major advantage is in calculation time, while the hyperbolic tangent's is its anti-symmetrical output-plane about the origin. This symmetry may assists in training. The mathematical implication of multiplying by numbers close to 0 in the logistic sigmoid, compared with number close to -1 in the hyperbolic tangent sigmoid, affect the rate of convergence. Its sensitivity is as adaptable as the logistic sigmoid, figures 5.7.

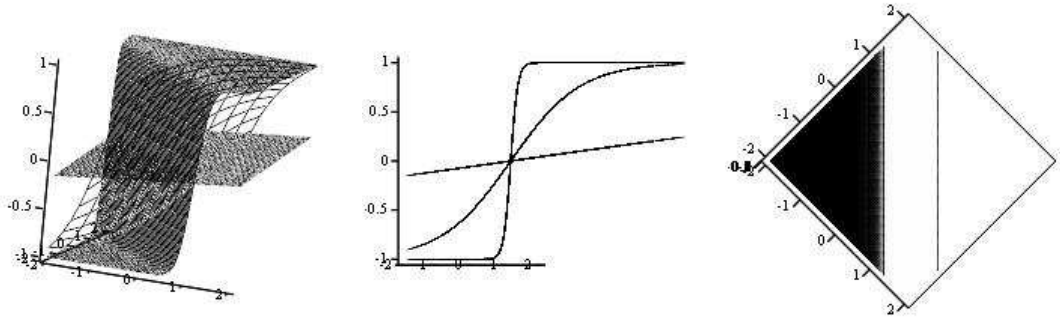


Figure F.6(a,b,c) – Hyperbolic tangent output functions with ρ

As $\rho \rightarrow 0$ or as $\rho \rightarrow \infty$, equivalent effects occur as, seen in the logistic sigmoid function.

From 5.4.2 Taylor Series Functions

The dramatic effects produced by using input values of differing polarity are examined below.

$$Sum = 0.5 \cdot x_1 + 1.5 \cdot x_2 + 0.5 \qquad \text{equation F.8}$$

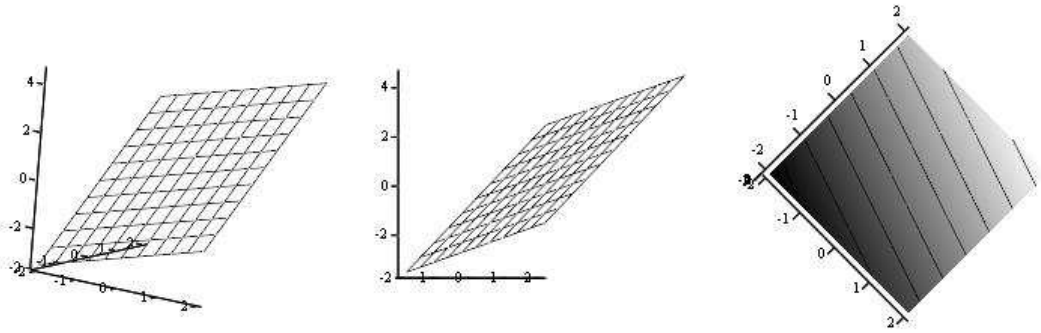


Figure F.7(a,b,c) – The Sum value expressed as a skewed function of inputs

The effect of skewing the inputs in the 1st order neuron (McCulloch-Pitts) is simply to tilt the flat plane towards the input-axis with the lower coefficient. This can be seen in equation F.8 and as a comparison between figures 5.4 and figures F.7.

$$Sum = 0.5 + 0.05 \cdot x_1 + 1.05 \cdot x_2 + \left[\frac{1.0 \cdot x_1^2 + 1.0 \cdot x_2^2}{2} \right] \qquad \text{equation F.9}$$

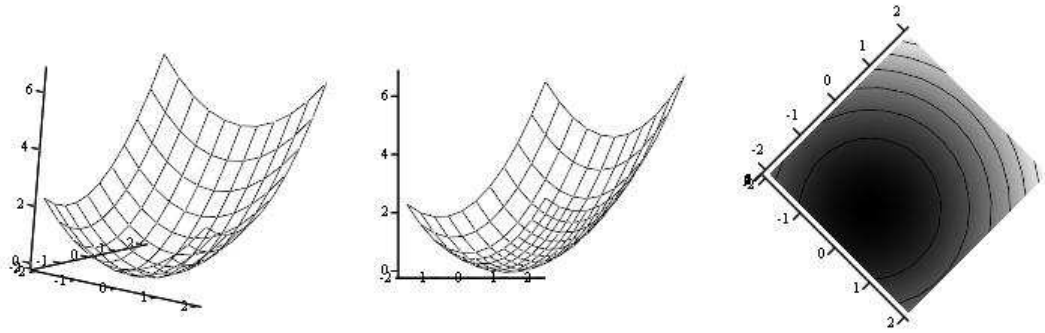


Figure F.8(a,b,c) – Sum value of 2nd order Taylor Series neuron focusing on decision region – skewed on 1st order

In equation F.9 the 1st order terms are skewed, while the coefficients of the 2nd order terms remain equal, the equivalent of what occurred in figures F.7. The curved plane of figures 5.10 is tilted in the same manner to give the figures F.8 and results in moving the output-domain decision centre towards the input-axis with the lower coefficient.

$$Sum = 0.5 + 0.05 \cdot x_1 + 0.05 \cdot x_2 + \left[\frac{1.0 \cdot x_1^2 + 3.0 \cdot x_2^2}{2} \right] \qquad \text{equation F.10}$$

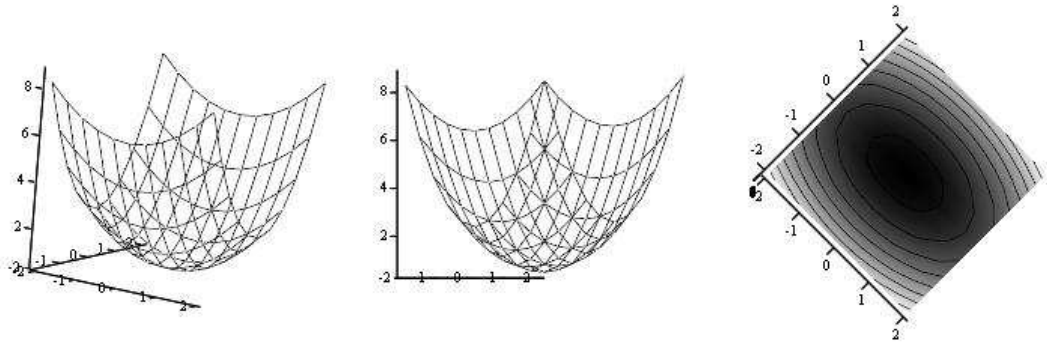


Figure F.9(a,b,c) – Sum value of 2nd order Taylor Series neuron focusing on decision region – skewed on 2nd order

Skewing 2nd order terms while keeping equal 1st order term coefficients, stretches the decision surface along the lower coefficient input-axis, shown by figures 5.10c to F.9c.

$$Sum = 0.5 + 0.05 \cdot x_1 + 1.05 \cdot x_2 + \left[\frac{1.0 \cdot x_1^2 + 3.0 \cdot x_2^2}{2} \right] \qquad \text{equation F.11}$$

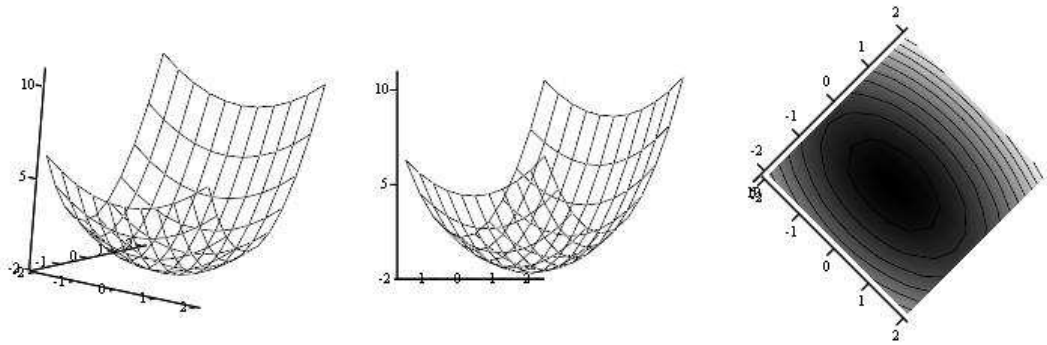


Figure F.10(a,b,c) – Sum value of 2nd order Taylor Series neuron focusing on decision region – skewed on same input for 1st and 2nd order

The independence of operation of the order terms is shown through skewing the 1st or 2nd order terms on their own. The altered 1st and 2nd order terms from equation F.9 and equation F.10 are applied simultaneously to give equation F.11. A comparison of figures F.8, figures F.9 and figures F.10 shows the independence of the actions of the 1st and 2nd orders. The combination of these effects are shown in Figures F.10.

$$Sum = 0.5 + 1.05 \cdot x_1 + 0.05 \cdot x_2 + \left[\frac{1.0 \cdot x_1^2 + 3.0 \cdot x_2^2}{2} \right] \qquad \text{equation F.12}$$

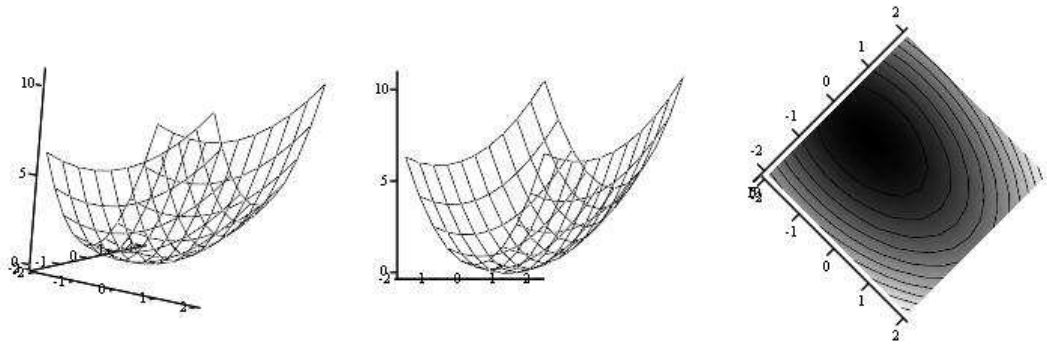


Figure F.11(a,b,c) – Sum value of 2nd order Taylor Series neuron focusing on the decision region – skewed on different inputs for 1st and 2nd order

The independence of the 1st and 2nd orders can be confirmed through figures F.11, where the effect of the 1st order shows a movement towards the other input-axis while the effect of the 2nd order remains the same. As the 1st order coefficients are swapped, but the 2nd order terms remain the same, this is consistent with what is expected.

Figures F.12 to figures F.24 examining the effects of different input domain values with added 3rd order terms. These show both the independence of the power terms and the effect of each order on the separator.

$$Sum = 0.5 + 0.05 \cdot x_1 + 1.05 \cdot x_2 + \frac{0.05 \cdot x_1^2 + 0.05 \cdot x_2^2}{2} + \left[\frac{3.0 \cdot x_1^3 + 3.0 \cdot x_2^3}{6} \right]$$

equation F.13

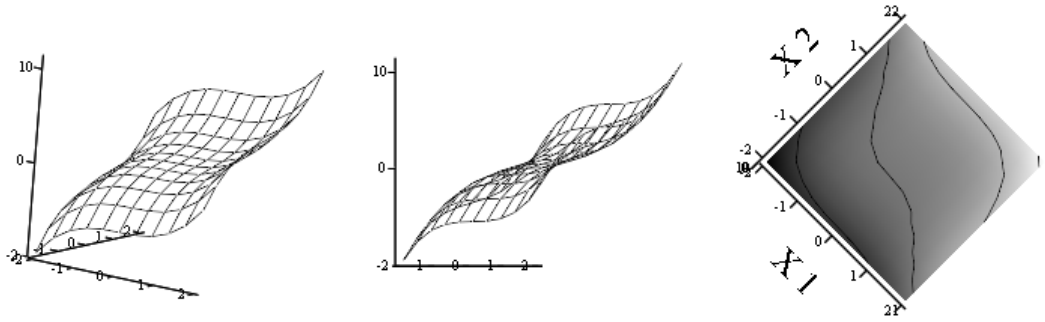


Figure F.12(a,b,c) – Sum value of 3rd order Taylor Series neuron focusing on the decision region – skewed on 1st order

In equations F.13, the 1st order terms are skewed, while the coefficients of the 2nd and 3rd order terms remain equal. As expected, the curved plane of figures 5.11 tilts towards the input-axis with the lower coefficient, in the same manner as before, to give the figures F.12.

$$Sum = 0.5 + 0.05 \cdot x_1 + 0.05 \cdot x_2 + \frac{0.05 \cdot x_1^2 + 1.05 \cdot x_2^2}{2} + \left[\frac{3.0 \cdot x_1^3 + 3.0 \cdot x_2^3}{6} \right]$$

equation F.13

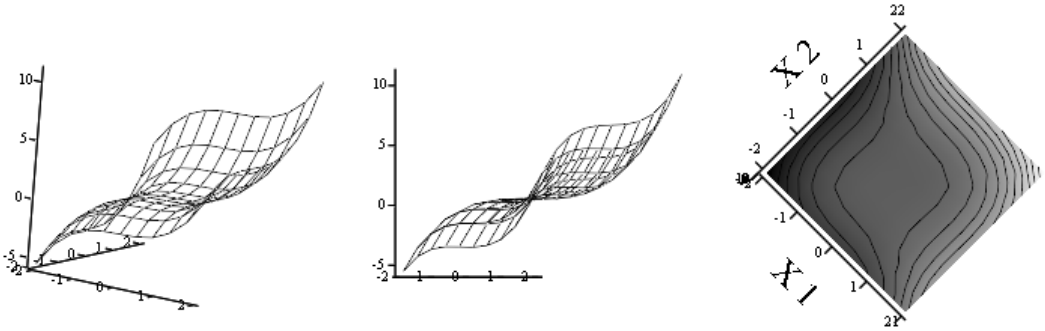


Figure F.13(a,b,c) – Sum value of 3rd order Taylor Series neuron focusing on the decision region – skewed on 2nd order

When the 2nd order terms are skewed, while the coefficients of the 1st and 3rd order terms remain equal, the decision surface is once more stretched along the input-axis with the lower coefficient.

$$Sum = 0.5 + 0.05 \cdot x_1 + 0.05 \cdot x_2 + \frac{0.05 \cdot x_1^2 + 0.05 \cdot x_2^2}{2} + \left[\frac{3.0 \cdot x_1^3 + 9.0 \cdot x_2^3}{6} \right]$$

equation F.15

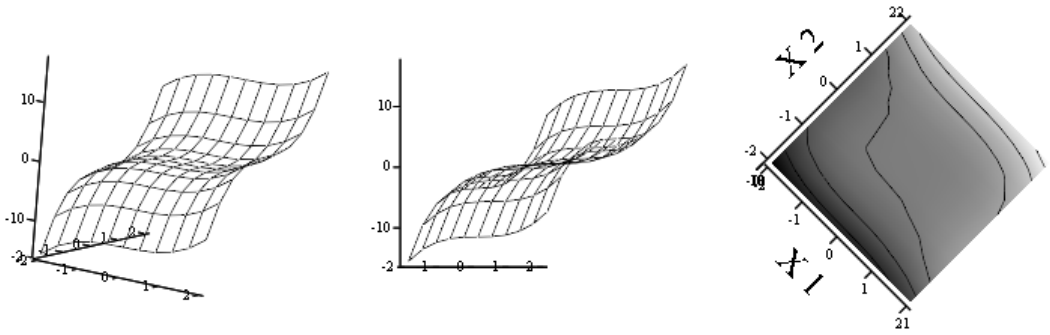


Figure F.14(a,b,c) – Sum value of 3rd order Taylor Series neuron focusing on the decision region – skewed on 3rd order

From the effect of skewing the 1st or 2nd order terms on their own, it is observable that they affect different aspects of the 3rd order decision surface in the same manner as they did for the 2nd order Taylor Series neuron. When the 3rd order is skewed the effect is to stretch the decision surface in the manner of the 2nd order; however, it is stretched with respect to the underlying 1st and 2nd order curved-plane, so the gradient of the 3rd order curve is affected as is its proximity to the axis along which it is stretched.

Equations F.16 to F.25 and their accompanying figures F.15 to F.24 show all the variation of skewing the coefficients of the orders 1st, 2nd, 3rd of powers with respect to the input terms (x_1,x_2).

$$Sum = 0.5 + 0.05 \cdot x_1 + 1.05 \cdot x_2 + \frac{0.05 \cdot x_1^2 + 1.05 \cdot x_2^2}{2} + \left[\frac{3.0 \cdot x_1^3 + 3.0 \cdot x_2^3}{6} \right]$$

equation F.16

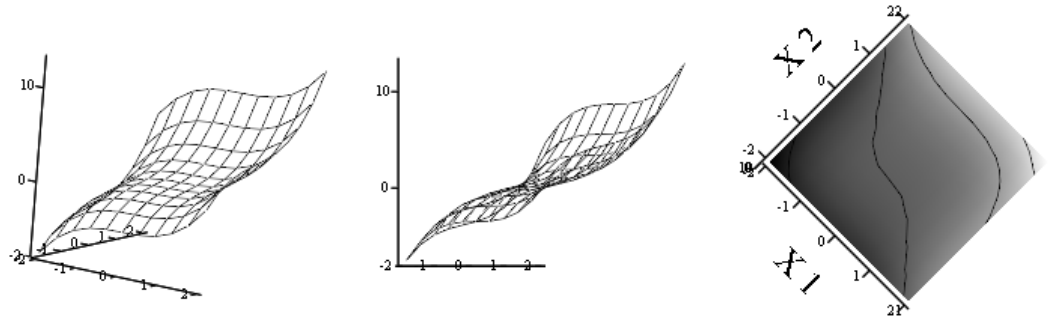


Figure F.15(a,b,c) – Sum value of 3rd order Taylor Series neuron focusing on the decision region – skewed on same inputs for 1st and 2nd order

$$Sum = 0.5 + 0.05 \cdot x_1 + 1.05 \cdot x_2 + \frac{0.05 \cdot x_1^2 + 0.05 \cdot x_2^2}{2} + \left[\frac{3.0 \cdot x_1^3 + 9.0 \cdot x_2^3}{6} \right]$$

equation F.17

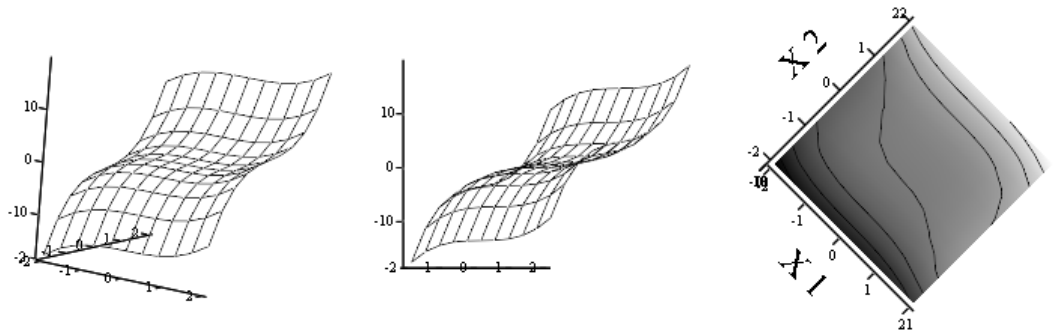


Figure F.16(a,b,c) – Sum value of 3rd order Taylor Series neuron focusing on the decision region – skewed on same inputs for 1st and 3rd order

$$Sum = 0.5 + 0.05 \cdot x_1 + 0.05 \cdot x_2 + \frac{0.05 \cdot x_1^2 + 1.05 \cdot x_2^2}{2} + \left[\frac{3.0 \cdot x_1^3 + 9.0 \cdot x_2^3}{6} \right]$$

equation F.18

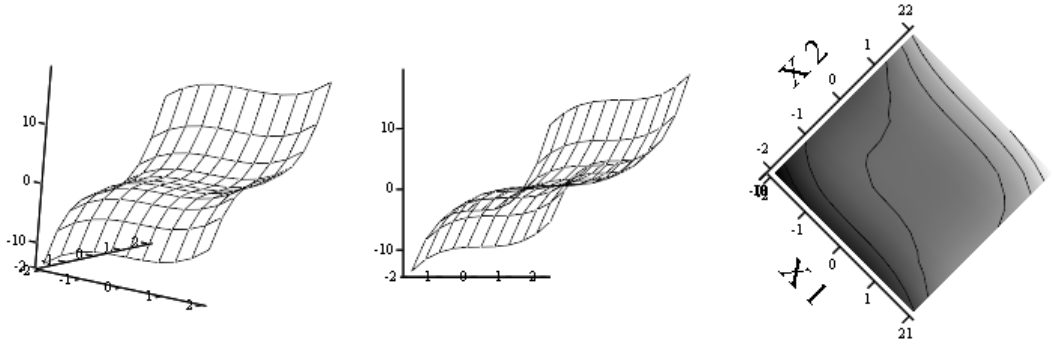


Figure F.17(a,b,c) – Sum value of 3rd order Taylor Series neuron focusing on the decision region – skewed on same inputs for 2nd and 3rd order

$$Sum = 0.5 + 0.05 \cdot x_1 + 1.05 \cdot x_2 + \frac{0.05 \cdot x_1^2 + 1.05 \cdot x_2^2}{2} + \left[\frac{3.0 \cdot x_1^3 + 9.0 \cdot x_2^3}{6} \right]$$

equation F.19

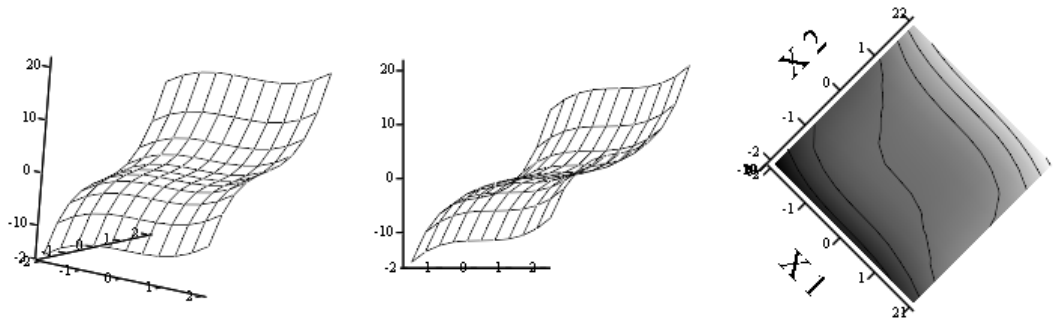


Figure F.18(a,b,c) – Sum value of 3rd order Taylor Series neuron focusing on the decision region – skewed on same inputs for 1st and 2nd and 3rd order

$$Sum = 0.5 + 0.05 \cdot x_1 + 1.05 \cdot x_2 + \frac{1.05 \cdot x_1^2 + 0.05 \cdot x_2^2}{2} + \left[\frac{3.0 \cdot x_1^3 + 3.0 \cdot x_2^3}{6} \right]$$

equation F.20

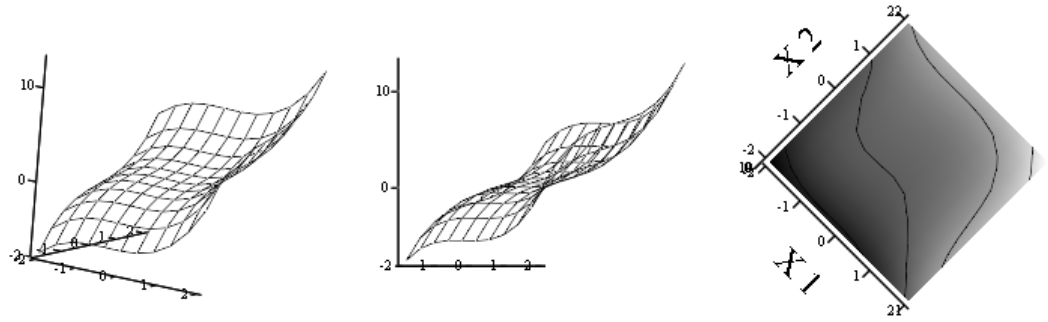


Figure F.19(a,b,c) – Sum value of 3rd order Taylor Series neuron focusing on the decision region – skewed on different inputs for 1st and 2nd order

$$Sum = 0.5 + 0.05 \cdot x_1 + 1.05 \cdot x_2 + \frac{0.05 \cdot x_1^2 + 0.05 \cdot x_2^2}{2} + \left[\frac{9.0 \cdot x_1^3 + 3.0 \cdot x_2^3}{6} \right]$$

equation F.21

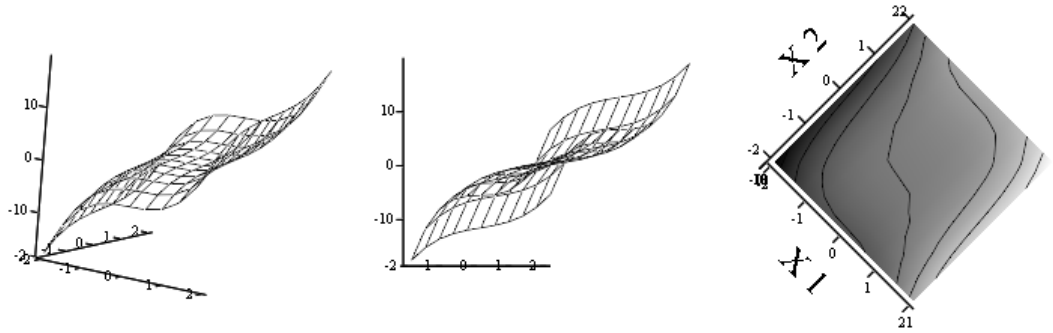


Figure F.20(a,b,c) – Sum value of 3rd order Taylor Series neuron focusing on the decision region – skewed on different inputs for 1st and 3rd order

$$Sum = 0.5 + 0.05 \cdot x_1 + 0.05 \cdot x_2 + \frac{0.05 \cdot x_1^2 + 1.05 \cdot x_2^2}{2} + \left[\frac{9.0 \cdot x_1^3 + 3.0 \cdot x_2^3}{6} \right]$$

equation F.22

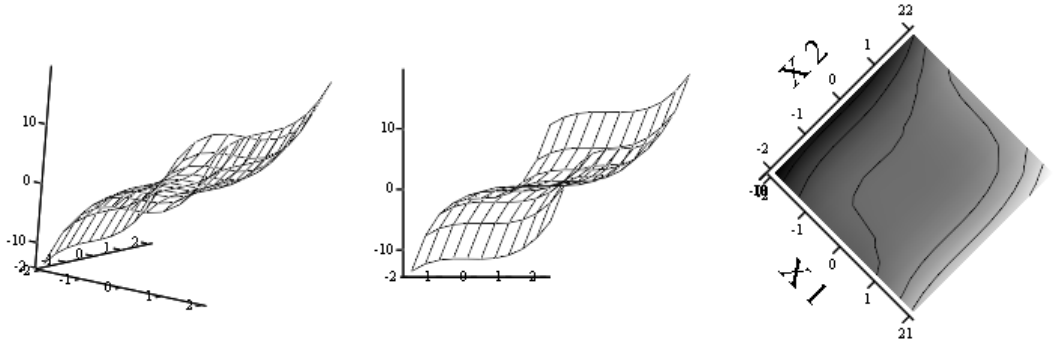


Figure F.21(a,b,c) – Sum value of 3rd order Taylor Series neuron focusing on the decision region – skewed on different inputs for 2nd and 3rd order

$$Sum = 0.5 + 0.05 \cdot x_1 + 1.05 \cdot x_2 + \frac{1.05 \cdot x_1^2 + 0.05 \cdot x_2^2}{2} + \left[\frac{9.0 \cdot x_1^3 + 3.0 \cdot x_2^3}{6} \right]$$

equation F.23

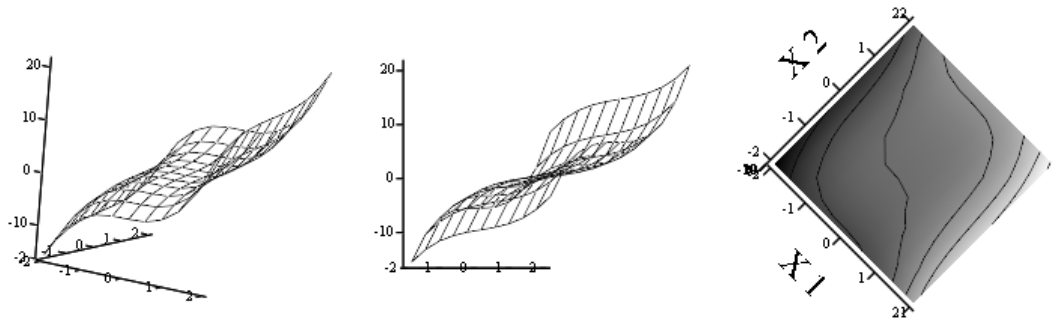


Figure F.22(a,b,c) – Sum value of 3rd order Taylor Series neuron focusing on the decision region – skewed on different inputs for 1st and same 2nd and 3rd order

$$Sum = 0.5 + 0.05 \cdot x_1 + 1.05 \cdot x_2 + \frac{1.05 \cdot x_1^2 + 0.05 \cdot x_2^2}{2} + \left[\frac{3.0 \cdot x_1^3 + 9.0 \cdot x_2^3}{6} \right]$$

equation F.24

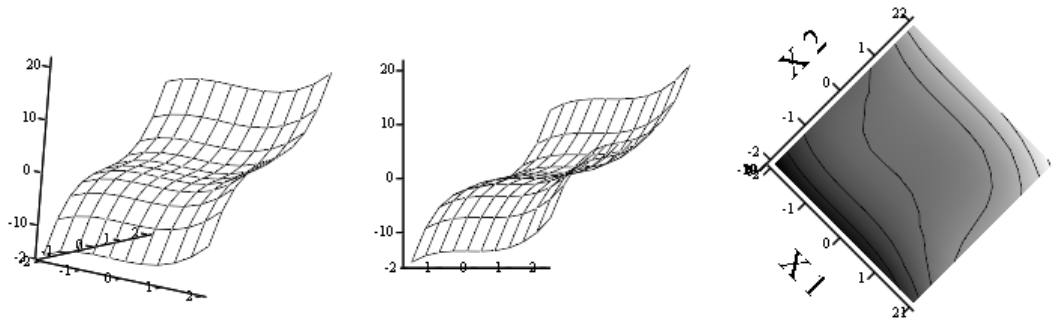


Figure F.23(a,b,c) – Sum value of 3rd order Taylor Series neuron focusing on decision region – skewed on inputs for same 1st and 3rd and different 2nd order

$$Sum = 0.5 + 0.05 \cdot x_1 + 1.05 \cdot x_2 + \frac{0.05 \cdot x_1^2 + 1.05 \cdot x_2^2}{2} + \left[\frac{9.0 \cdot x_1^3 + 3.0 \cdot x_2^3}{6} \right]$$

equation F.25

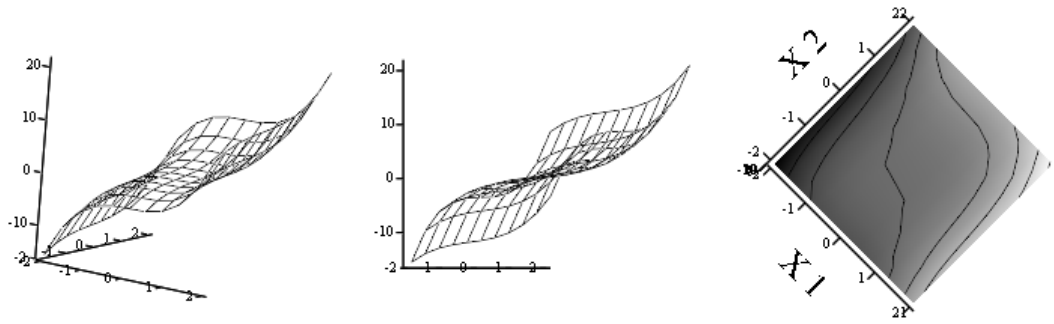


Figure F.24(a,b,c) – Sum value of 3rd order Taylor Series neuron focusing on decision region – skewed on inputs for same 1st and 2nd and different 3rd order

The above figures demonstrate the flexibility of the Taylor Series neuron, showing that the variation in the coefficients is independent. Thus allowing control of the neuron while permitting exploitation all the variation of the output-domain.

From Taylor Series neuron - mixed orders.

$$Sum = 0.5 + 0.05 \cdot x_1 - 0.05 \cdot x_2 + \frac{0.05 \cdot x_1^2 - 0.005 \cdot x_2^2}{2} + \left[\frac{-1.5 \cdot x_1^3 + 1.5 \cdot x_2^3}{6} \right]$$

equation F.26

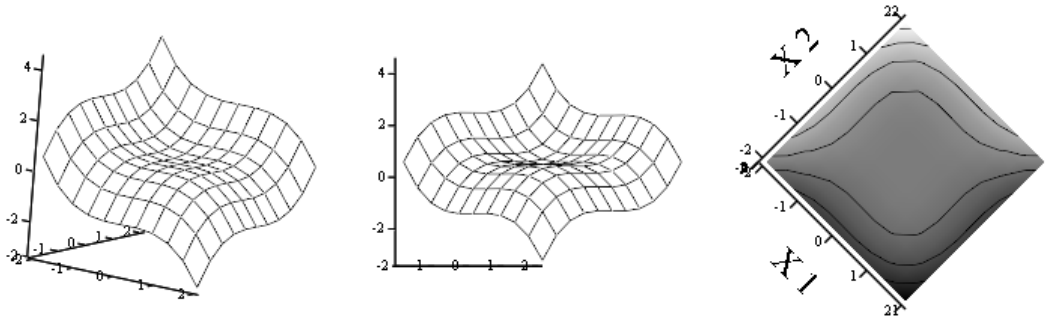


Figure F.25(a,b,c) – Sum value of 3rd order Taylor Series neuron focusing on the decision region with opposing polarity of various coefficients

$$Sum = 0.5 - 0.95 \cdot x_1 + 1.15 \cdot x_2 + \left[\frac{0.5 \cdot x_1^2 + 3.0 \cdot x_2^2}{2} \right]$$

equation F.27

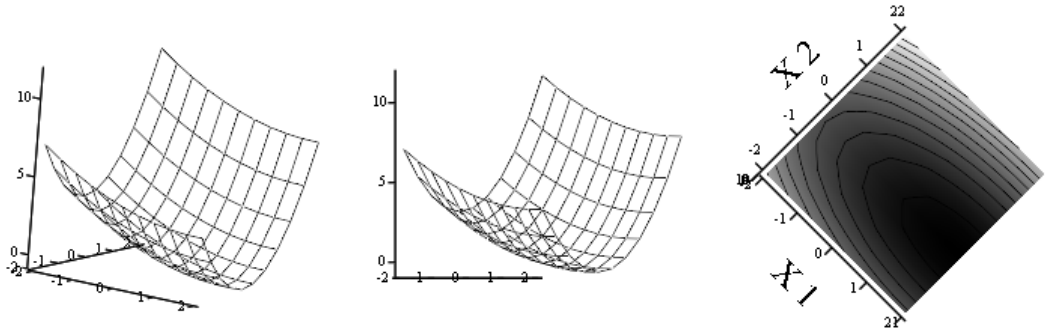


Figure F.26(a,b,c) – Sum value of 2nd order Taylor Series neuron focusing on the decision region with opposing polarity of 1st order coefficients

$$Sum = 0.5 - 0.11 \cdot x_1 - 0.06 \cdot x_2 + \frac{1.22 \cdot x_1^2 + 1.70 \cdot x_2^2}{2} + \left[\frac{4.26 \cdot x_1^3 + 5.79 \cdot x_2^3}{6} \right]$$

equation F.28

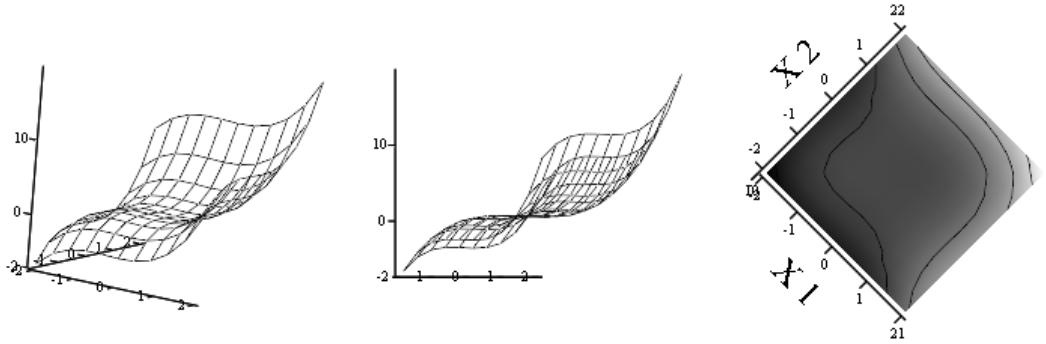


Figure F.27(a,b,c) – Sum value of 3rd order Taylor Series neuron focusing on the decision region with negative polarity of 1st order coefficients

From Taylor Series neuron –output functions.

The output functions are applied to the 2nd order Taylor Series neuron as expressed in equation 5.11 and shown in figures 5.10.

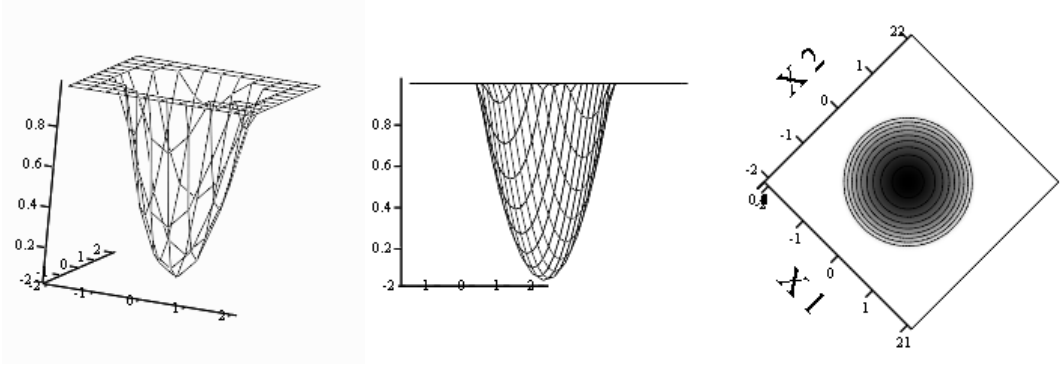


Figure F.28(a,b,c) – Piecewise linear output functions

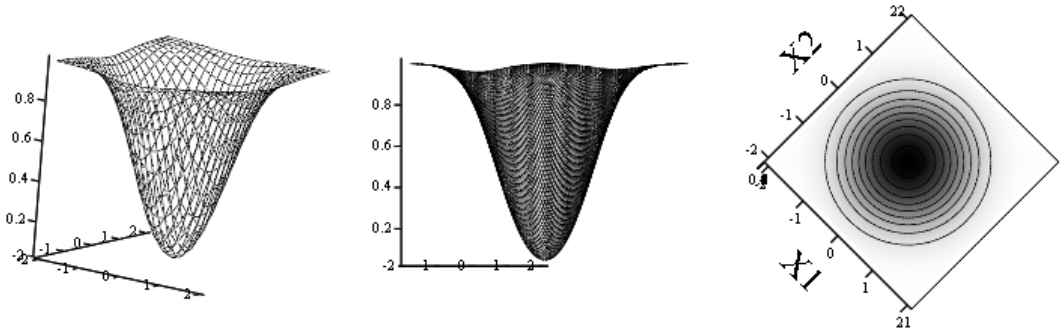


Figure F.29(a,b,c) – Hyperbolic tangent output functions

The same output functions are now applied to the 3rd order Taylor Series neuron as expressed in equation 5.12 and shown in figures 5.11.

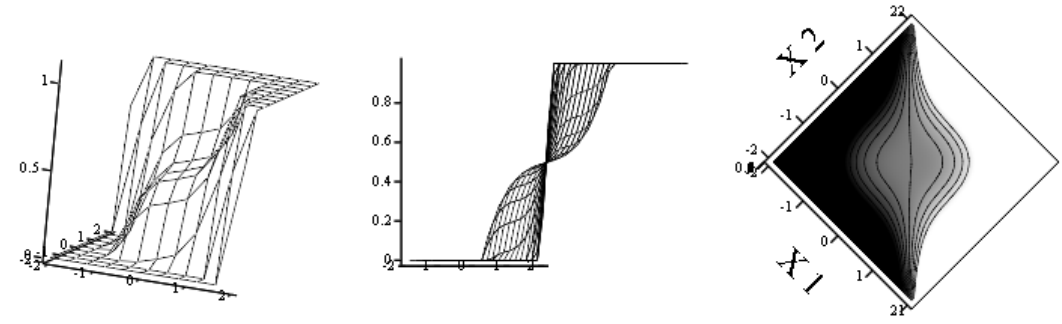


Figure F.30(a,b,c) – Piecewise linear output functions

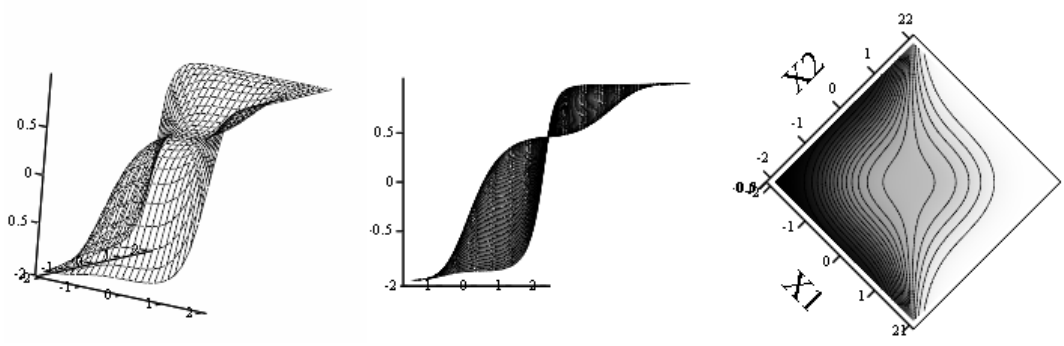


Figure F.31(a,b,c) – Hyperbolic tangent output functions

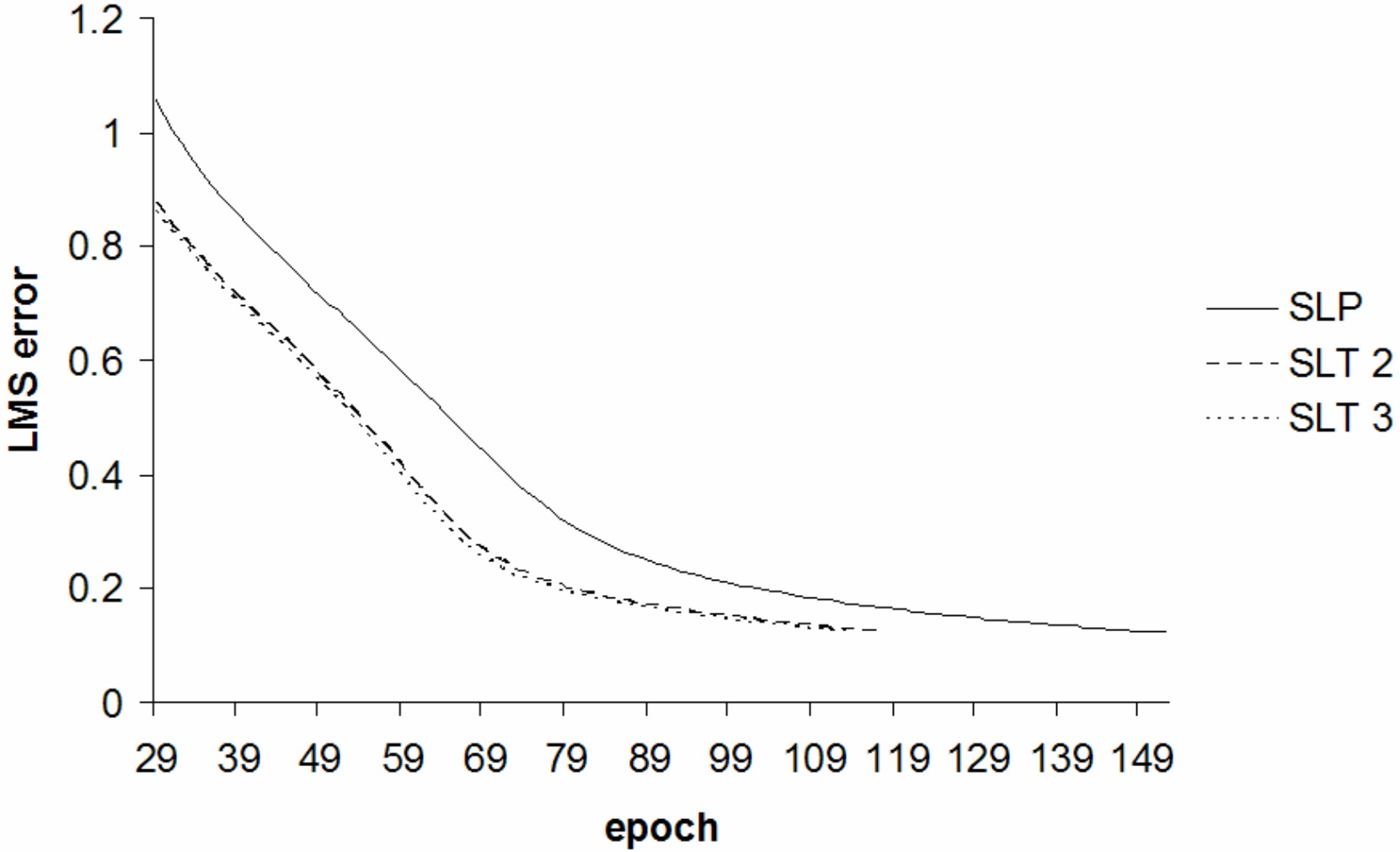


Figure 5.31a – Comparison of error vs. epoch for SLP and SLT networks

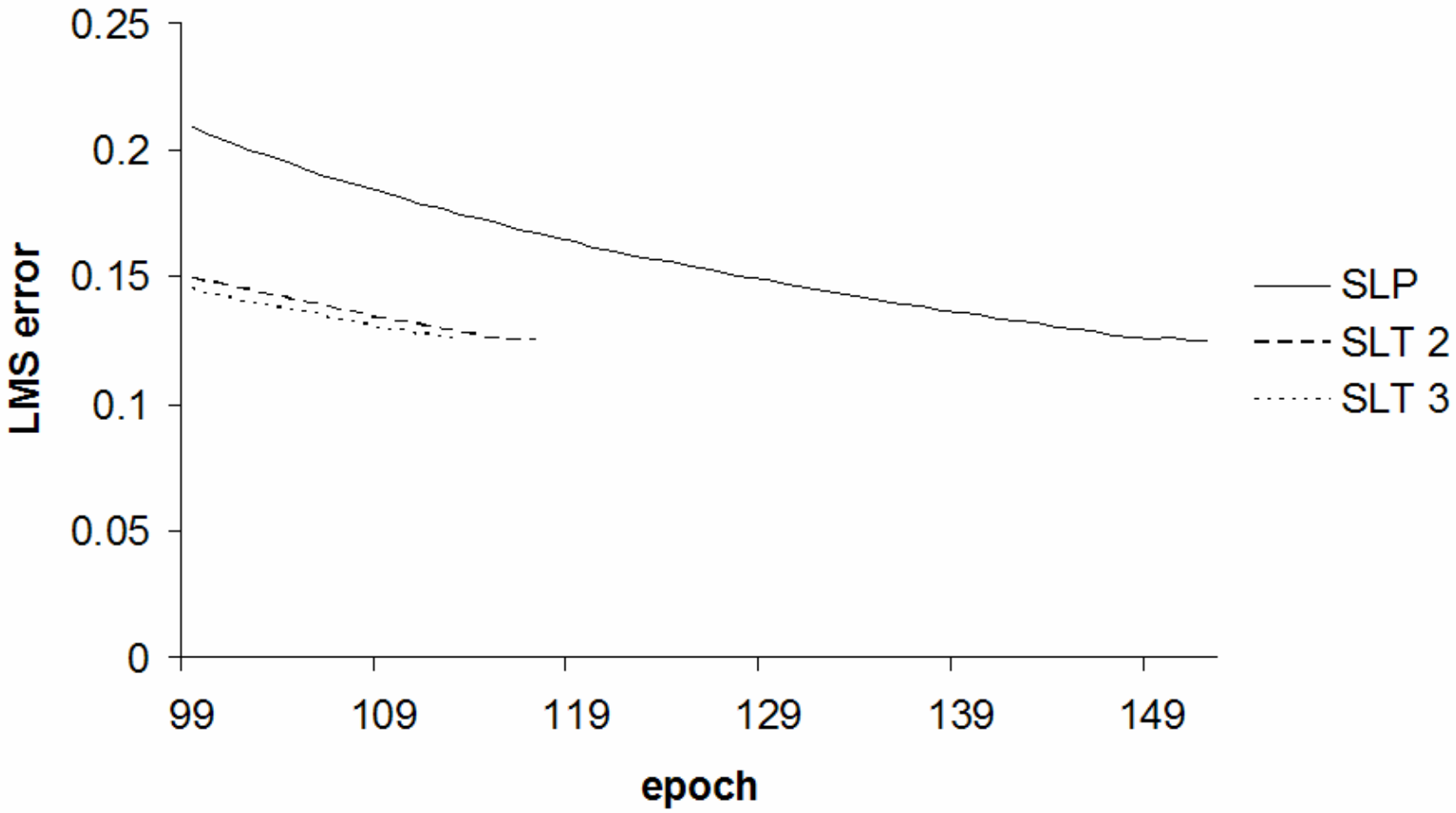


Figure 5.31b – Comparison of error vs. epoch for SLP and SLT networks

epoch ≥ 99

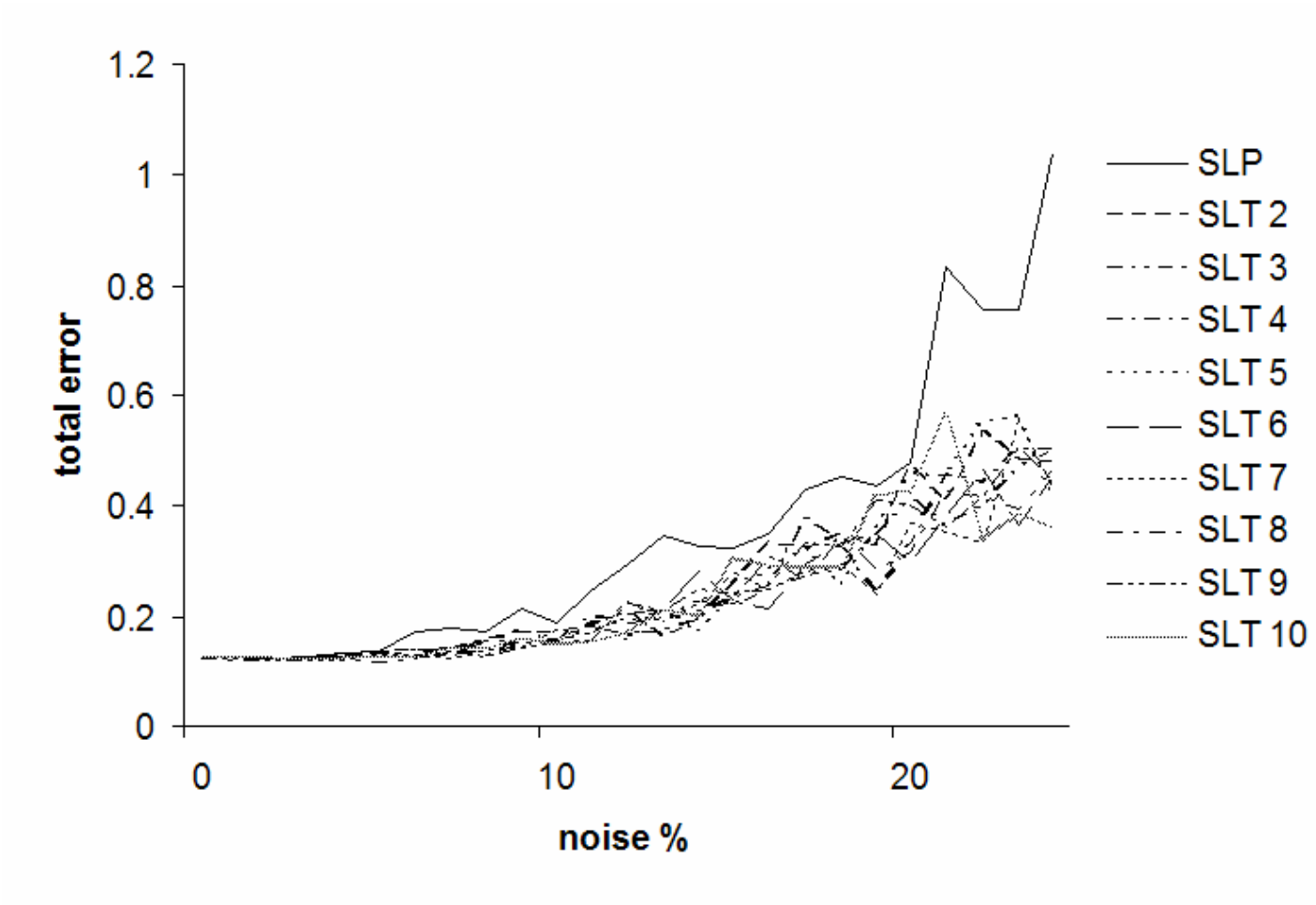


Figure 5.33a – Comparison of error vs. noise% for SLP and SLT networks

network inputs ϵ {0,1}

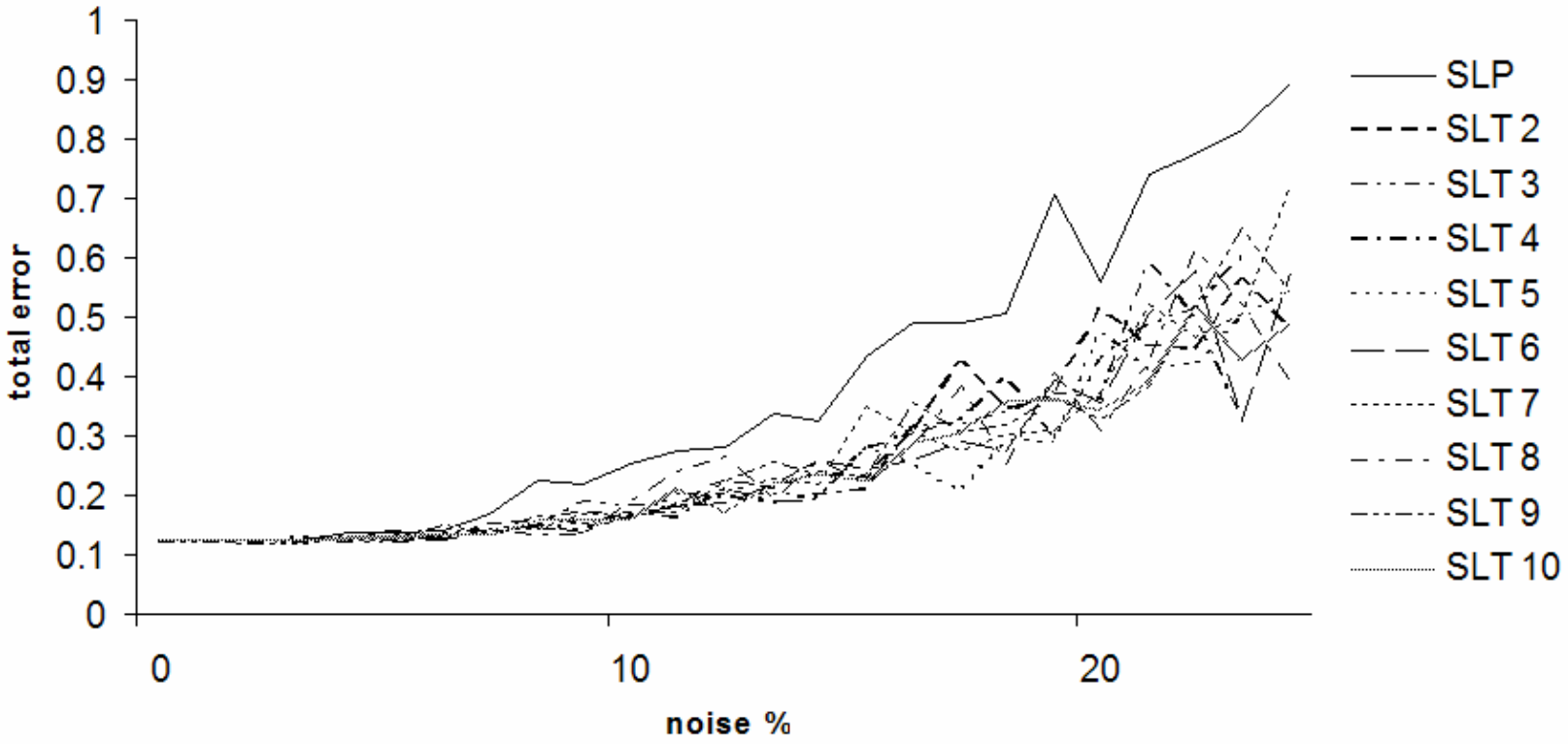


Figure 5.33b – Comparison of error vs. noise% for SLP and SLT networks

network inputs ϵ {0.1,0.9}

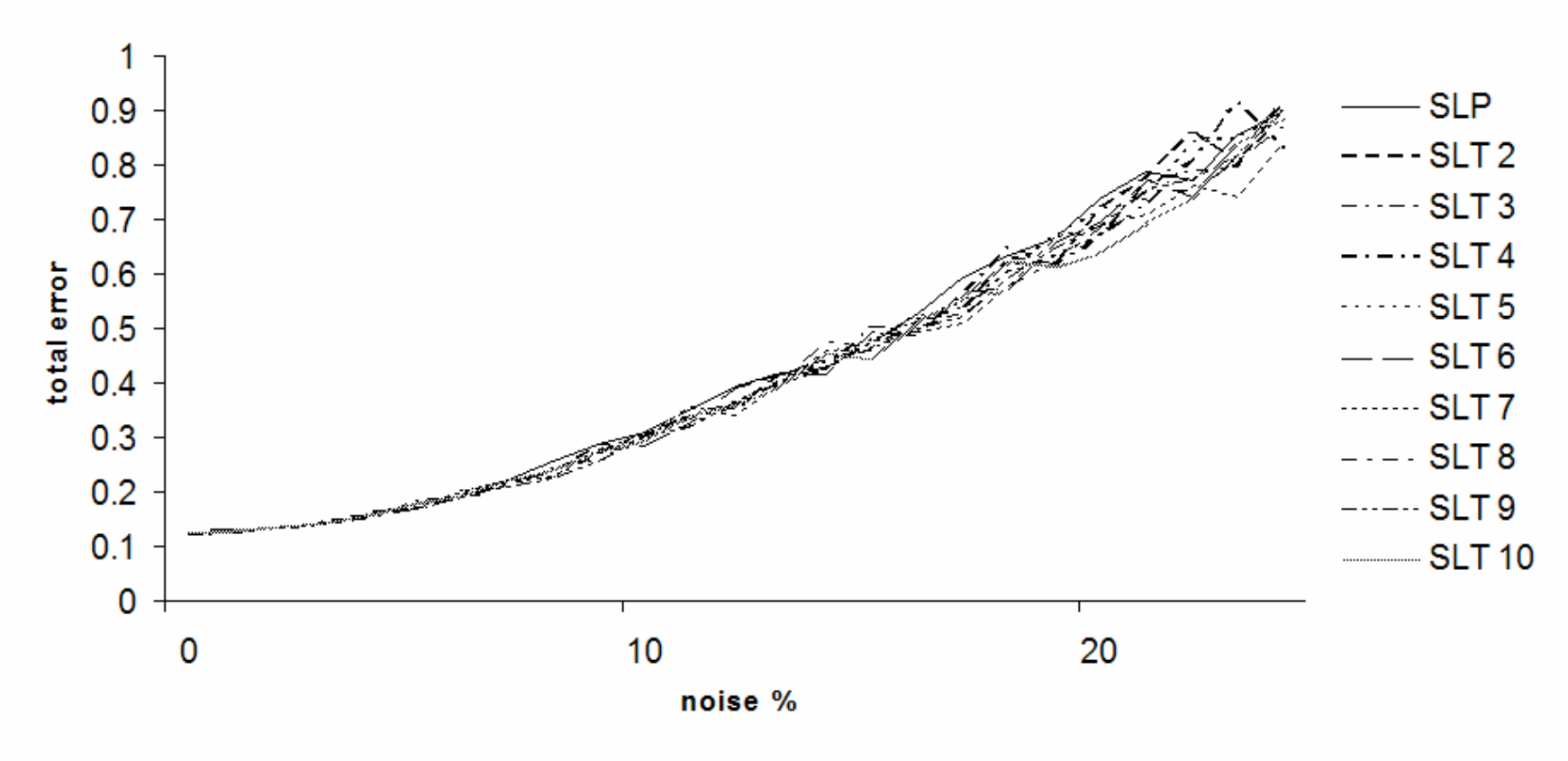


Figure 5.34 – Comparison of error vs. noise% for SLP and SLT networks

network inputs ϵ {0,1} - targets ϵ {0.1,0.9}

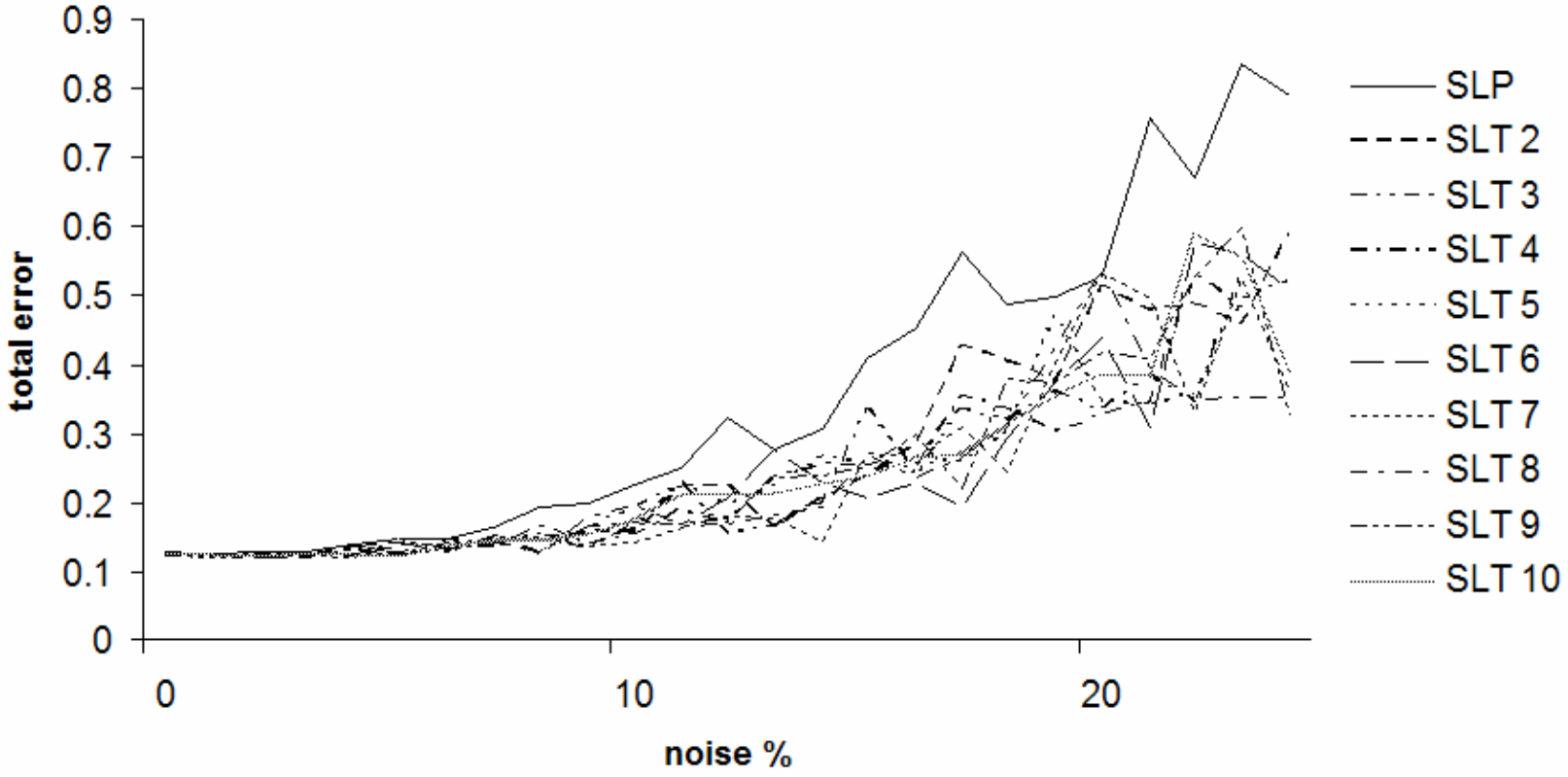


Figure 5.35 – Comparison of error vs. noise% for SLP and SLT networks

network inputs ϵ [0,1] - targets ϵ {0,1}

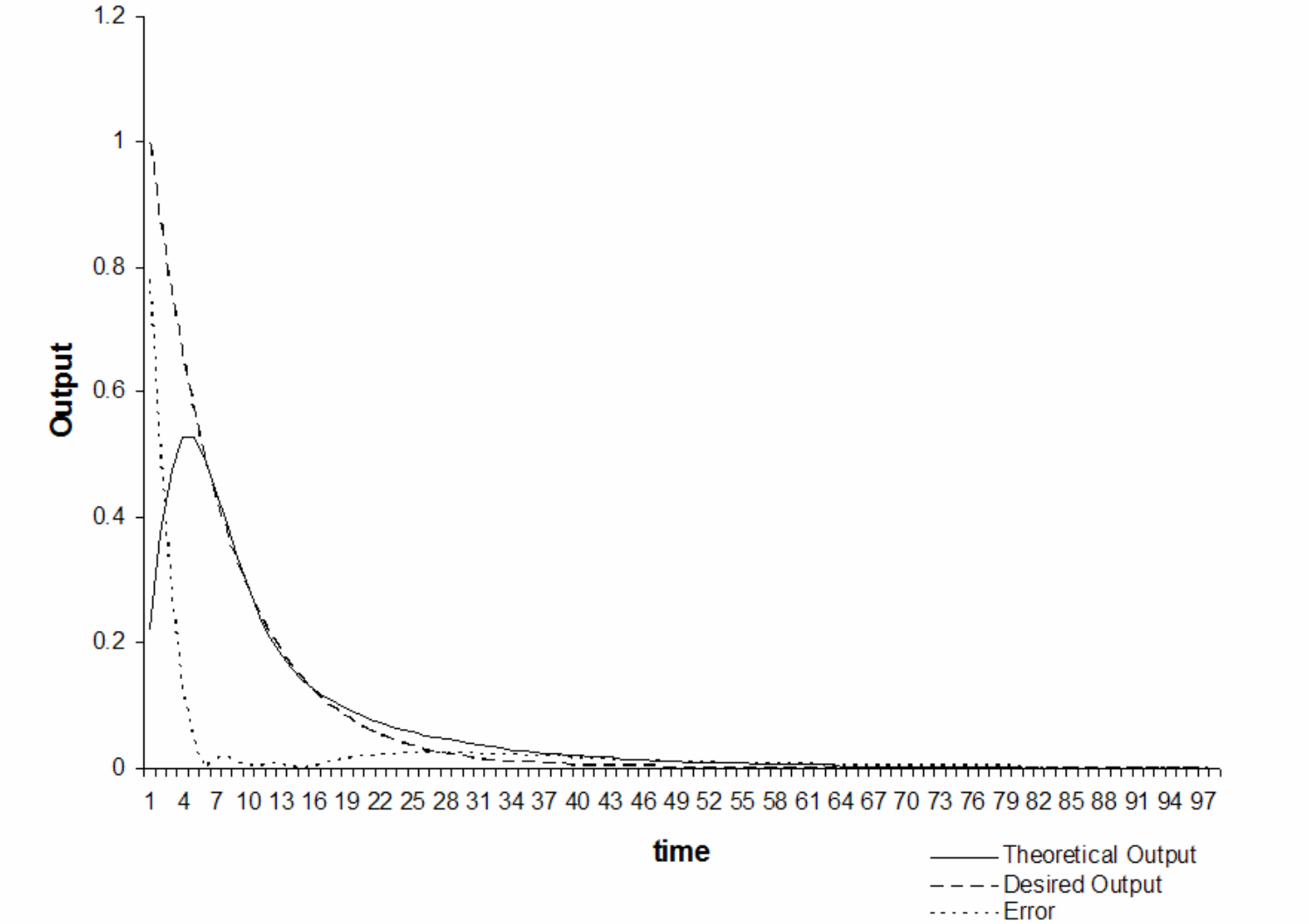


Figure 5.40 – Time-Series exponential decay – theoretical output

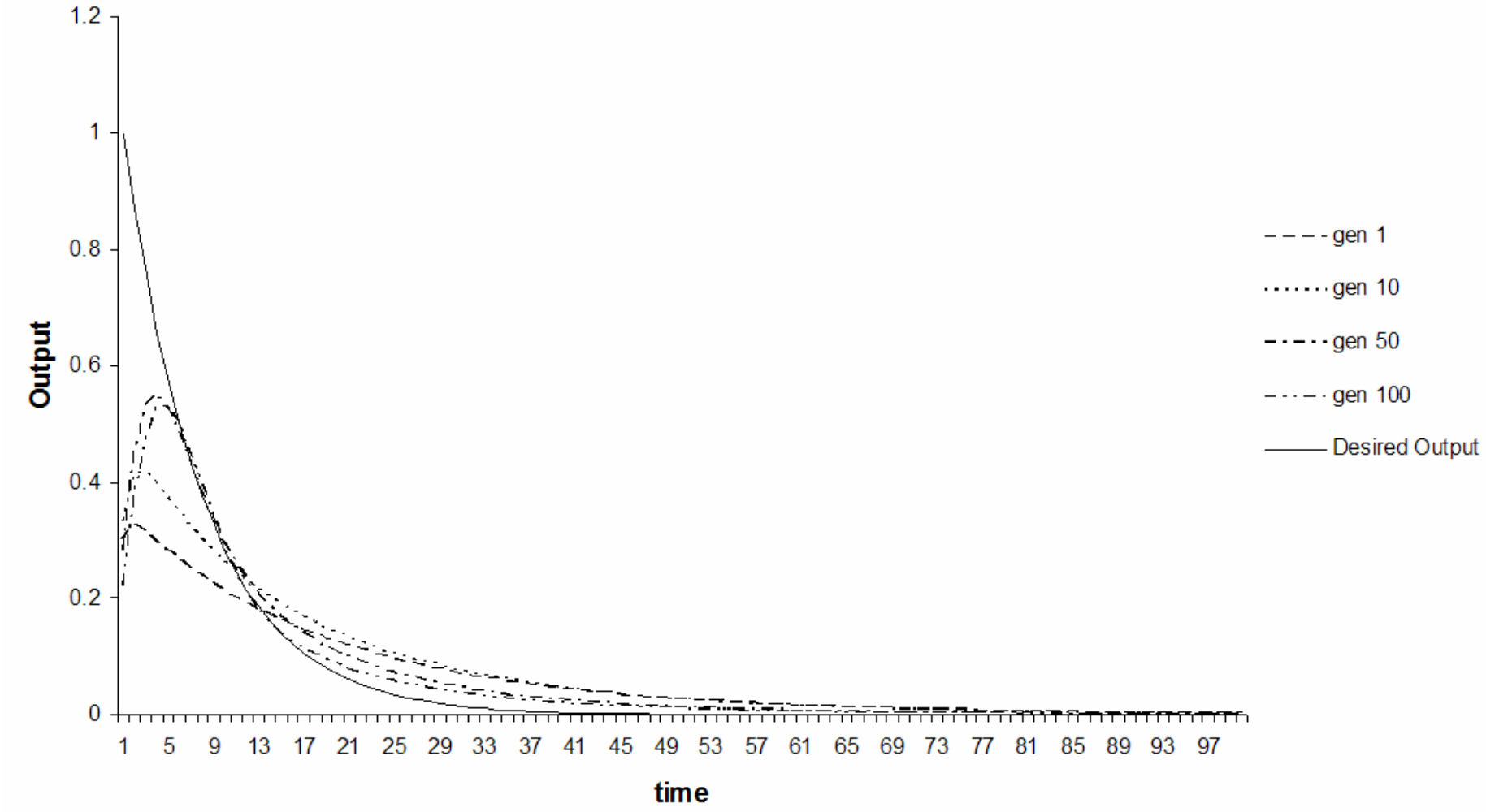


Figure 5.41 – Time-Series exponential decay - achieved

Appendix G

Artificial BioChemical Networks Results

G.1 Introduction to the Appendix

Additional results and enlarged figures from chapter 8 are included in this appendix for fullness and clarification. Each is placed under the title of the section which they relate too.

From 8.2.2 Successful ABN_w – Trained using a GA – Success

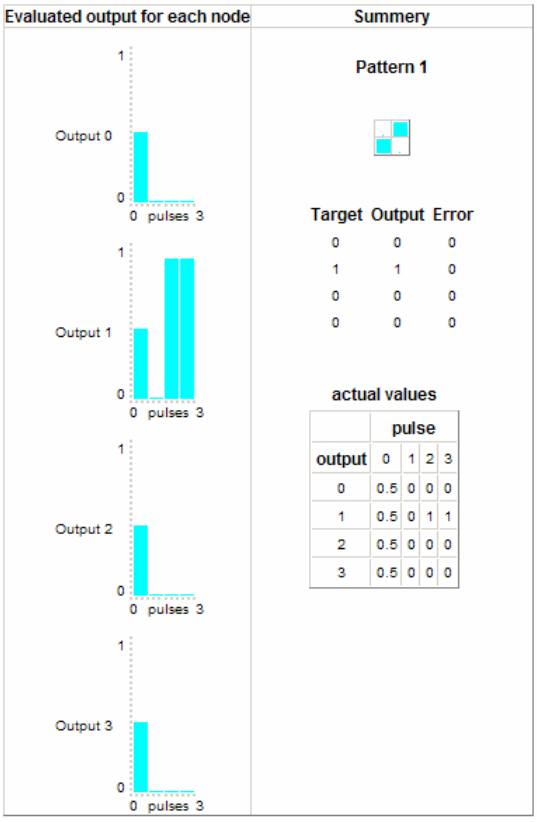


Figure G.1 – ABN_w-GA output pulse – pattern 1

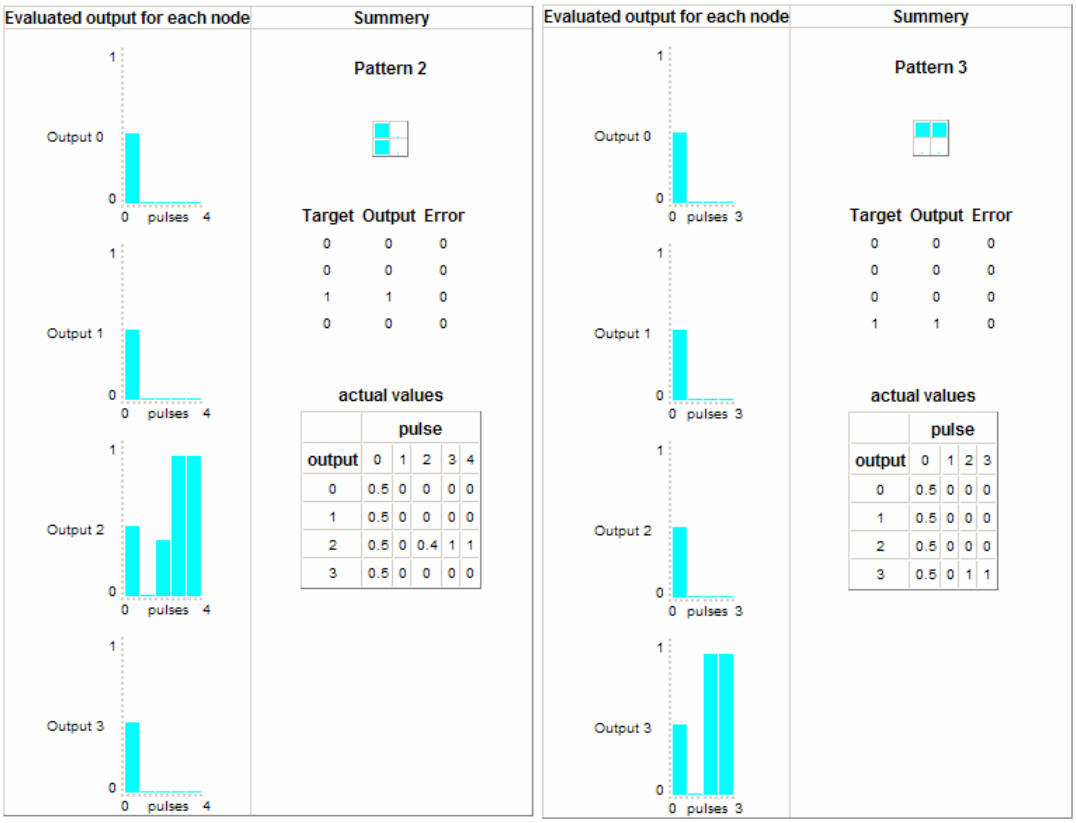


Figure G.2 G.3 – ABN$_w$-GA output pulse – pattern 2,3

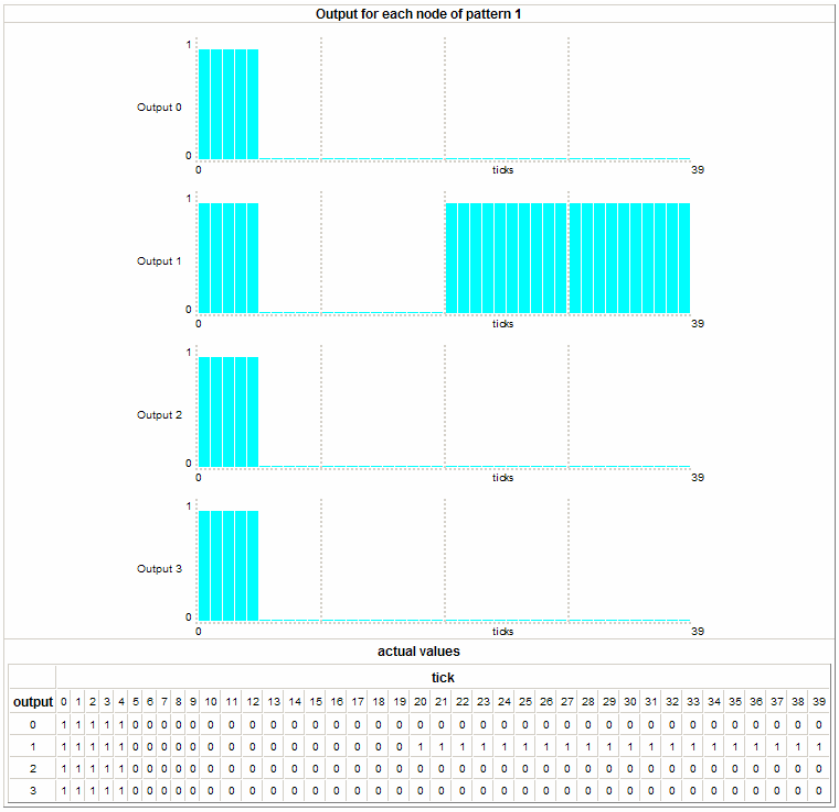
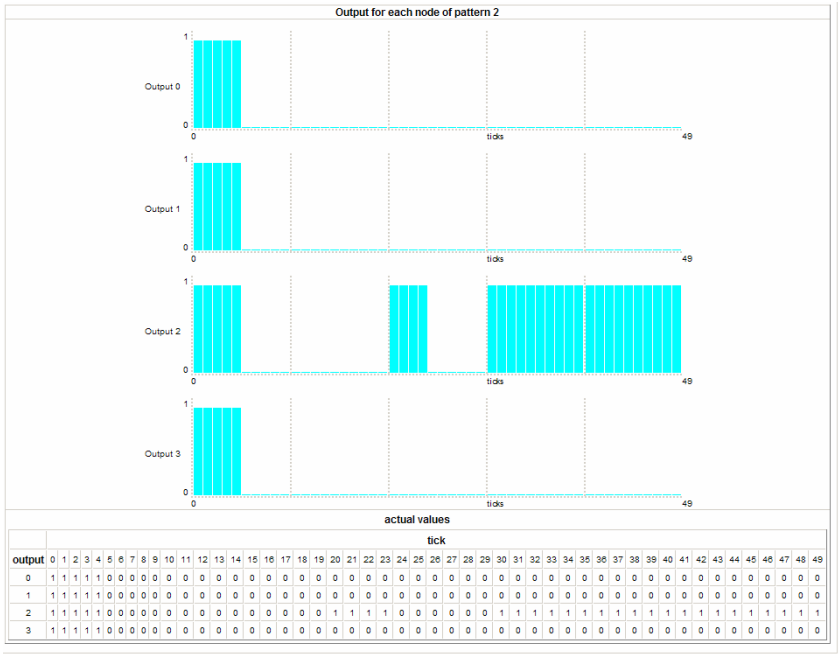
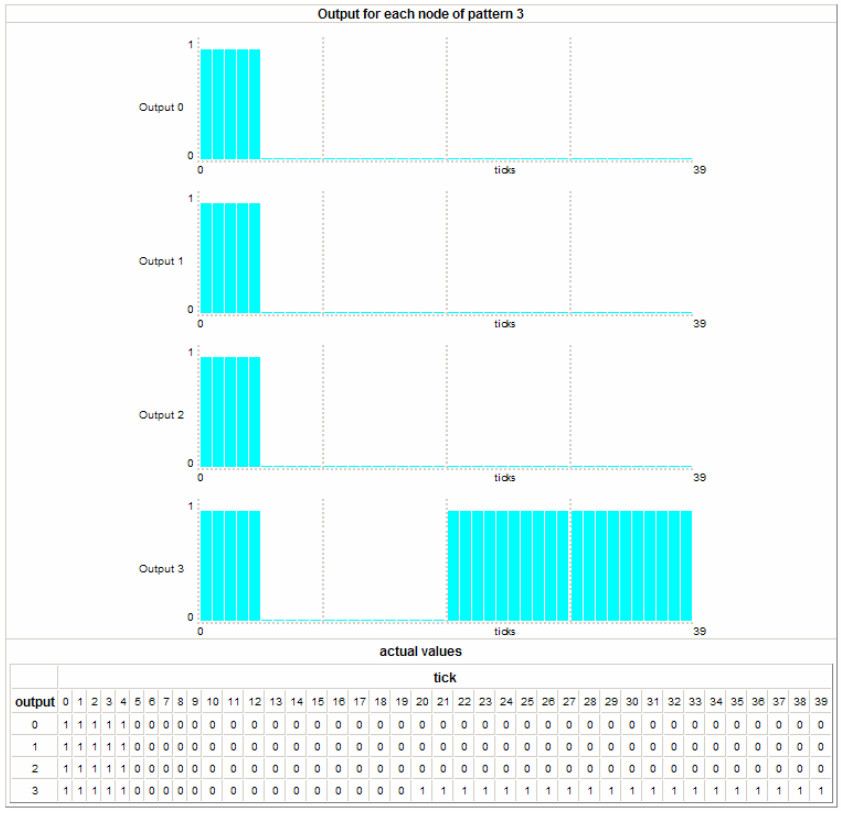


Figure G.4 – ABN$_w$-GA output ticks – pattern 1



output	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49
0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure G.5 – ABN$_w$-GA output ticks – pattern 2



output	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure G.6 – ABN$_w$-GA output ticks – pattern 3

From 8.2.8 ABN_w – Trained using a GA - Results of Noise Tolerance

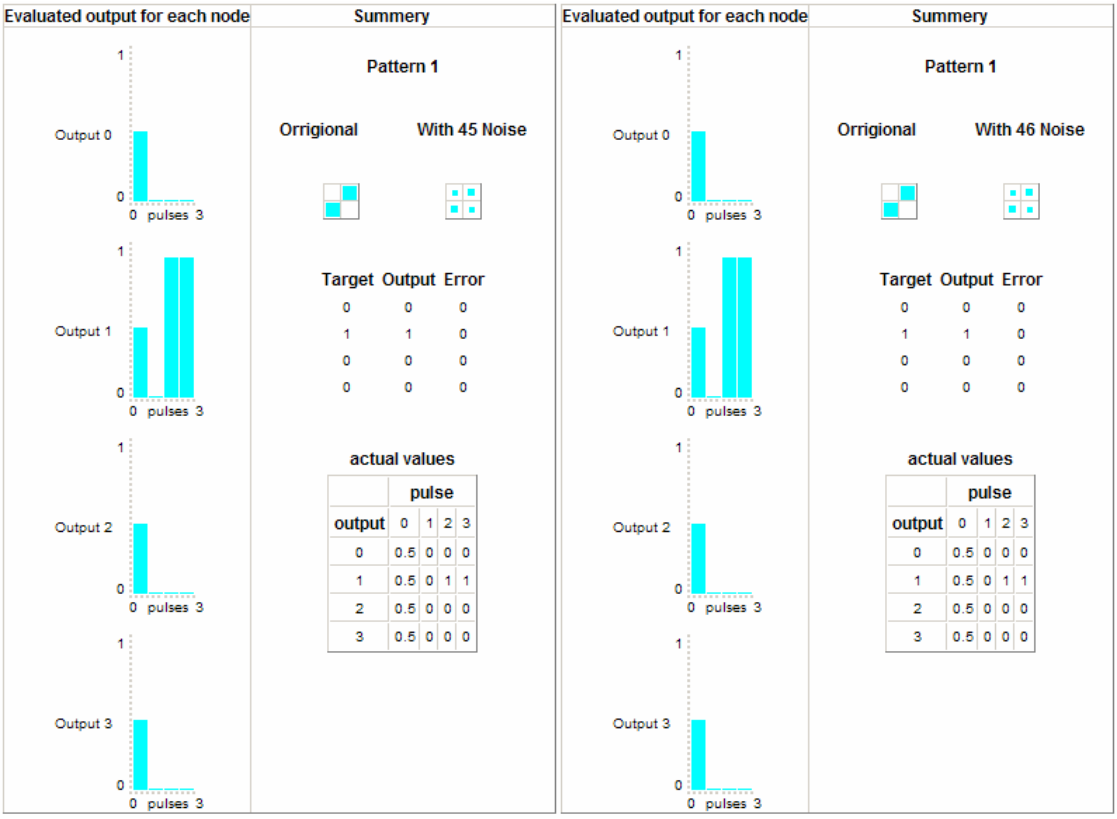


Figure G.7 – ABN_w-GA output pulse - noise 45%,46% – pattern 1

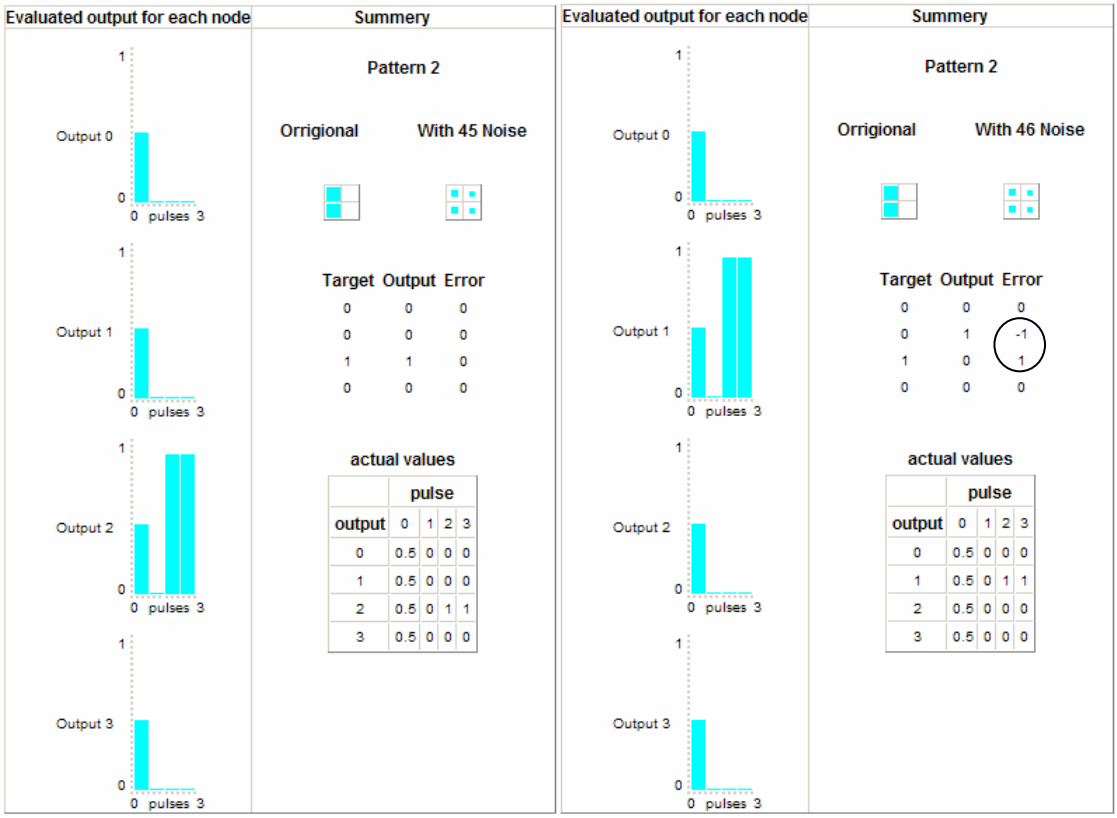


Figure G.8 – ABN_w-GA output pulse - noise 45%,46% – pattern 2

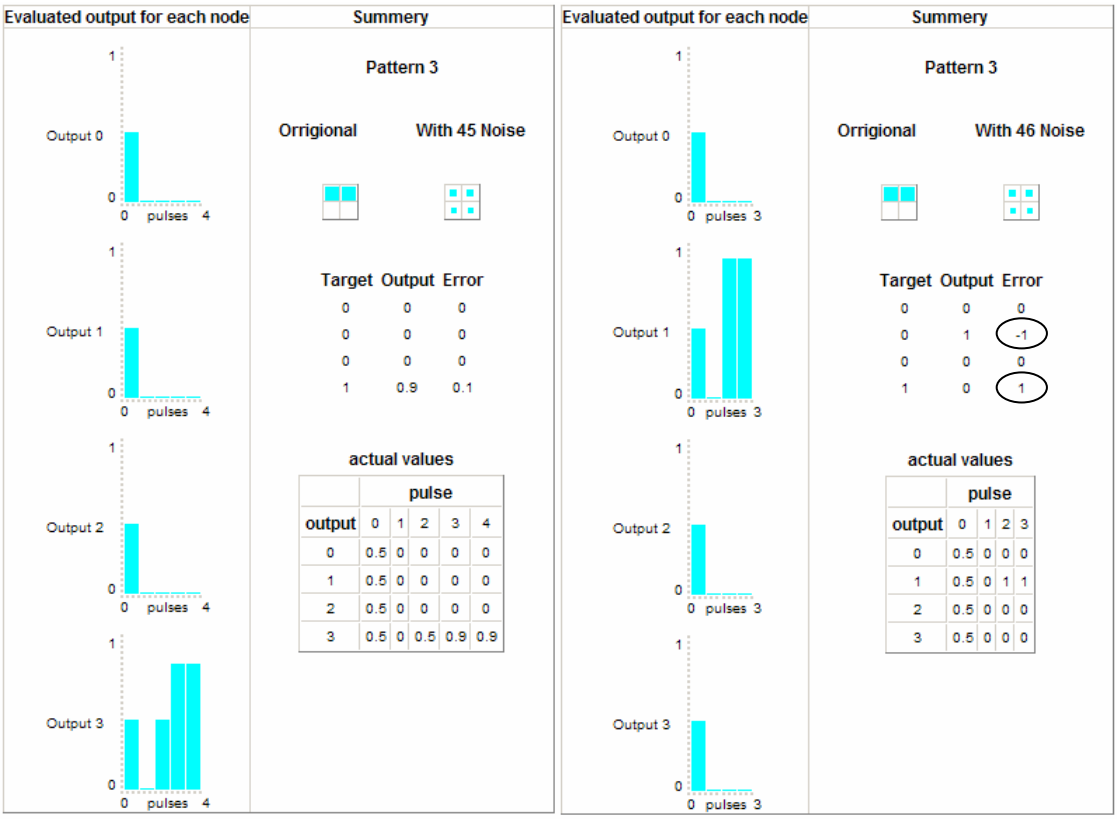


Figure G.9 – ABN$_w$-GA output pulse - noise 45%,46% – pattern 3

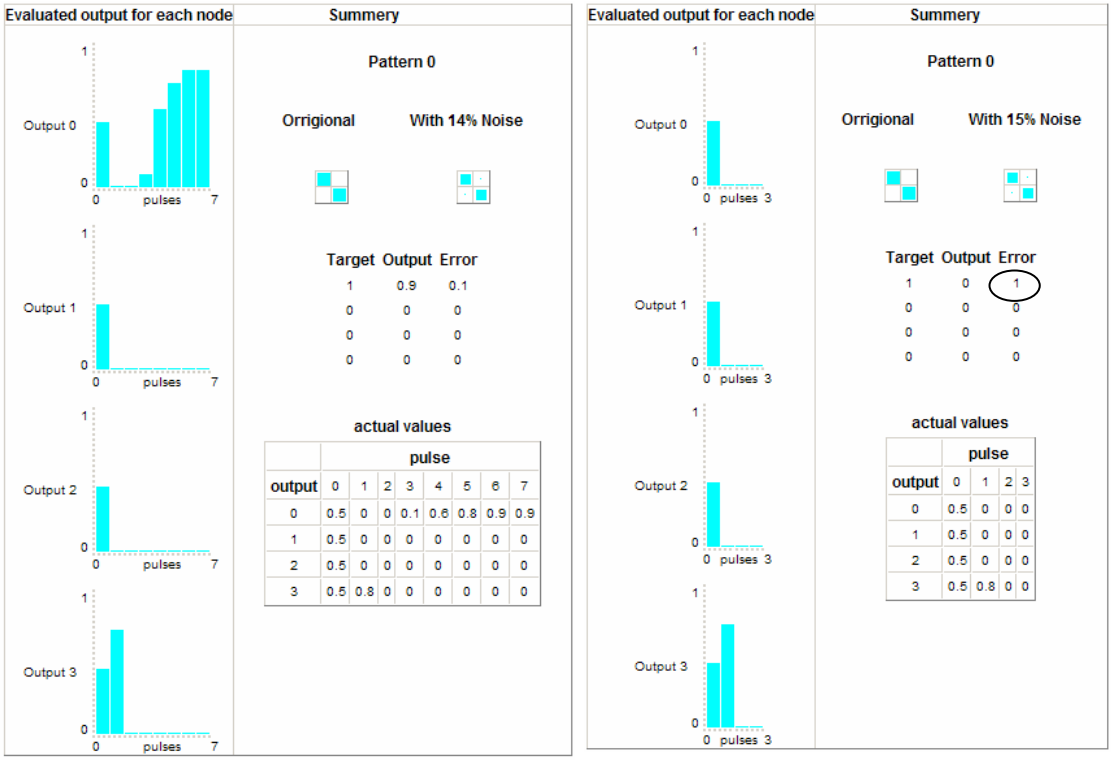


Figure G.10 – ABN$_w$-GA output pulse - noise 14%,15% - pattern 0

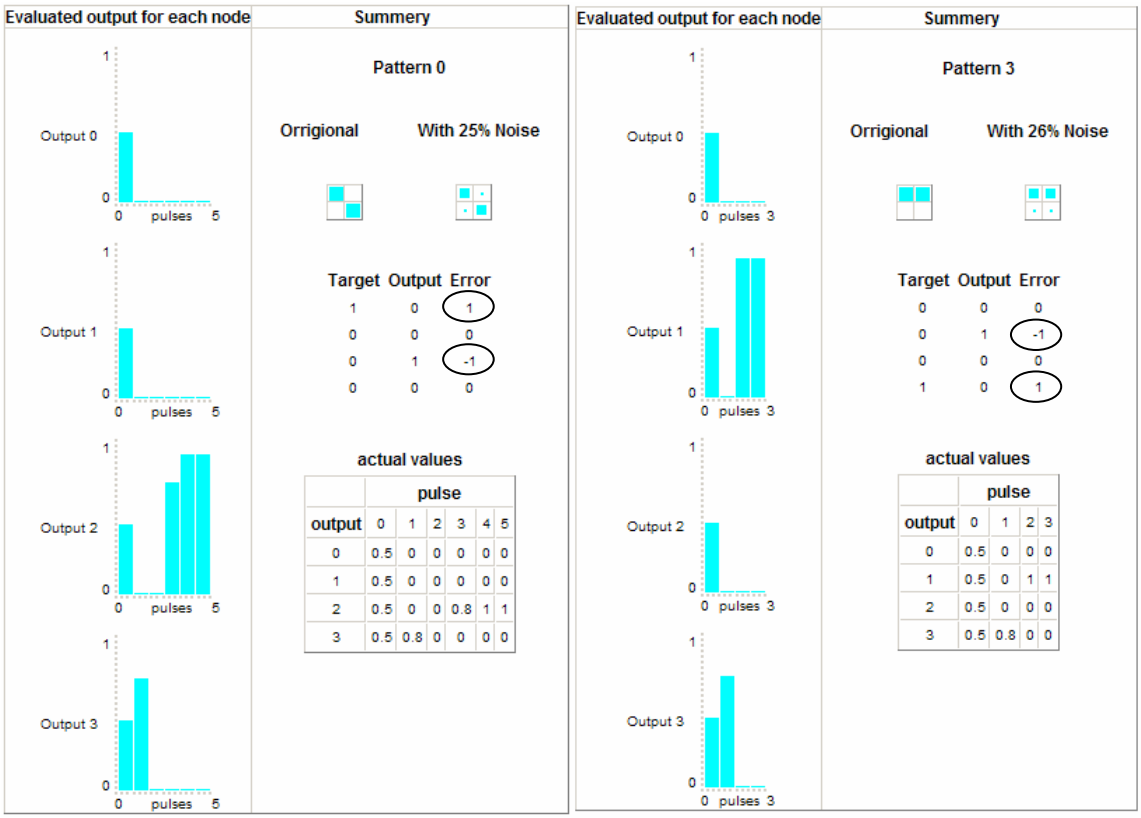


Figure G.11 – ABN$_w$-GA output pulse - noise 25%,26% - patterns 0,3

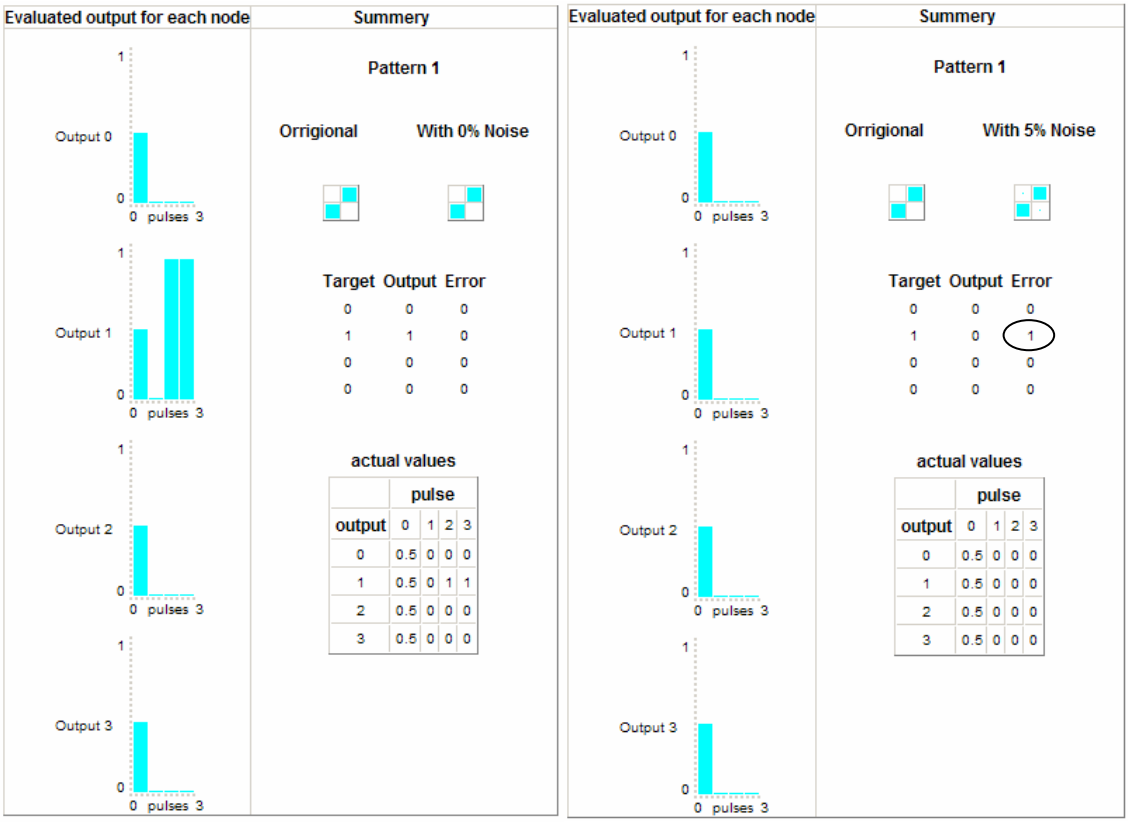


Figure G.12 – ABN$_w$-GA output pulse - noise 0%,5% - pattern 1

From 8.3.4 Comparison of ABN_w – BP and ABN_w – GA

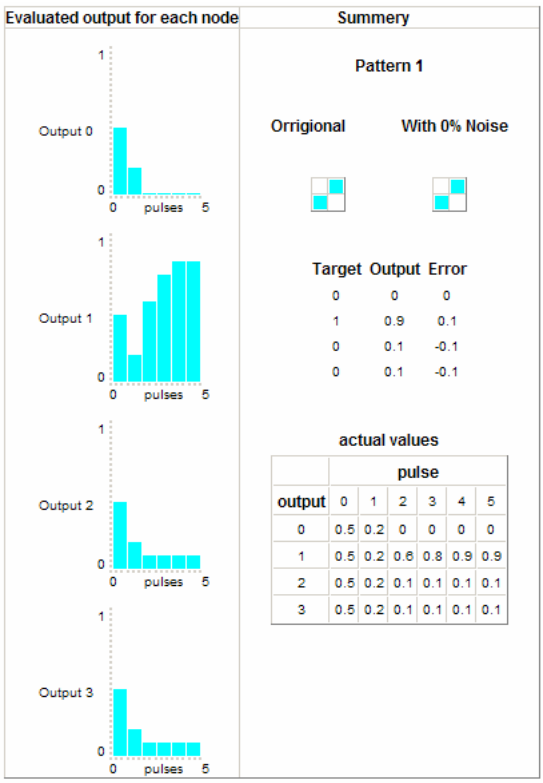
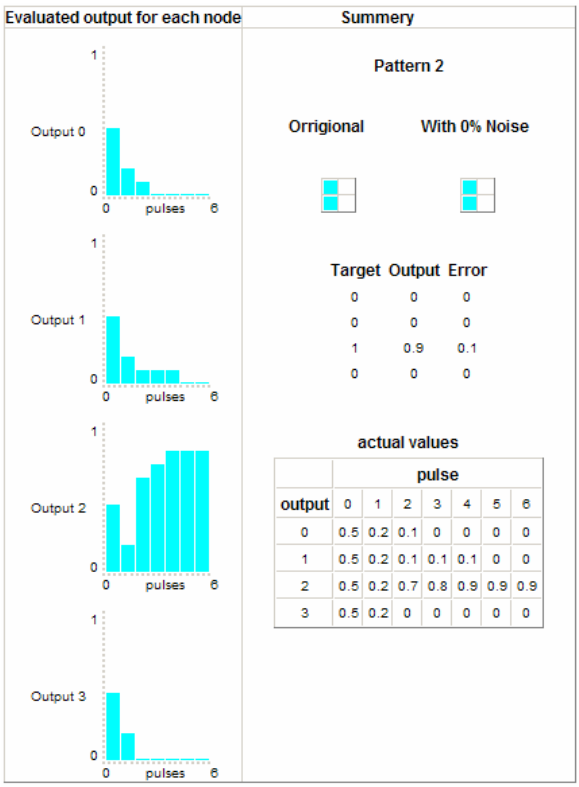


Figure G.13 – ABN_w-BP output pulse – pattern 1



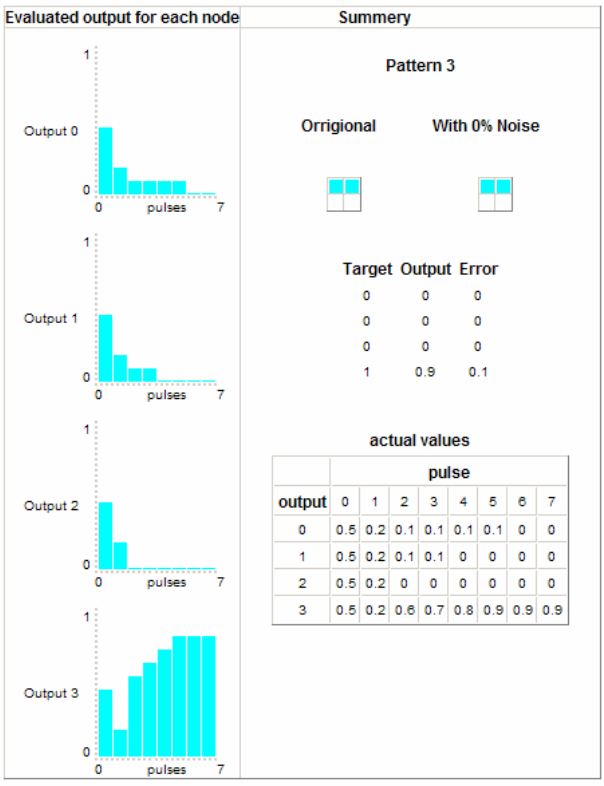
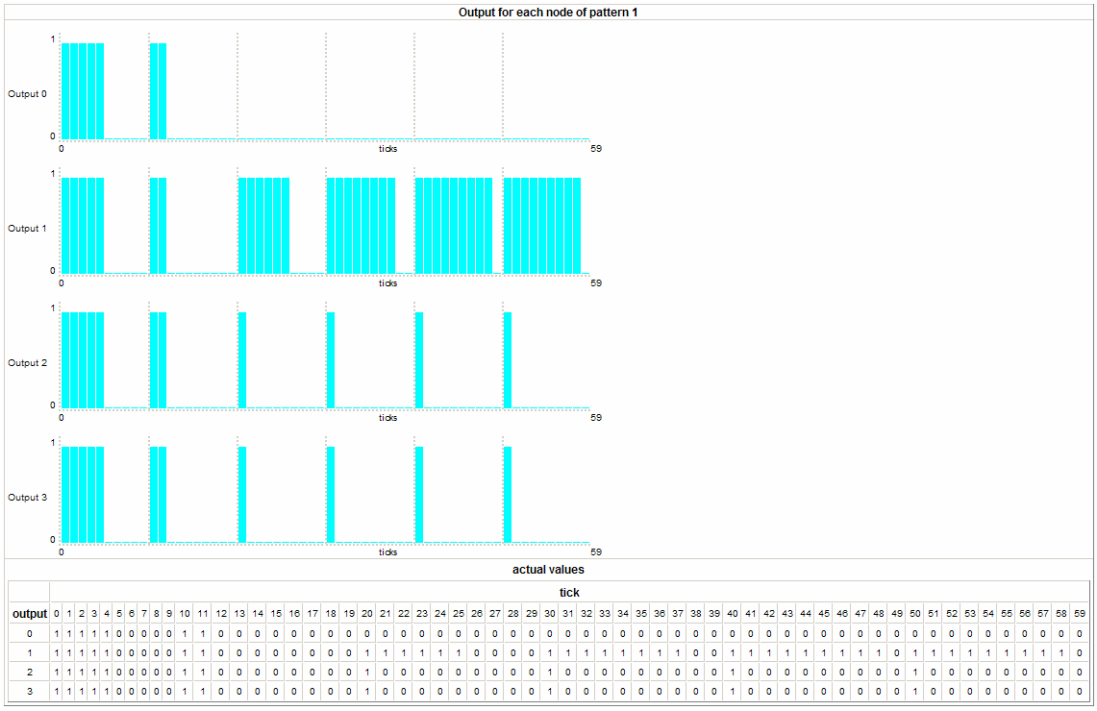
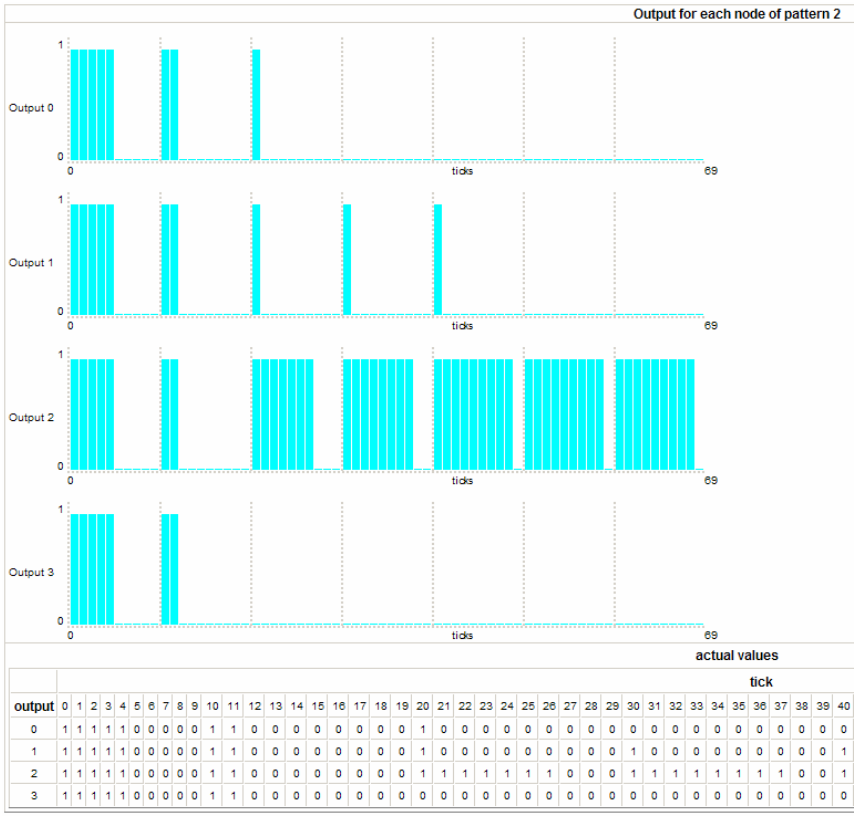


Figure G.14, G.15 – ABN_w-BP output pulse – patterns 2,3



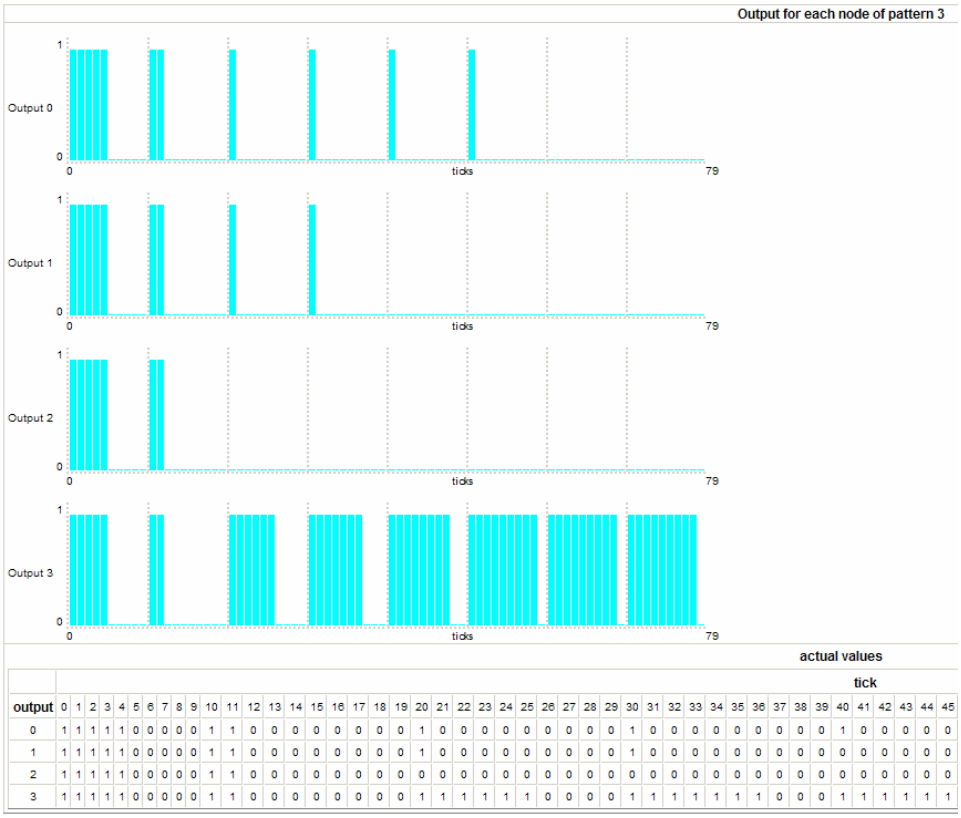
output	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
0	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	0
2	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
3	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

Figure G.16 – ABN$_w$-BP output ticks – pattern 1



output	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
0	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
2	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	0	0	1	
3	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure G.17 – ABN$_w$-BP output ticks – pattern 2



output	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
0	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1

Figure G.18 – ABN$_w$-BP output ticks – pattern 3

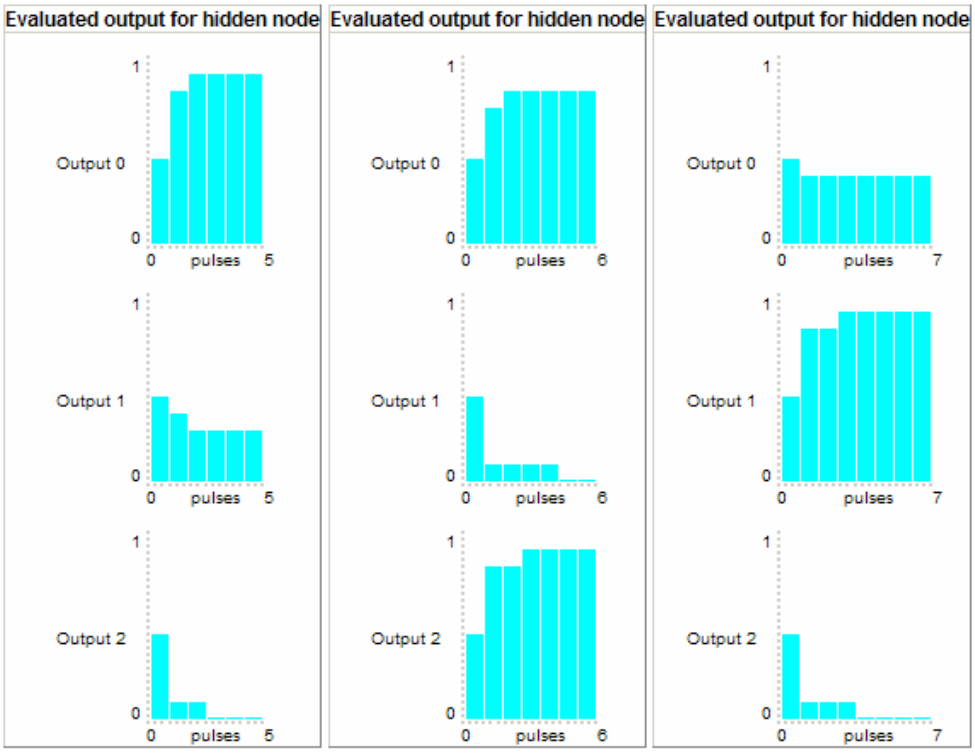


Figure G.19, G.20, G.21 – ABN$_w$-BP hidden nodes – patterns 1,2,3

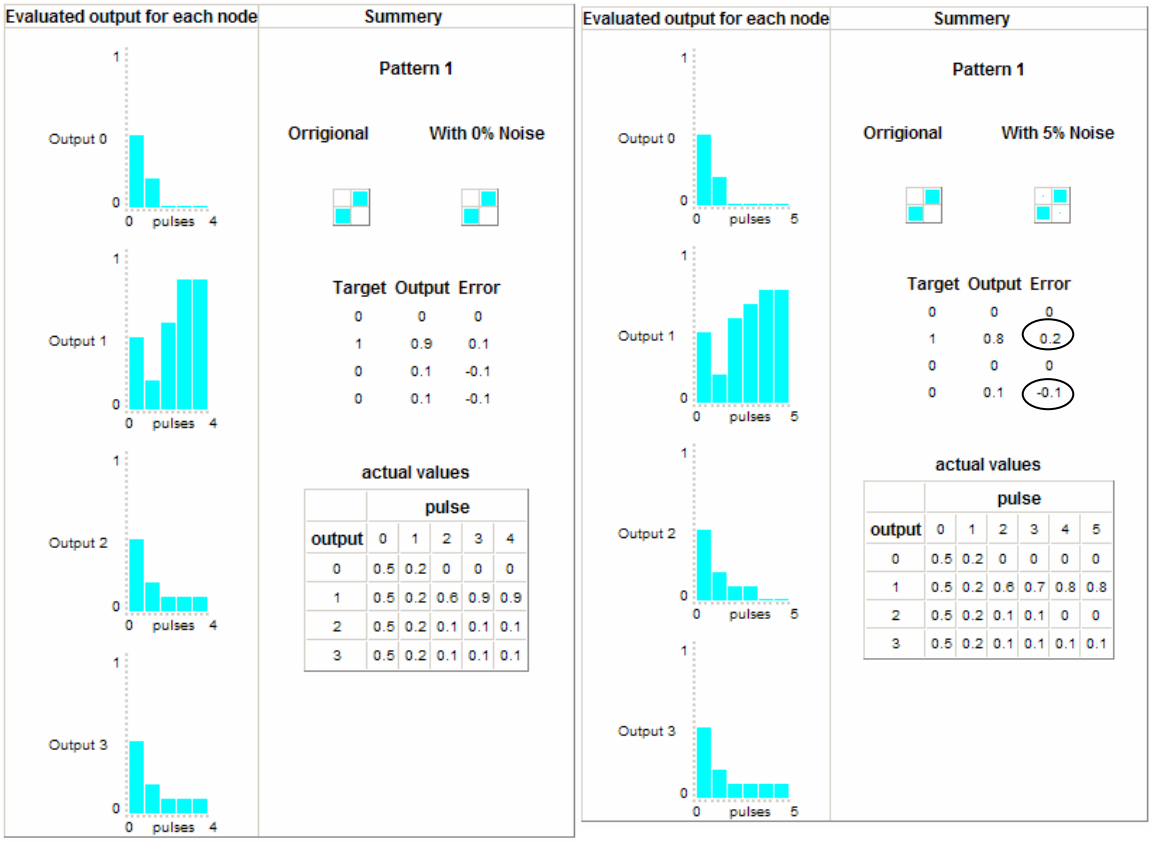


Figure G.22 – ABN_w-BP output pulse - target e_{ABN} 0.5 - noise 0%,5% - pattern 1

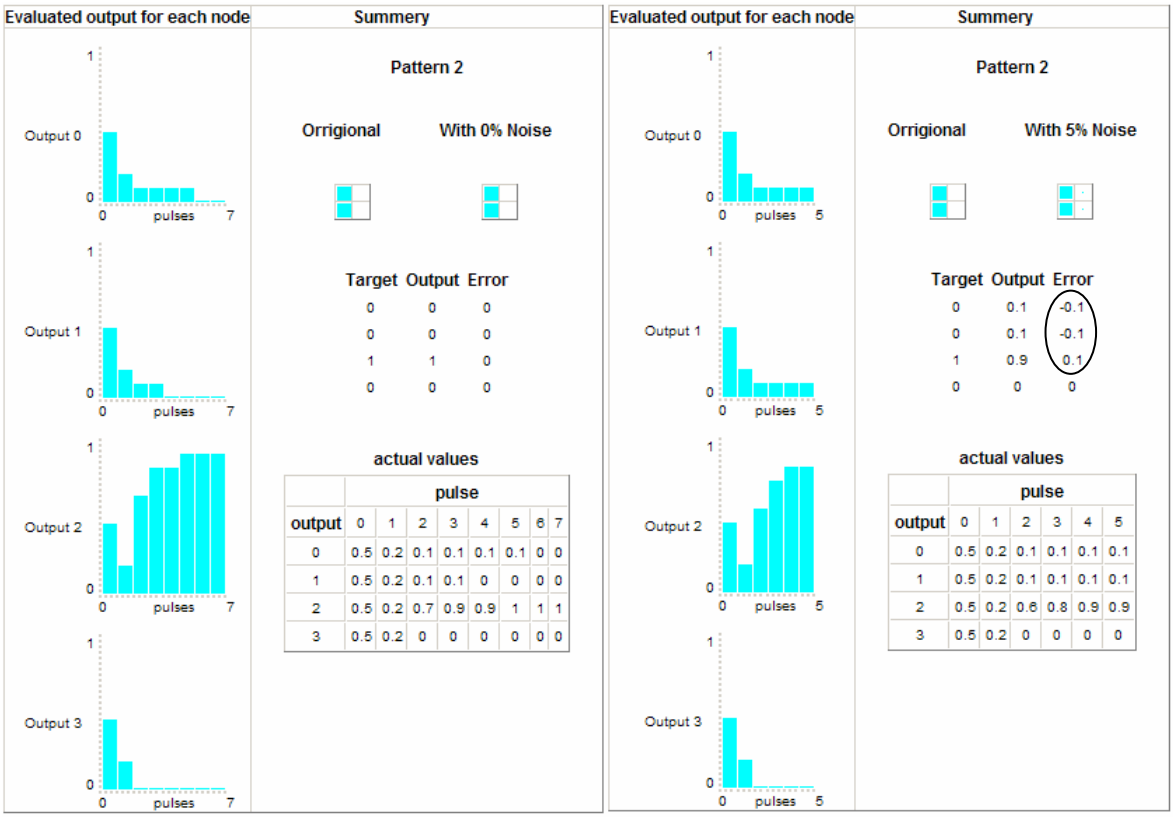


Figure G.23 – ABN_w-BP output pulse - target e_{ABN} 0.5 - noise 0%,5% - pattern 2

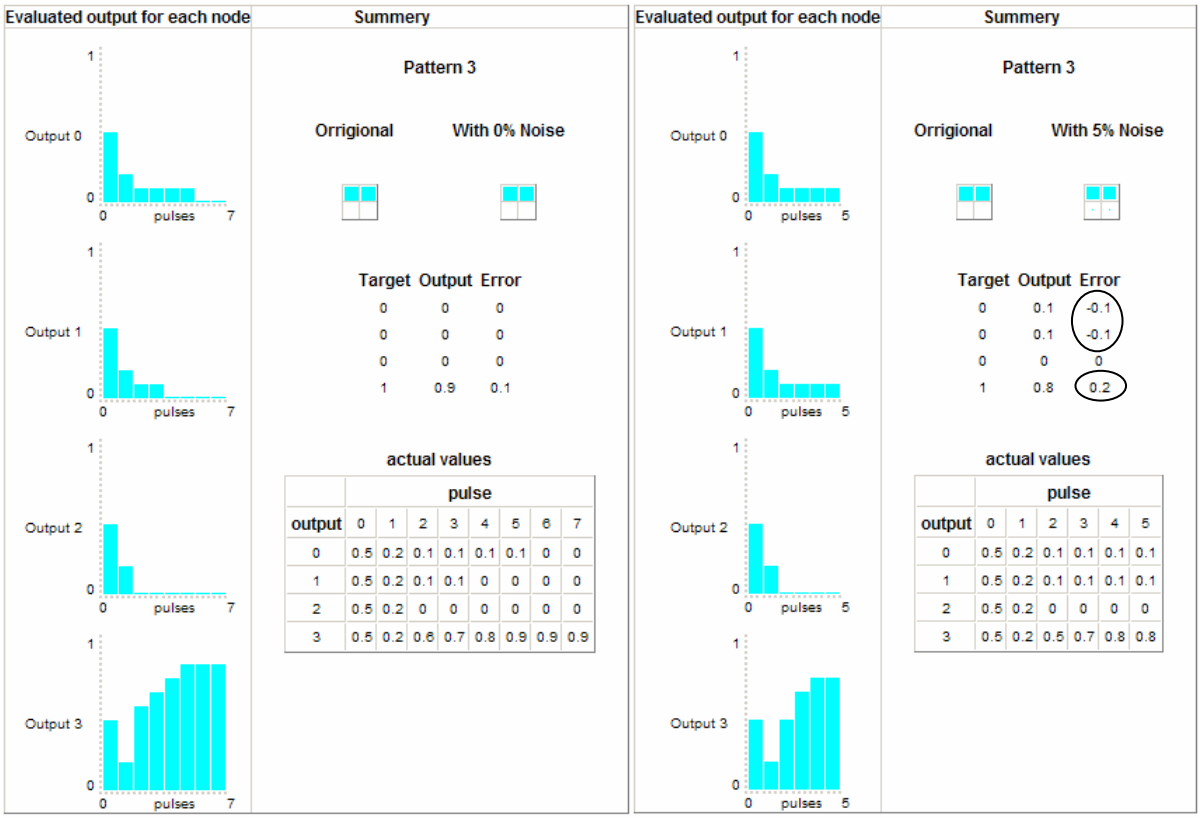


Figure G.24 – ABN_w-BP output pulse - target e_{ABN} 0.5 - noise 0%,5% - pattern 3

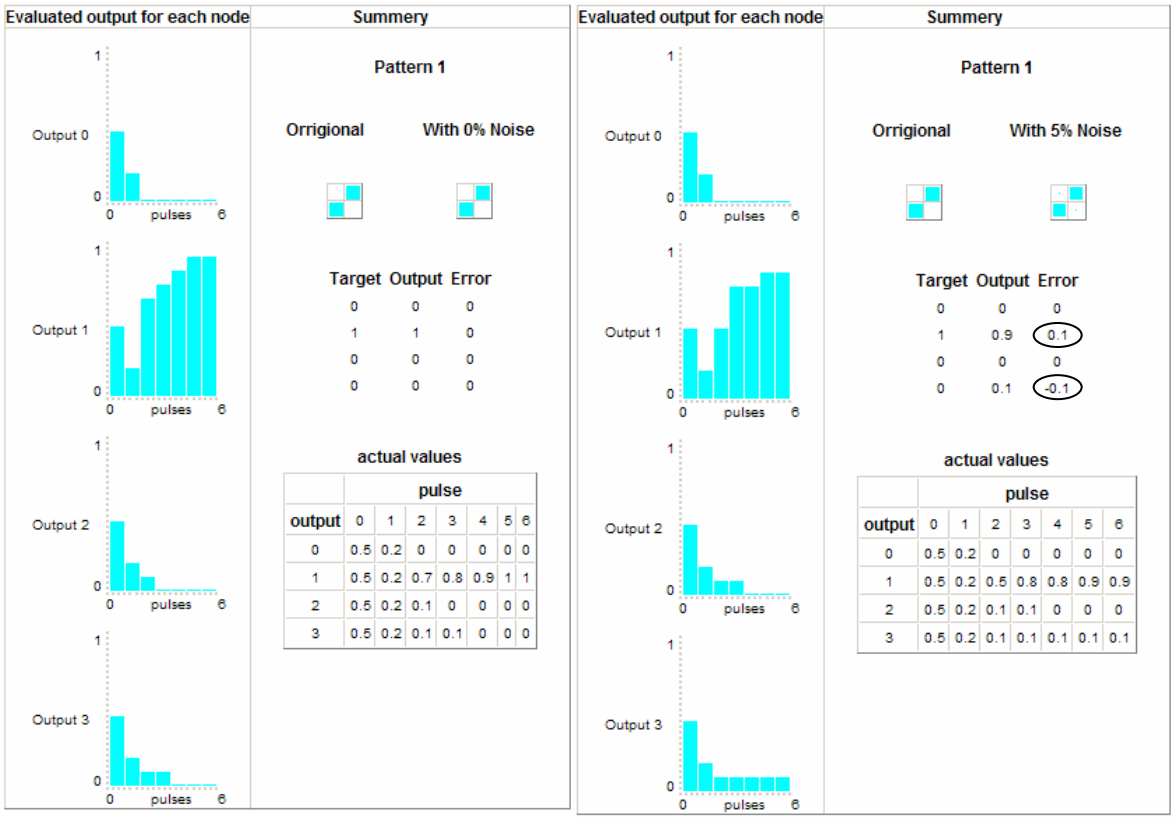


Figure G.25 – ABN_w-BP output pulse - target e_{ABN} 0.05 - noise 0%,5% - pattern 1

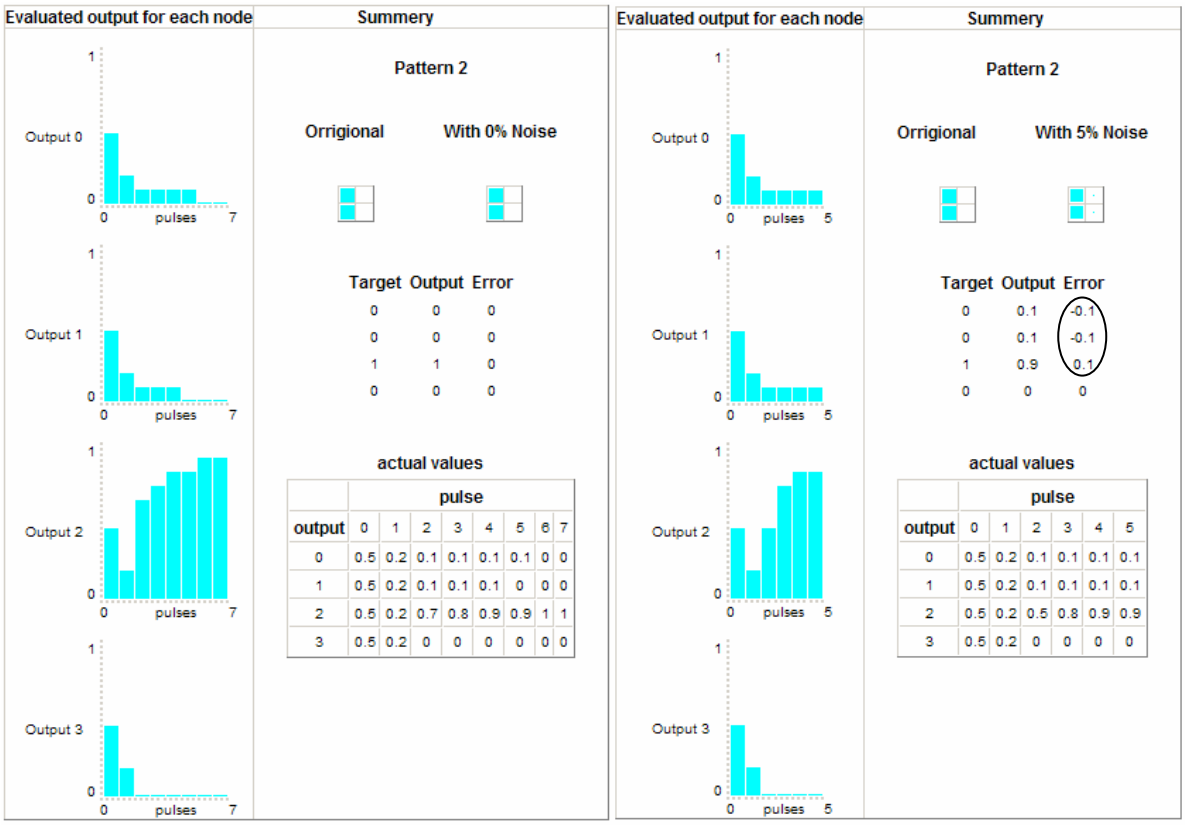


Figure G.26 – ABN_w-BP output pulse - target e_{ABN} 0.05 - noise 0%,5% - pattern 2

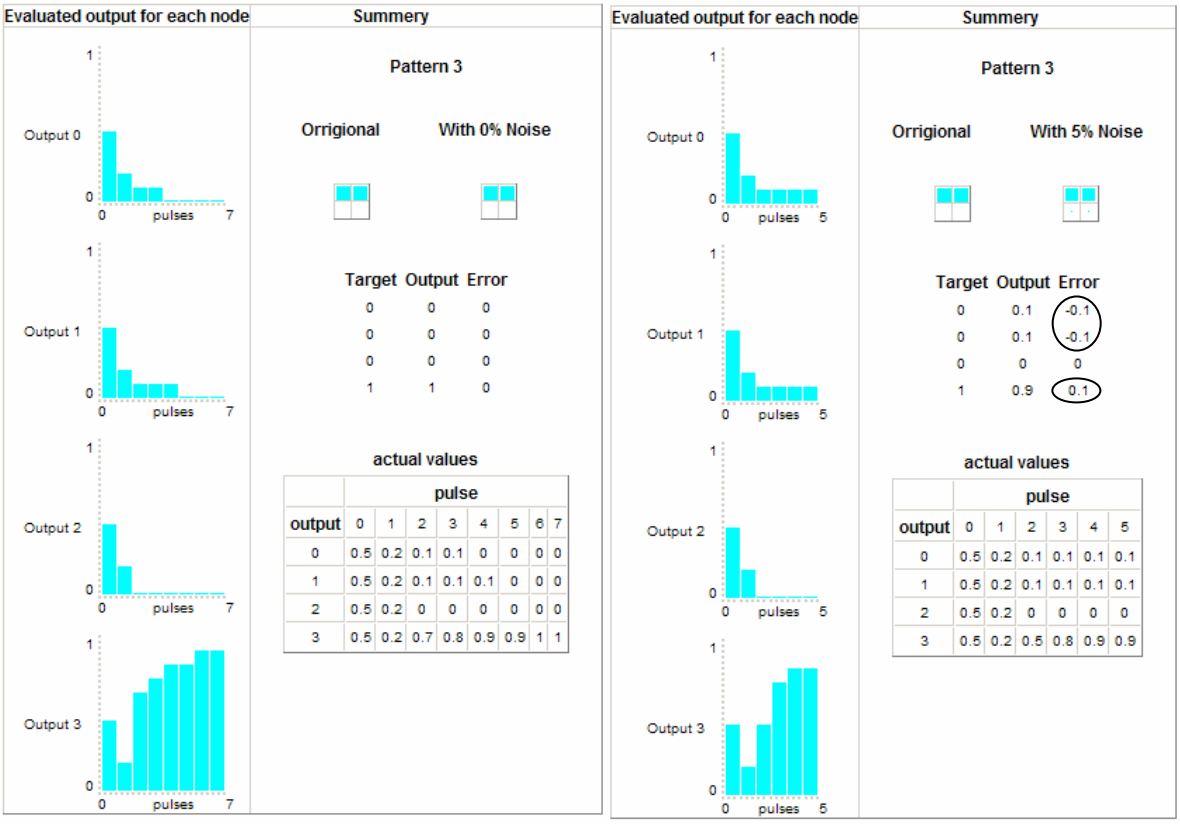


Figure G.27 – ABN_w-BP output pulse - target e_{ABN} 0.05 - noise 0%,5% - pattern 3

From 8.5.2 Successful ABN_F – GA Implementation

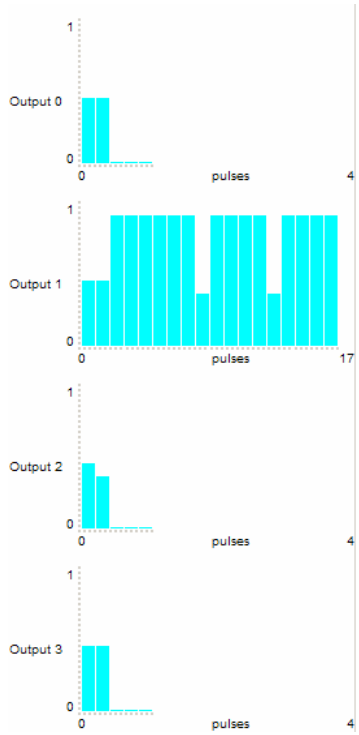


Figure G.28 – ABN_F-GA output pulse – pattern 1

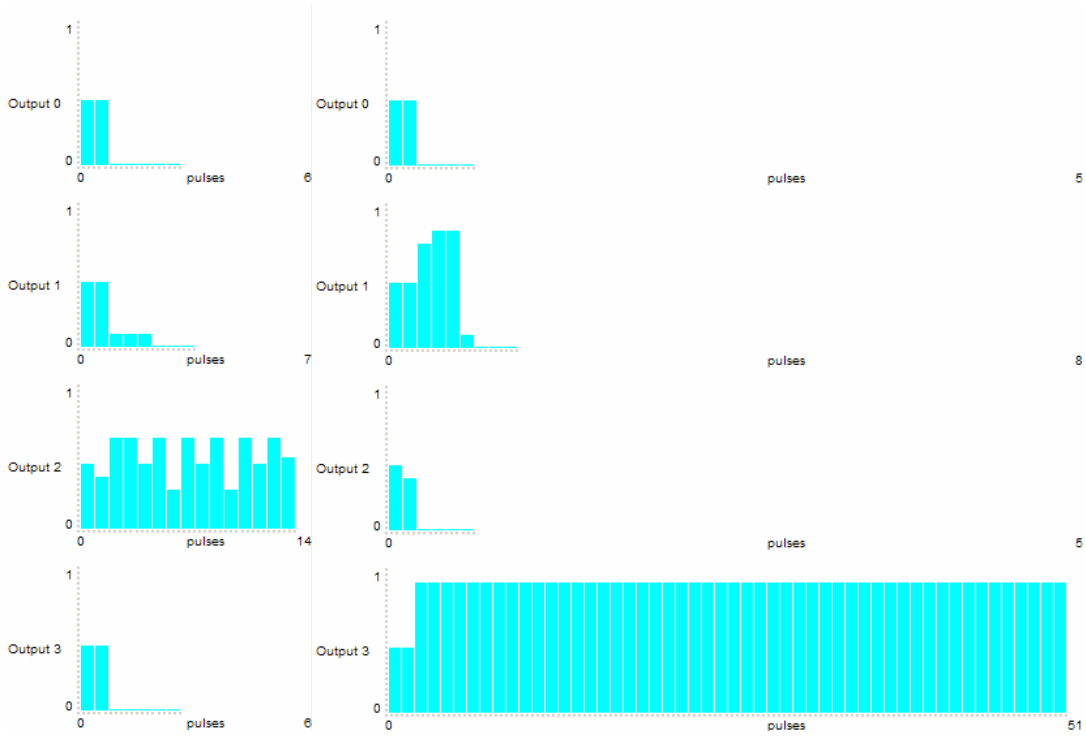


Figure G.29, G.30 – ABN_F-GA output pulse – patterns 2,3

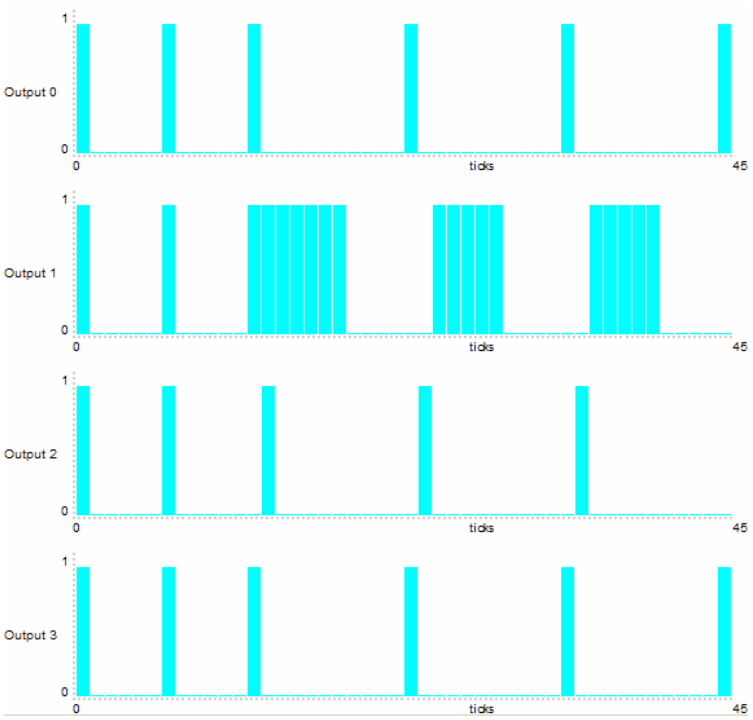


Figure G.31 – ABN$_F$-GA output ticks – pattern 1

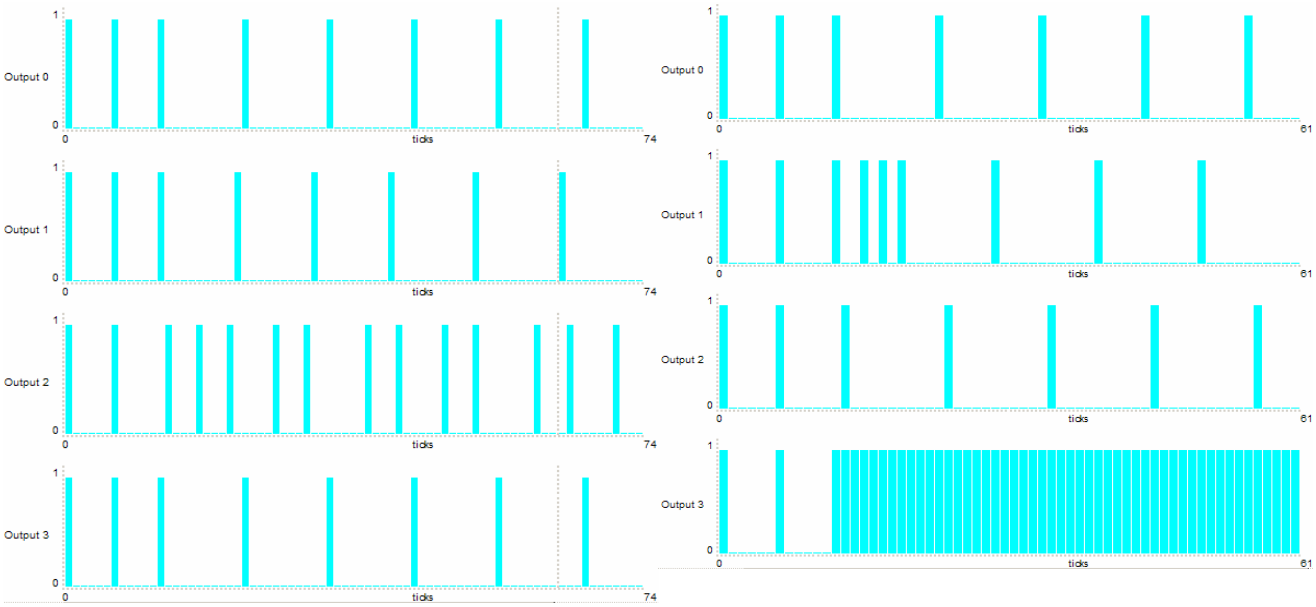


Figure G.32, G.33 – ABN$_F$-GA output ticks – patterns 2,3

From 8.5.8 ABN_F – Trained using a GA - Noise Tolerance

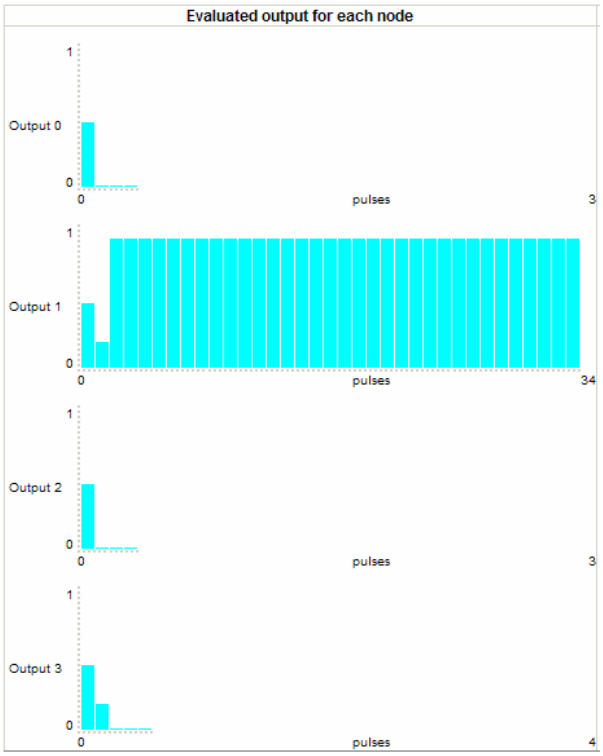


Figure G.34 – ABN_F-GA output pulse - target e_{ABN} 0.5 - noise 0% - pattern 1

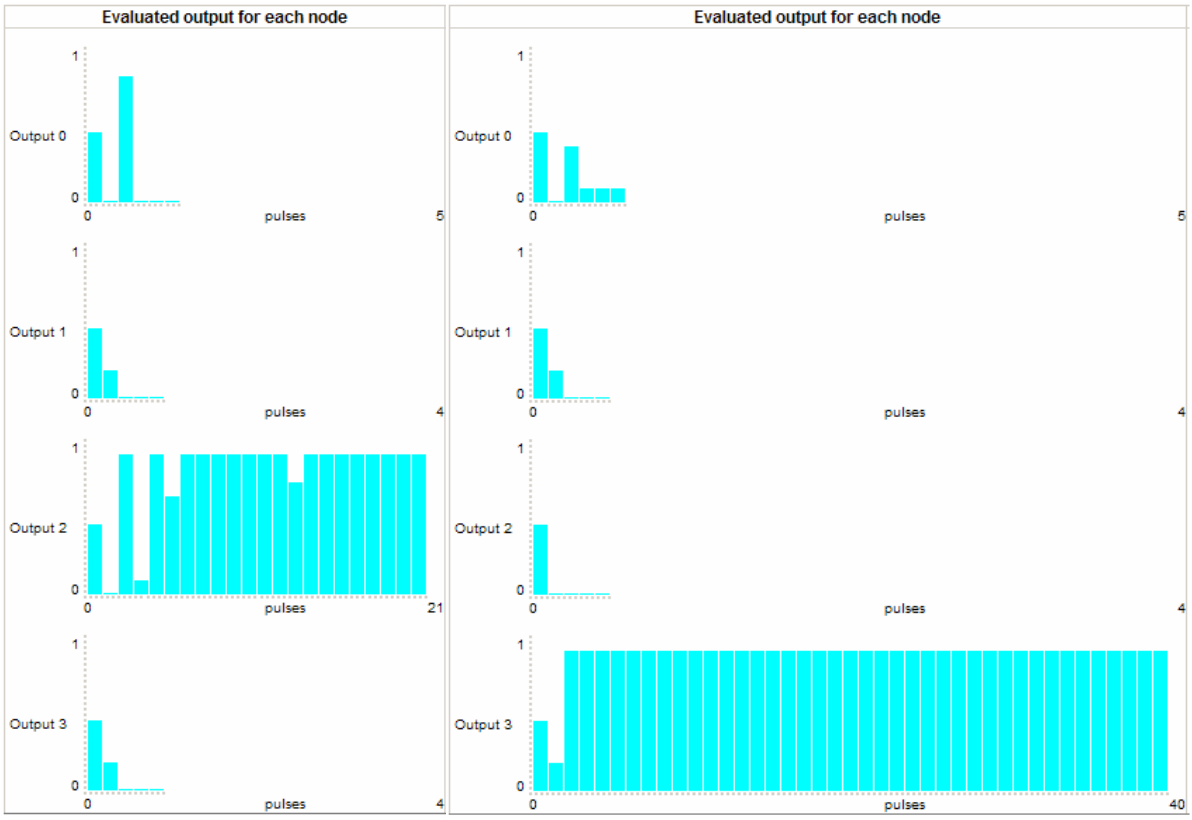


Figure G.35, G.36 – ABN_F-GA output pulse - target e_{ABN} 0.5 - noise 0% - patterns 2,3

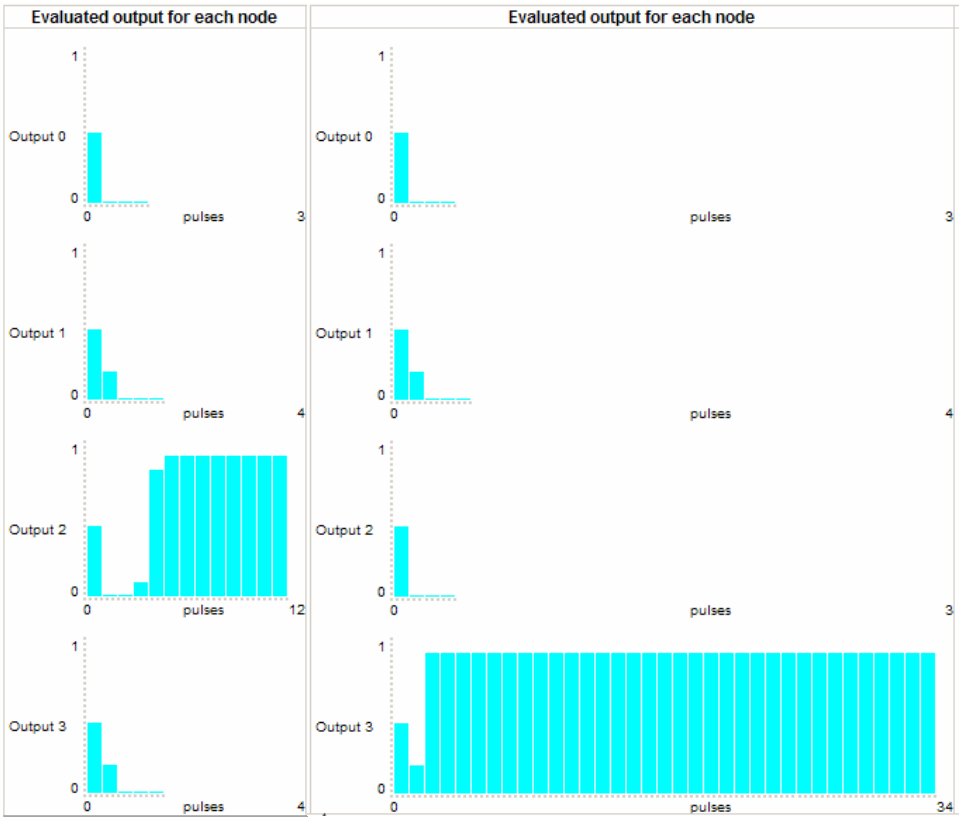


Figure G.37, G.38 – ABN_F-GA output pulse - target e_{ABN} 0.5 - noise 15% - patterns 2,3

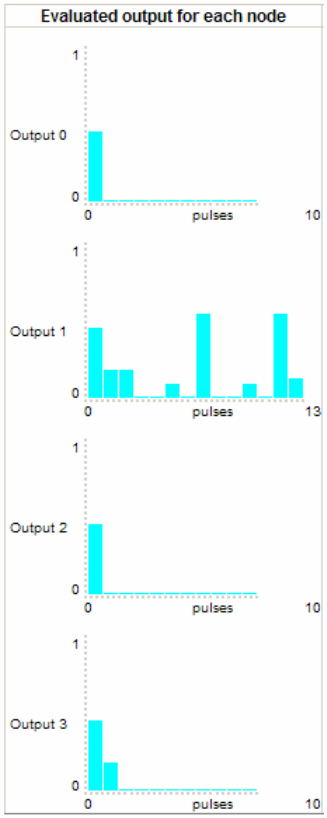


Figure G.39 – ABN_F-GA output pulse - target e_{ABN} 0.5 - noise 25% - pattern 0

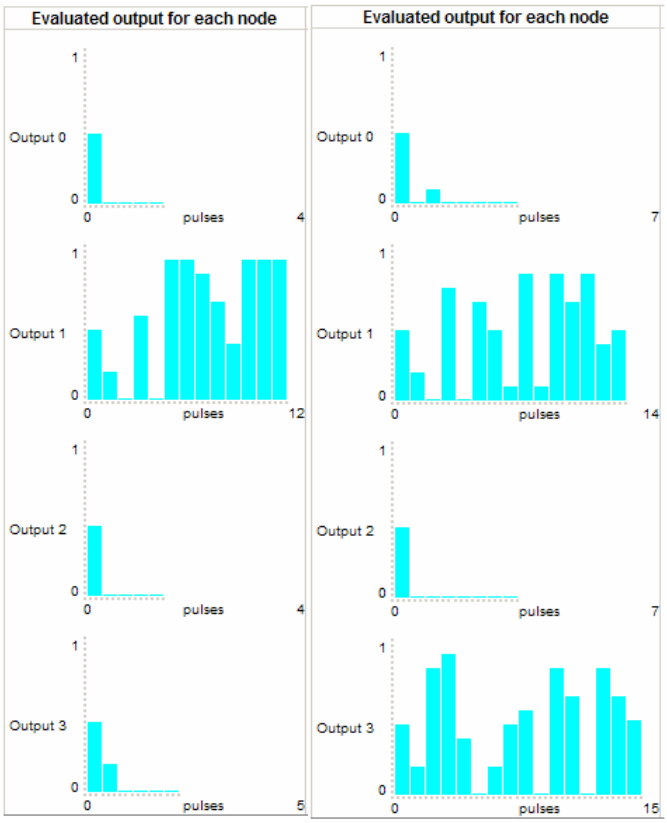


Figure G.40, G.41 – ABN_F-GA output pulse - target e_{ABN} 0.5 - noise 36% - patterns 2,3