A Review of Artificial Neural Networks

by

H R Maier

Department of Civil and Environmental Engineering

The University of Adelaide

Research Report No. R 131

August 1995
ABSTRACT

Artificial neural networks (ANNs) are a form of computing inspired by the functioning of the brain and nervous system. Although the first artificial neuron was developed in 1943, the true capabilities of ANNs have only become apparent since backpropagation networks were developed in 1986. The number of uses for ANNs is increasing rapidly, and the world market for the commercial use of ANNs has expanded from virtual non-existence in 1985 to be in the order of tens of millions of dollars at present. It is expected to reach approximately $5 billion by the year 2000.

Traditionally, ANNs have been used to carry out cognitive tasks performed naturally by the brain, including recognising a familiar face, learning to speak and understand a natural language, identifying handwritten characters and determining that a target seen from different angles is in fact the same object. However, in recent years ANNs have been used extensively for optimisation and forecasting, and have proved to be an attractive alternative to more conventional methods.

This report gives an introduction to ANNs and provides guidance to potential users of ANNs in terms of different network architectures, potential applications of ANNs and ways of implementing them. The main focus in this report is on the use of ANNs for the prediction of environmental, hydrological and water resources data. Guidelines for the design of back-propagation networks are reviewed in detail, as this is the network type most suited to such applications.
# TABLE OF CONTENTS

1. INTRODUCTION ................................................................. 1

2. ARTIFICIAL NEURAL NETWORKS IN PERSPECTIVE ........... 2
   2.1 Parallel Versus Serial Computing .................................. 2
   2.2 ANNs as a Form of Artificial Intelligence ..................... 2
      2.2.1 Expert systems ................................................. 2
      2.2.2 Artificial neural networks ................................... 3

3. HISTORY OF NEURAL COMPUTING ................................... 4

4. NATURAL NEURAL NETWORKS ............................................. 6

5. ARTIFICIAL NEURAL NETWORKS ....................................... 7
   5.1 Introduction .......................................................... 7
   5.2 Operation .................................................................... 8
      5.2.1 Connection weights ........................................... 10
      5.2.2 Threshold ......................................................... 10
      5.2.3 Transfer functions ............................................ 10
         5.2.3.1 Threshold functions .................................. 11
         5.2.3.2 Hard limiters ............................................. 11
         5.2.3.3 Continuous functions .................................. 12
      5.2.4 Learning .......................................................... 13
         5.2.4.1 Unsupervised learning ................................ 13
         5.2.4.2 Supervised learning ................................... 14
         5.2.4.3 Graded learning ......................................... 16
         5.2.4.4 Hybrid learning ......................................... 16
         5.2.4.5 Non adaptive learning ................................. 16
   5.3 Architecture ............................................................ 17
   5.4 Performance ............................................................ 17

6. IMPLEMENTATION OF ARTIFICIAL NEURAL NETWORKS .... 17
   6.1 Software Simulation .................................................. 17
   6.2 Hardware Accelerators .............................................. 17
   6.3 Optical Processors .................................................... 18
   6.4 Direct Hardware Implementations ................................. 18
      6.4.1 Analog neural network architecture networks ............ 18
      6.4.2 Digital neural network architecture networks .......... 19
      6.4.3 Hybrid neural network architectures .................... 21
7. DIFFERENT NEURAL NETWORK ARCHITECTURES ...........................................21
  7.1 Network Classification ........................................................................ 21
  7.2 Multilayer Feed Forward Neural Networks I: Delta Rule
      Learning ................................................................. 24
      7.2.1 Perceptron networks ...................................................... 24
      7.2.2 ADALINE and MADALINE networks .............................. 27
  7.3 Multilayer Feed Forward Neural Networks II: Optimising
      and Random Learning Methods .................................................. 28
      7.3.1 The Boltzmann Machine ................................................. 28
      7.3.2 Directed random search networks .................................. 31
  7.4 Laterally-Connected, Auto Associative Networks ............................... 32
      7.4.1 Hopfield / Tank networks .............................................. 32
      7.4.2 Brain-State-In-A-Box networks ..................................... 34
  7.5 Vector-Matching Networks .................................................................. 35
      7.5.1 Learning Vector Quantisation (LVQ) networks ................. 36
      7.5.2 Self-Organising Maps ...................................................... 37
  7.6 Feedforward/Feedback (Resonating) Hetero-Associative
      Networks ............................................................... 38
      7.6.1 Carpenter / Grossberg Adaptive Resonance
            Theory (ART) networks .................................................. 38
      7.6.2 Bidirectional Associative Memory (BAM)
            networks ............................................................. 40
      7.6.3 Recirculation networks ......................................................... 41
  7.7 Multilayer Cooperative / Competitive Networks ................................. 42
  7.8 Hybrid Networks ............................................................................ 42
      7.8.1 Hamming networks ........................................................ 43
      7.8.2 Counter-propagation networks ....................................... 44
      7.8.3 Probabilistic neural networks ........................................... 45

8. BACK-PROPAGATION NETWORKS ......................................................... 46
  8.1 Learning ....................................................................................... 48
      8.1.1 The delta rule ............................................................. 48
      8.1.2 The generalised delta rule .............................................. 48
      8.1.3 Use of training data ........................................................ 52
      8.1.4 Stopping criteria .......................................................... 53
  8.2 Measuring Network Performance ...................................................... 54
  8.3 Network Design .............................................................................. 54
      8.3.1 Network geometry ............................................................ 55
      8.3.2 Initial weight distribution .................................................. 59
      8.3.3 Learning rate .............................................................. 59
  8.4 Deficiencies of the Back-Propagation Algorithm ................................ 59
  8.5 Ways of Improving the Back-Propagation Algorithm ...................... 60
8.5.1 Momentum ................................................................. 60
8.5.2 Modifying the derivative .............................................. 60
8.5.3 Different error functions ............................................. 61
8.5.4 Cumulative update of weights ..................................... 61
8.5.5 Fast back-propagation .............................................. 62
8.5.6 Dynamic learning rate .............................................. 62
8.5.7 Delta-Bar-Delta algorithm .......................................... 63
8.5.8 Adaptive back-propagation ....................................... 64
8.5.9 Dynamic training sets and weighting factors ................. 65
8.5.10 Functional link networks .......................................... 65
8.5.11 Supplementary link networks ................................... 66
8.5.12 Radial basis functions ........................................... 67

9. APPLICATIONS .................................................................. 69
   9.1 Networks for Pattern Recognition ................................. 70
   9.2 Networks for Conceptualisation ................................. 71
   9.3 Networks for Filtering ............................................... 71
   9.4 Networks for Optimisation .......................................... 71
   9.5 Networks for Prediction ............................................... 73
      9.5.1 General applications ........................................... 73
      9.5.1.1 Economic time series prediction ......................... 73
      9.5.1.2 Chaotic time series prediction ......................... 74
      9.5.1.3 Predicting electrical power system loads ............ 75
   9.5.2 Applications in water engineering ............................ 75
      9.5.2.1 Estimation of water consumption / demand ............ 76
      9.5.2.2 Regional flood estimation ................................ 77
      9.5.2.3 Predicting salinity ........................................... 77
      9.5.2.4 Flow prediction ............................................. 79
      9.5.2.5 Runoff prediction ........................................... 79
      9.5.2.6 Prediction of nutrient concentrations ................. 79
      9.5.2.7 Forecasting minimum temperature ...................... 79
      9.5.2.8 Prediction of blue-green algae concentrations ...... 80
      9.5.2.9 Design of a reliable groundwater remediation strategy .......................................................... 80

10. SUMMARY .................................................................. 80

11. REFERENCES .................................................................. 82
APPENDIX A  NOTATION .................................................................................. 95
APPENDIX B  ABBREVIATIONS ........................................................................ 96
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Typical Structure of a Neuron</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>Typical ANN</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>Operation of a Typical Processing Element</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>An Example of a Threshold Function</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>An Example of a Hard Limiter</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>The Sigmoid Function</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>The Hyperbolic Tangent Function</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>Representation of weights and signals by stochastic pulse trains</td>
<td>20</td>
</tr>
<tr>
<td>9</td>
<td>Modifying the input signal by its connection weight</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>A Taxonomy of Neural Networks</td>
<td>23</td>
</tr>
<tr>
<td>11</td>
<td>Six Types of Network Architectures</td>
<td>24</td>
</tr>
<tr>
<td>12</td>
<td>Typical Perceptron Architecture</td>
<td>25</td>
</tr>
<tr>
<td>13</td>
<td>Decision Regions Formed by Various Networks</td>
<td>26</td>
</tr>
<tr>
<td>14</td>
<td>Typical Adaline/Madaline Architecture</td>
<td>27</td>
</tr>
<tr>
<td>15</td>
<td>Typical Boltzmann Machine Architecture</td>
<td>29</td>
</tr>
<tr>
<td>16</td>
<td>Marble on a Surface in a Box</td>
<td>30</td>
</tr>
<tr>
<td>17</td>
<td>Cauchy versus Boltzmann Distributions</td>
<td>31</td>
</tr>
<tr>
<td>18</td>
<td>Typical Hopfield Architecture</td>
<td>33</td>
</tr>
<tr>
<td>19</td>
<td>Typical Brain-State-in-a-Box Architecture</td>
<td>35</td>
</tr>
<tr>
<td>20</td>
<td>Weight Vectors During the Ordering Process</td>
<td>37</td>
</tr>
<tr>
<td>21</td>
<td>Typical ART Architecture</td>
<td>39</td>
</tr>
<tr>
<td>22</td>
<td>Typical BAM Architecture</td>
<td>41</td>
</tr>
<tr>
<td>23</td>
<td>Typical Recirculation Architecture</td>
<td>41</td>
</tr>
<tr>
<td>24</td>
<td>Typical Hamming Architecture</td>
<td>44</td>
</tr>
<tr>
<td>25</td>
<td>Typical Counter-Propagation Architecture</td>
<td>45</td>
</tr>
<tr>
<td>26</td>
<td>Typical Back-Propagation Architecture</td>
<td>46</td>
</tr>
<tr>
<td>27</td>
<td>The Sigmoidal Transfer Function and its Derivative</td>
<td>47</td>
</tr>
<tr>
<td>28</td>
<td>A Typical Error Surface</td>
<td>49</td>
</tr>
<tr>
<td>29</td>
<td>The Phenomenon of Overtraining</td>
<td>53</td>
</tr>
<tr>
<td>30</td>
<td>The Phenomenon of Overfitting</td>
<td>56</td>
</tr>
<tr>
<td>31</td>
<td>Illustration of One Iteration Step</td>
<td>63</td>
</tr>
<tr>
<td>32</td>
<td>An Example of a Radial Basis Transfer Function</td>
<td>68</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 1: Advantages and disadvantages of analog neural networks ..........19
Table 2: Advantages and disadvantages of digital neural networks ..........21
1. INTRODUCTION

Artificial neural networks (ANNs) are a form of computing inspired by the functioning of the brain and nervous system. Although the first artificial neuron was developed in 1943 (McCulloch and Pitts, 1943), the true capabilities of ANNs have only become apparent since backpropagation networks were developed in 1986 (Rumelhart et al., 1986). The number of uses for ANNs is increasing rapidly, and the world market for the commercial use of ANNs has expanded from virtual non-existence in 1985 to be in the order of tens of millions of dollars at present (Hubick, 1992). It is expected to reach approximately $5 billion by the year 2000 (Hubick, 1992).

The brain has a novel way of processing information. It utilises vast numbers of neurons, whose individual processing capabilities are quite limited. These neurons are linked by, and transmit information via, electrochemical pathways. The resulting neural network is able to learn by adjusting the strength of the pathways. As a result, parallel computation can occur and memory is represented by the strength of the connections between neurons.

Artificial neural networks (ANNs) are modelled on the gross structure of the brain outlined above. Neurons are represented by simple processing elements, which are generally arranged in layers; an input layer, an output layer and one or more layers in between called hidden layers. The processing elements are fully or partially linked by weighted connections. Different values of weights represent connections of varying strength. These weights are adjusted using a learning rule.

Artificial neural networks are able to determine relationships between input data and corresponding output data. When presented with input-output paired training examples, artificial neural networks adjust their connection weights on the basis of the training samples and discover the rules governing the association between variables.

Traditionally, ANNs have been used to carry out cognitive tasks performed naturally by the brain, including recognising a familiar face, learning to speak and understand a natural language, identifying handwritten characters and determining that a target seen from different angles is in fact the same object. However, in recent years ANNs have been used extensively for forecasting and have proved to be an attractive alternative to more conventional forecasting techniques.

ANNs are particularly suited to modelling environmental, hydrological and water resources systems, as they are able to perform well when a large
amount of data is available, when the input data are noisy, when the
distribution of the data is unknown, when a large number of attributes
describe the system being modelled, when uncertainty exists about the rules
governing the relationship between the variables and when non-linearities are
suspected.

2. ARTIFICIAL NEURAL NETWORKS IN PERSPECTIVE

2.1 Parallel Versus Serial Computing

ANNs are a type of parallel computer, which differ from the more traditional
serial (Von Neumann) computers.

Von Neumann computers are the type used most widely today. Their
operations are controlled by a single central processing unit (CPU), which
holds the computer's memory and processes information in a sequential
manner. The sequential manner in which data is processed limits the overall
processing speed of the computer, which can lead to a "bottle neck" of
information transfer if a large amount of information has to be processed
(Hubick, 1992).

Parallel computers consist of a number of smaller processing units that are
linked together. This allows information to be processed in a parallel manner,
which is useful when many pieces of information are required to carry out a
particular operation (e.g. when the brain makes a decision).

2.2 ANNs as a Form of Artificial Intelligence

Artificial intelligence was first introduced in the 1950s and includes expert
systems, natural language, ANNs, voice recognition, machine translation,
intelligent text or media and other related activities (Hubick, 1992). A
comparison of expert systems and artificial neural networks is given below.

2.2.1 Expert systems

Rule-based expert systems attempt to duplicate the behaviour of a human
expert in a particular field of knowledge. The knowledge of an expert system
is derived from a large database of knowledge which is created by a human
expert. When using an expert system, this knowledge base is interrogated to
obtain the desired answer, as though it were an expert on the subject in
question (Schloldt, 1987; Hancox et al., 1990). The expert system also reveals
the rules that were used to arrive at a particular conclusion (Hubick, 1992). The use of expert systems is well suited to serial computers.

The major advantage of expert systems is that they allow the user to check the logic used by the expert system to arrive at its conclusion (Hubick, 1992). However, there are a number of disadvantages associated with using expert systems (Hubick, 1992), including:

- A set of rules needs to be developed for each individual problem.
- If one aspect of a particular problem changes, the rules have to be adjusted. This may lead to a chain reaction, as some rules affect others and so on.
- They do not tolerate any errors in the rules.
- The rules need to be converted to the appropriate computer code.
- They are unable to cope with breakdowns or unexpected events - all possible outcomes need to be taken into account when the knowledge base is established.
- They are expensive to develop commercially.
- There is no methodology for developing the knowledge base.

2.2.2 Artificial neural networks

ANNs process information in a parallel manner. They develop their own set of rules as they learn “by example” and are therefore well suited for complex processes for which explicit rules are unknown (Hubick, 1992). The knowledge of ANNs is distributed and is stored in the interconnections between the processing elements. This is in contrast to expert systems, whose knowledge is represented by its rules.

The advantages of using artificial neural networks include (Jones and Hoskins, 1987; Lippmann, 1987; Daniell, 1991; Burke and Ignizio, 1992; Burke, 1991; NeuralWare, Inc, 1991; Kimoto and Asakawa, 1990; Josin, 1987; Saito and Nakano, 1990; Hecht-Nielsen, 1988; Maren et al., 1990; Hubick, 1992):

- High processing speeds as a result of their parallel nature.
- They have the ability to deal with bad or missing data.
- They are very robust or fault tolerant. If some processing elements, or the links between them, are damaged or disabled, the overall performance of the network is not degraded significantly. This property arises from the fact that information is distributed and not contained in one place (e.g. the CPU in Von Neumann computers).
- They generate their own rules internally by learning from examples. This gives neural networks an advantage when dealing with complex problems, for which the underlying rules or algorithms are unknown. Even if an algorithm is known, it is often too computationally intensive to generate solutions in an acceptable time frame.
• They have the ability to produce continuously graded inputs or outputs. This can be used to represent the intensity of an input feature or the certainty of a classification.
• They are virtually unaffected by noisy input data.
• They can be inserted into existing technology with ease. They can be designed quickly and the direct hardware implementation of artificial neural networks is cost effective. As a result, artificial neural networks can be used for incremental system improvement and upgrade.
• They have good generalisation ability.

The disadvantages of ANNs include (Hubick, 1992):
• There is no set methodology for determining the optimal structure of the neural network for a particular problem.
• There is no means for determining the rules used by the network to arrive at a particular decision.
• They are not suited to problems where the relationship between the model inputs and outputs is deterministic.

3. HISTORY OF NEURAL COMPUTING

The history of neural computing is discussed by various authors (NeuralWare, Inc., 1991; Daniell, 1991; Vemuri, 1988; Maren et al., 1990; Hecht-Nielsen, 1990).

At the inception of the field of computing, there were two philosophically opposing views about what the purpose of computing should be. One philosophy stated that the mind, as well as computers, are symbol manipulating systems and that the aim of computing is to represent the computational capabilities of the brain. The opposing view was that the aim of computing is to model the structure of the brain itself (Vemuri, 1988). Neural computing follows the latter philosophy.

McCulloch and Pitts (1943) introduced one of the first theoretical models of artificial neurons, the M-P neuron. M-P neurons feature binary inputs and outputs assuming values 0 or 1 and an internal threshold level. Operation is such that the neuron output is one if the total input is greater than or equal to the threshold level. Individual M-P neurons can be interconnected to form artificial neural networks.

Hebb (1949) proposed that the connection strength between neurons changes as learning occurs. Hebb also postulated that the connection between two neurons gets stronger if the two neurons are excited simultaneously.
The first conference on artificial intelligence was held at Dartmouth in 1956. The fields of artificial intelligence and neural computing were launched at this conference (NeuralWare, Inc., 1991).

The first major research project in neural computing was conducted by Rosenblatt in 1958. Rosenblatt (1958, 1961) showed that if M-P neurons are linked with connections of adjustable strength, they can be "trained" to perform pattern classifications. These networks are called "perceptrons" and combine McCulloch and Pitts' work with Hebb's postulate. Perceptrons are able to make limited generalisations and are able to perform classifications even when the input is noisy. Perceptron networks consist of an input sensory layer, a hidden layer and an output layer. Initially, all the connection weights are given arbitrary values producing an arbitrary response to a given input. The weights are adjusted during "training" to produce the desired output. Rosenblatt's work on perceptrons sparked great interest in the research area of neural computing and ANNs.

Widrow and Hoff (1960) developed the ADALINE (ADApative LINear Element) learning algorithm. The ADALINE is a variant of the perceptron and is also based on simple neuron-like elements.

The two-layer version of the ADALINE, the MADALINE (Multiple ADALINE), was introduced in 1962 (Widrow and Smith, 1963). The ADALINE and MADALINE were used in a variety of applications including weather prediction, character recognition, speech recognition and adaptive control.

Minsky and Papert (1969) published a critical review of perceptron networks called "perceptrons". They showed that single layer perceptrons cannot solve non-linear separable problems such as representing the Exclusive-OR function. As a result, funding for research into ANNs and neural computing decreased markedly and caused most research into this area to be abandoned. This is despite the fact that perceptrons with three layers of processing elements have the ability to implement any continuous function as proven by Kolmogorov (1957). The only obstacle to using multilayer perceptrons was the lack of a suitable learning rule to adjust the weights in the hidden layer. In spite of this drawback, some research into neural networks continued.

Grossberg (1976) developed the Adaptive Resonance Theory (ART) network, which is a neurally inspired computing model.

Kohonen (1982) introduced the Self Organising Feature Map (SOM). This model is based on the principle of competitive learning, where elements compete to respond to an input stimulus and the winner adapts itself to
respond more strongly to that stimulus. This results in an unsupervised
learning procedure, as the internal organisation of the network is only
dependent on the input stimulus.

Hopfield (1982, 1984) developed the Hopfield model, which sparked a
resurgence into research in the field of neural computing. This model consists
of interconnected processing elements which seek an energy minimum and
store information in their connections.

Anderson (1983) presented his Brain-State-in-a-Box model, which was based
on his earlier Linear Associator model. It is a distributed, associative model
based on Hebb's postulate.

The Boltzmann Machine was developed by Ackley et al. (1985). It is based
on the Hopfield model but incorporates stochastic updating procedures during
recall to allow the network to escape from local minima in the energy surface.
The Boltzmann Machine makes use of a search method called simulated
annealing.

Rumelhart et al. (1986) introduced the Back-propagation network, which is
the most widely used network today. The back-propagation learning rule
enables the weights in the hidden layer(s) to be adjusted successfully and
hence the back-propagation network is able to implement the Exclusive-OR
function.

The Bi-directional Associative Memory (BAM) network was designed by
Kosko (1987) and is similar to Grossberg's models.

Another resurgence in the field of neural network research is taking place at
present due to the development of new network topologies, new learning
algorithms, new analog VLSI implementation techniques and a growing
fascination with the functioning of the human brain (Lippmann, 1987).

4. NATURAL NEURAL NETWORKS

The structure and operation of natural neural networks (NNNs) is addressed
by several authors (NeuralWare, Inc., 1991; Vemuri, 1988; Hecht-Nielsen,

NNNs consist of tens of billions of nerve cells called neurons, which are
densely interconnected. Each neuron is connected to as many as 10,000
others. The cerebral cortex of the human brain is an example of a NNN.
Neurons essentially behave as microprocessors. Each neuron receives the combined output of many other neurons through input paths called dendrites. If this input signal is strong enough, the neuron is activated and produces an output which is transmitted through output structures called axons. This information transfer is electrochemical in nature.

The axon splits up and connects to the dendrites of many other neurons via junctions called synapses. The connection is chemical and is made possible by the release of neurotransmitters by the axon, which are received by the dendrites. The strength of the connection is modified as the brain learns and it depends on the amount of neurotransmitter released. The way neural networks process information combined with the action of the synapse results in the basic memory mechanism of the brain. The typical structure of a neuron is shown in Figure 1.

![Typical Structure of a Neuron](image)

Figure 1: Typical Structure of a Neuron (Source: NeuralWare, Inc., 1991)

5. ARTIFICIAL NEURAL NETWORKS

5.1 Introduction

ANNs are described by a number of authors (NeuralWare, Inc, 1991; Lippmann, 1987; Hecht-Nielsen, 1988; Josin, 1987). They are loosely based on the structure of NNNs, but exhibit only a very small portion of their capabilities. ANNs are similar to NNNs in that they consist of interconnected processing elements (neurons) and satisfy the "locality constraint", which means that processing elements are only allowed to receive information supplied locally. As a result, the input to a processing element can only be
directly affected by a node connected to its input path (Burke and Ignizio, 1992).

In ANNs, processing elements (nodes) are equivalent to neurons in natural neural networks. Processing elements are usually analog, non-linear, possess a small local memory and are slow compared with advanced digital circuitry. Individual processing elements are usually arranged in layers. Two of these layers, the input layer and the output layer, are connected to the environment. Data are presented to the network at the input layer and the response to the input is presented at the output layer. The layers in-between the input layer and the output layer are called hidden layers. Hidden layers enable the network to cater for non-linearities.

Nodes in the various layers are either fully or partially interconnected. Each connection has associated with it a particular adaptation coefficient or "weight" representing the synaptic strength of neural connections. A typical network is shown in Figure 2.

![Figure 2: Typical ANN (Source: NeuralWare, Inc., 1991)](image)

5.2 Operation

The operation of ANNs is described extensively by a number of authors (NeuralWare, Inc., 1991; Vemuri, 1988; Lippmann, 1987; Daniell, 1991; Maren et al., 1990; Hecht-Nielsen, 1988; Jones and Hoskins, 1987; Burke and Ignizio, 1992).

The propagation of data through the network starts with the presentation of an input stimulus at the input layer. The data then progress through, and are
operated on by, the network until an output stimulus is produced at the output layer.

Individual processing elements receive inputs from many other processing elements via weighted input connections. These weighted inputs are summed and an optional threshold value is added or subtracted producing a single activation level for the processing element. This activation level constitutes the argument of a transfer function, which produces the node output. This output is passed to the weighted input connections of many other processing elements. This process is shown for node "j" in Figure 3.

The following notation applies:

NP = the number of nodes in the previous layer
xi = the input from node i, i = 0, 1, ..., NP
wjij = the connection weight between nodes i and j
Ij = the activation level of node j
 F(·) = the transfer function
yj = the output of node j
Tj = the threshold for node j

Figure 3: Operation of a Typical Processing Element (Source: NeuralWare, Inc., 1991)
The governing equations are given by (NeuralWare, Inc., 1991; Maren et al., 1990):

\[ I_j = \sum_{i=1}^{N^p} w_{ji} x_i + T_j \quad \text{(summation)} \quad (1) \]

\[ y_j = F(I_j) \quad \text{(transfer)} \quad (2) \]

The performance of processing elements can be affected by changing transfer functions and adding new parameters or functions such as thresholds or gains.

Processing at each node occurs independently of the processing at all other nodes. At the same time, the processing done at each node effects the network as a whole as the output of one node becomes the input to many other nodes.

In a similar way to NNNs, ANNs learn by altering the connection strength between the processing elements. This is done by adjusting the weights on presentation of a set of training data using a learning rule. Once the learning phase is complete, the weights are "frozen".

### 5.2.1 Connection weights (Hecht-Nielsen, 1988; Jones and Hoskins, 1987)

Connection weights have the function of amplifying, attenuating or changing the sign of the input signal. A zero weight represents the absence of a connection and a negative weight represents an inhibitory relationship between two nodes. In general, the output of node \( i \) is multiplied by the weight of the connection between nodes \( i \) and \( j \) to produce the input signal to node \( j \). Hence connection weights represent the strength of the connection between two nodes. Weights are stored in the local memory of nodes and also hold the long-term memory of the network.

### 5.2.2 Threshold (Maren et al., 1990)

A threshold (bias) acts like another processing element that has a constant output. The effect of the threshold is to add a constant value to the summed input. The purpose of this is to scale the input to a useful range.

### 5.2.3 Transfer functions

The use of different transfer functions is addressed by various authors (Burke and Ignizio, 1992; Lippmann, 1987; Daniell, 1991; Maren et al., 1990; NeuralWare, Inc., 1991). Transfer functions are mathematical formulae that
give the output of a processing element as a function of its input signal. Transfer functions can take a variety of forms.

5.2.3.1 Threshold functions

Threshold functions only produce an output if the total input exceeds an internal threshold ($T_j$). The use of such functions is limited, as the output can only take on one of two values, as shown in Figure 4.

\[ F(I_j) = \begin{cases} 
+1 & \text{if } I_j \geq T_j \\
-1 & \text{otherwise}
\end{cases} \]  \hspace{1cm} (3)

5.2.3.2 Hard limiters

Hard limiters produce a linearly varying output between defined bounds of node activation (-$\alpha$, $\alpha$). Outside these bounds, the output is either -1 or 1. This is shown in Figure 5.

Figure 4: An Example of a Threshold Function (Source: Maren et al., 1990)

Figure 5: An Example of a Hard Limiter (Source: Maren et al., 1990)
\[
F(I_j) = \begin{cases} 
+1 & \text{if } I_j \geq -\alpha \\
\frac{I_j}{\alpha} & \text{if } -\alpha < I_j < \alpha \\
-1 & \text{if } I_j \leq \alpha
\end{cases}
\] (4)

5.2.3.3 Continuous functions

Continuous functions produce a continuous output as a function of the total input. Common transfer functions of that type include the sigmoid function and the hyperbolic tangent function, both of which are examples of squashing functions. A squashing function is a special type of transfer function that is differentiable and non-decreasing. It enables a trained network to perform non-linear interpolations, to provide a reasonable output in response to a range of inputs. It also limits the output for extreme input values. The sigmoid and hyperbolic tangent transfer functions are shown in Figures 6 and 7. Sometimes other smooth functions are used as transfer functions. The sine function, for example, enables the network to perform a mode decomposition. Usually all processing elements in a particular layer have the same transfer function.

![Figure 6: The Sigmoid Function](image)

\[
F(I_j) = \frac{1}{1 + e^{-I_j}}
\] (5)
Figure 7: The Hyperbolic Tangent Function

\[ F(I_j) = \frac{e^{(I_j)} - e^{-(I_j)}}{e^{(I_j)} + e^{-(I_j)}} \]  

(6)

### 5.2.4 Learning (Hecht-Nielsen, 1988)

Learning is the process in which the weights are adjusted in response to training data provided at the input layer and, depending on the learning rule, at the output layer. The learning process allows the network to adapt its response with time in order to produce the desired output. Weights are updated in accordance with a learning rule using an on-line or off-line algorithm (Jones et al., 1990). When using an off-line algorithm, all the training data need to be collected before learning can commence. On-line algorithms take into account training data as they become available and are more widely used than off-line algorithms. On-line algorithms fall into two categories. Conjugate gradient learning and the multidimensional Newton's method make use of the entire training data each time the weights are updated (Jones et al., 1990). The method used by Jones et al. (1990), on the other hand, only utilises the most recent training data, resulting in increased training speed and some reduction in network accuracy.

Learning procedures can be divided into five main categories, namely unsupervised learning, supervised learning, graded learning, hybrid learning and non-adaptive learning.

#### 5.2.4.1 Unsupervised learning

Unsupervised learning rules are discussed by Thorpe and Imbert (1989), Maren et al. (1990), NeuralWare, Inc. (1991) and Hecht-Nielsen (1990). In unsupervised learning, the network is only presented with input stimuli.
There is no feedback from the environment to say what the corresponding output should be or whether the output is right or wrong. The network itself discovers patterns, features, regularities or categories in the input space and reorganises itself internally so that individual hidden nodes correspond to a specific input or group of closely related inputs. Such groupings represent clusters in the input space corresponding to real life concepts. There are two different kinds of unsupervised learning, including unsupervised Hebbian learning and unsupervised competitive learning. In unsupervised Hebbian learning, several output units are often active together, whereas in unsupervised competitive learning, the processing elements compete with each other and only the winner is active. Unsupervised learning is similar to the way learning takes place in NNNs.

5.2.4.2 Supervised learning

Supervised learning rules are discussed by a number of authors, including Vemuri (1988), NeuralWare, Inc. (1991), Hecht-Nielsen (1988, 1990) and Maren et al. (1990). In supervised learning, the network is presented with an input stimulus as well as the desired response to that stimulus. The actual output of the network is then compared with the desired output. The aim of learning is to find a set of weights that minimises the mean squared error between the actual output and the desired output. The global error function at discrete time $t$, $E^G(t)$, is given by (Vemuri, 1988):

$$E^G(t) = \frac{1}{2} \sum_{k=1}^{N^O} (o^p_k(t) - o^d_k(t))^2$$  \hspace{1cm} (7)

where

- $o^p_k(t)$ = the predicted output at discrete time $t$
- $o^d_k(t)$ = the desired (actual, historical) output at discrete time $t$
- $N^O$ = the number of nodes in the output layer

Initially, the weights are assigned arbitrary values. The weights are then updated systematically using a learning rule. The weight update equation typically takes the form

$$w_{ji}(t+1) = w_{ji}(t) + \Delta w_{ji}$$  \hspace{1cm} (8)

There are several methods for finding the weight increment, $\Delta w_{ji}$ including the following:
Hebbian learning (NeuralWare, Inc., 1991; Hecht-Nielsen, 1990)

The basis for Hebbian learning is a postulate made by Hebb in 1949, which states: "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased" (Thorpe and Imbert, 1989).

The implications for ANNs are that the weight between two processing elements is incremented if both processing elements are activated simultaneously. The weights are updated in accordance with the following equation (Vemuri, 1988)

\[
\Delta w_{ji} = a_1 y_i y_j
\]  

(9)

where

\[ a_1 = \text{a constant of proportionality} \]

Grossberg learning

Grossberg's method is another weight update procedure that is biologically plausible. The weight update equation is given as follows (Vemuri, 1988):

\[
\Delta w_{ji} = a_2 f(y_j - w_{ji})
\]  

(10)

where

\[ a_2 = \text{a constant of proportionality} \]

Gradient descent learning (Vemuri, 1988; Maren et al., 1990)

As each combination of weights produces a different error, an error surface exists as a function of the weight space. The gradient descent method results in weights being changed in the direction of steepest descent down the error surface. As a result, the weight update is carried out in the direction that yields the maximum error reduction. The size of the step taken down the error surface is determined by a learning rate, \( \eta \).

\[
\Delta w_{ji} = \eta \frac{\partial E^G}{\partial w_{ji}}
\]  

(11)

There is little evidence that this type of learning occurs in NNNs (Thorpe and Imbert, 1989).
Competitive learning (NeuralWare, Inc., 1991)

In this type of learning, a competition occurs between all processing elements. The processing element that produces the strongest response to a particular input is the winner and is adjusted to become more like the input.

5.2.4.3 Graded learning

Graded learning is discussed by Maren et al. (1990), NeuralWare, Inc. (1991) and Hecht-Nielsen (1988, 1990). This type of learning is similar to supervised learning. Rather than receiving the correct response to a given stimulus, the network receives a score or grade to see how well it is performing. This reinforcement signal is only evaluative, not instructive. Graded learning is used for control and optimisation problems where it is impossible to know the correct response.

5.2.4.4 Hybrid learning

Sometimes it is advantageous to combine supervised and unsupervised learning techniques in the same network. The most common method of hybrid learning is to have one layer using an unsupervised learning rule followed by a network using a supervised learning rule (Maren et al., 1990). Although the tasks performed by such a network could usually be achieved using a network that learns purely in a supervised manner, using a hybrid learning scheme speeds up the learning process considerably.

5.2.4.5 Non adaptive learning (Burke and Ignizio, 1992; Looi, 1992)

In this type of learning, weights are not changed adaptively, but are derived initially from an energy function. This energy function forms a representation of the problem being considered and contains information specific to that problem. While weights are fixed, node states change until the energy function reaches a steady state. The energy function must be a Liapunov function, for which lower values correspond to better solutions to the problem being represented. One such energy function, the generic Hopfield function, is given below (Burke and Ignizio, 1992).

\[ E^N = -0.5 \left( \sum_i \sum_j w_{ji} x_i y_j + \sum_i T_i x_i \right) \]  

(12)
5.3 Architecture

Network architecture describes the way individual processing elements are joined to form neural networks. The factors which affect network architecture include (Maren et al., 1990):
1. The number of layers (e.g. single layer, bi-layer or multi-layer)
2. The number of processing elements per layer
3. The type of connection between individual processing elements (e.g. feed forward connections, feedback connections, lateral connections)
4. The degree of connectivity between processing elements

5.4 Performance

The performance of artificial neural networks is affected by three factors (Vemuri, 1988; Lippmann, 1987; Josin, 1987):
1. Node characteristics
2. Learning rules
3. Network architecture

6. IMPLEMENTATION OF ARTIFICIAL NEURAL NETWORKS

There are a number of ways of implementing artificial neural networks, some of which are discussed below.

6.1 Software Simulation

The operation of artificial neural networks can be simulated on conventional computers using software. Special computing languages have been developed which enable the user to describe neural network architectures. The AXON neural networks description language is one such a language (Hecht-Nielsen, 1990). A list of commercially available software simulators is given by Taylor (1993). The major disadvantage of using software simulators is that they do not make use of the inherent processing speed resulting from the parallel nature of ANNs, as processing is carried out sequentially.

6.2 Hardware Accelerators

Hardware accelerators can be used in conjunction with software simulators. The use of hardware accelerators is ideally suited to the parallel nature of neural networks. Hardware accelerators improve the processing speed of simulated networks to an extent where they can solve problems in real time.
Examples of such accelerators include the Mark III and Mark IV, the Network Emulation Processor (NEP), the ANZA and ANZA Plus and the Delta-II Floating Point Processor (NeuralWare, Inc., 1991).

6.3 Optical Processors

Optical processors can also be used in the implementation of artificial neural networks and are discussed by Maren et al. (1990) and NeuralWare, Inc. (1991). Since optical processors use light, they achieve a high density of channels as no insulation is needed. In fact, the resulting density is four times as high as that obtained with electrical cables. Another advantage using optical processors is that there is no interference.

6.4 Direct Hardware Implementations

Direct hardware implementations are discussed by various authors (NeuralWare, Inc, 1991; Daniell, 1991; Hecht-Nielsen, 1988 and 1990; Maren et al, 1990).

Direct hardware implementation of artificial neural networks is desirable, as it makes full use of the parallelism of neural networks. The resulting advantages include improved processing speed and increased maximum network size. Direct hardware implementation can be achieved with the use of neurocomputers.

Neurocomputers allow parallel processing and hence implement neural networks efficiently and in a cost-effective manner. Neurocomputers take the form of co-processor boards which plug into standard serial computers to enhance their parallel processing capabilities. Neurocomputers feature silicon chips that contain neural network architectures. Neurocomputers are of the analog, digital or hybrid type. The performance of neurocomputers can be assessed using criteria such as capacity, speed, cost, accuracy, generality, software interface provisions and configuration provisions. A list of available neurocomputers is given by Taylor (1993).

6.4.1 Analog neural network architecture networks

Analog neural networks are continuous, non-linear, dynamic systems. Their electronic circuits are made up of a combination of linear and non-linear devices including transistors, capacitors and resistors (Maren et al., 1990). The advantages and disadvantages of analog neural networks are summarised in Table 1.
<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>High noise immunity</td>
<td></td>
</tr>
<tr>
<td>Speed of operation</td>
<td></td>
</tr>
<tr>
<td>Precise computation</td>
<td></td>
</tr>
<tr>
<td>Readily designed using existing tools</td>
<td></td>
</tr>
<tr>
<td>Can include programmable components</td>
<td></td>
</tr>
<tr>
<td>Easy to multiplex/demultiplex</td>
<td></td>
</tr>
<tr>
<td>Can store both fixed and adaptive weights</td>
<td></td>
</tr>
<tr>
<td>Lack of thermal stability</td>
<td></td>
</tr>
<tr>
<td>Noise</td>
<td></td>
</tr>
<tr>
<td>Interconnection problems</td>
<td></td>
</tr>
<tr>
<td>Limited accuracy</td>
<td></td>
</tr>
<tr>
<td>Difficult to test</td>
<td></td>
</tr>
<tr>
<td>Hard to build simple components (resistors &amp; capacitors)</td>
<td></td>
</tr>
<tr>
<td>Lack of design tools</td>
<td></td>
</tr>
<tr>
<td>Non-uniform processing</td>
<td></td>
</tr>
<tr>
<td>Difficult to mass-produce chips with predetermined weight matrices</td>
<td></td>
</tr>
</tbody>
</table>

An example of an analog neural network chip is the Intel M64 Electrically Trainable Neural Network Chip (Maren et al., 1990).

### 6.4.2 Digital neural network architecture networks
(NeuralWare, Inc., 1991)

The success of digital neural networks stems from the fact that they are able to implement the sigmoid activation function using standard digital components. This network architecture can also be placed on a silicon chip, which can be produced cheaply and achieves high connection densities per silicon area. Processing speeds of up to 100,000 patterns per second have been achieved. This high speed derives from the fact that weight and signal values are represented by stochastic pulse trains, the magnitude of which is given by the firing frequency as shown in Figure 8. The product $w_{ji}x_j$ is achieved by logically AND-ing the pulse trains together as shown in Figure 9.

The summation of the weighted inputs and the action of a sigmoidal transfer function on that input is combined into one operation. This operation is implemented by logically OR-ing the weighted inputs. The connection weights can be obtained using the back-propagation algorithm and its variations.
Figure 8: Representation of weights and signals by stochastic pulse trains
(Source: NeuralWare, Inc., 1991)

Figure 9: Modifying the input signal by its connection weight
(Source: NeuralWare, Inc., 1991)

The advantages and disadvantages of digital neural networks are summarised in Table 2.

An example of a digital neural network chip is the Oxford Computer "64K-by-1b / 64p Intelligent Pattern Recognition Memory (IPRM) Chip" (Maren et al, 1990).
Table 2: Advantages and disadvantages of digital neural networks (Source: Maren et al., 1990)

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>• High noise immunity</td>
<td>• Requires more transistors to implement fundamental operations</td>
</tr>
<tr>
<td>• Speed of operation</td>
<td>• High power dissipation rates</td>
</tr>
<tr>
<td>• Precise computation</td>
<td>• Requires analog -to-digital conversions of most incoming sensor data</td>
</tr>
<tr>
<td>• Readily designed using existing tools</td>
<td></td>
</tr>
<tr>
<td>• Can include programmable components</td>
<td></td>
</tr>
<tr>
<td>• Easy to multiplex/demultiplex</td>
<td></td>
</tr>
<tr>
<td>• Can store both fixed and adaptive weights</td>
<td></td>
</tr>
</tbody>
</table>

6.4.3 Hybrid neural network architectures (Maren et al., 1990)

Hybrid neural network architectures combine digital and analog techniques. This results in networks that display the best features of the networks it is derived from.

7. DIFFERENT NEURAL NETWORK ARCHITECTURES

7.1 Network Classification

Network classification is a topic discussed by many authors (Vemuri, 1988; Lippmann, 1987; Josin, 1987; Daniell, 1991; Hecht-Nielsen, 1988; Maren et al., 1990). Many criteria can be used to classify networks of various architectures including.

1. Biological Plausibility:
Networks can be divided into two groups: those which employ biologically plausible learning rules (e.g. Hebbian learning) and those which employ learning rules that are not biologically plausible (e.g. back-propagation) (Thorpe and Imbert, 1989).

2. Information Flow:
Networks can be divided into feedforward and feedback networks. In feedforward networks, there are no feedback connections and information
flows in the forward direction only. In feedback networks, feedback connections exist and information flows forwards and backwards.

3. Learning:
Networks can be divided into those trained with and without supervision.

4. Association:
Networks can be divided into auto-associative and hetero-associative networks. In auto-associative networks, the output is the same as the input. Hetero-associative networks associate an input pattern with a different output pattern.

5. Input:
Certain networks only accept binary inputs whereas other networks operate on continuous valued inputs.

Various authors have devised a system of classification for neural networks.

Daniell (1991) suggests the factors which should be used to categorise ANNs are the way information flows through the network and the learning rules used.

Lippmann (1987) has devised a taxonomy for ANNs. As shown in Figure 10, the primary distinction is between networks using binary and continuous valued inputs. Below this, networks are divided into those trained with and without supervision.
Maren et al. (1990) have used structural similarity as the criterion for network classification. This classification has been adopted in this thesis and is summarised in Figure 11. Using this classification, networks are divided into six structurally related groups.

1. Multi layered, feedforward networks
2. Single-layer, laterally-connected networks
3. Single-layer, topographically ordered networks
4. Bi-layer, feed forward/feedback networks
5. Multilayer cooperative/competitive networks
6. Hybrid networks
7.2 Multilayer Feed Forward Neural Networks I: Delta Rule Learning

The networks discussed in this section all have layered architectures, use supervised learning rules and only have feedforward connections between processing elements (i.e. there are no lateral, self-, or back-connections) (Maren et al, 1990). The difference between the various networks is in the way they learn.

7.2.1 Perceptron networks

Perceptron networks are discussed by many authors (NeuralWare, Inc., 1991; Lippmann, 1987; Adeli and Yeh, 1990; Maren et al., 1990; Hertz et al., 1991).

Perceptron networks were developed by Rosenblatt (1958) in an effort to model the pattern recognition capabilities of the visual system. The original
perceptrons had no hidden nodes. However, multilayer perceptrons were developed at a later stage (Figure 12)

A Single-Output, Bilayer Perceptron  
A Multi-Output, Multi-Layer Perceptron

Figure 12: Typical Perceptron Architecture (Source: Maren et al., 1990)

Perceptron networks have the ability to decide whether an input belongs to one of several classes. Each of the classes is represented by a node in the output layer. The input may be binary or continuous valued. Each node computes the weighted sum of the inputs connected to it, subtracts a threshold value and passes the result through a threshold transfer function resulting in a binary output.

Learning in perceptron networks is supervised. The weights in the input and hidden layers are fixed and only the output layer weights are adjustable. Initially these weights are assigned random values within set limits. The basic philosophy for adjusting weights is as follows (NeuralWare, Inc, 1991):

1. If the output is correct, the weights remain unchanged.
2. If the desired output is 1.0 and the actual output is 0.0, the weights on the active input lines are incremented.
3. If the desired output is 0.0 and the actual output is 1.0, the weights on the active input lines are decremented.

An active input line is defined as one which has an activation level greater than the threshold level.

There are various methods that can be used to determine the absolute value of the weight change.

1. The weight increment or decrement can be fixed.
2. The weight increment or decrement can vary in proportion with the difference between the actual and desired output values.
3. The weight increment or decrement can be a mixture of the above two methods.
Rosenblatt proved that the above method for determining connection weights converges if the inputs come from two linearly separate classes. For a perceptron network without hidden units, this means that the input classes have to be able to be separated by a single straight line (Figure 13). This was the criticism that Minsky and Papert (1969) had of perceptrons, since functions such as the Exclusive-OR function do not have a linearly separable input space and hence cannot be represented by perceptron networks. However, multi-layer perceptrons are able to form more complex decision regions, as illustrated in Figure 13. In fact, a three-layer perceptron-like feed forward network can represent any function, as a three-layer network can generate arbitrarily complex decision regions (Lippmann, 1987).

The problem with multilayer networks is deciding on how many nodes to use. There must be enough nodes to ensure that the decision region has the necessary complexity. On the other hand, the number of nodes must not be too large, as too many weights may not be reliably estimated from the available training data (Lippmann, 1987).

<table>
<thead>
<tr>
<th>Structure</th>
<th>Types of Decision Regions</th>
<th>Exclusive Or Problem</th>
<th>Classes with Meshed Regions</th>
<th>Most General Region Shapes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-Layer</td>
<td>Half Plane Bounded by Hyperplane</td>
<td>A</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Two-Layer</td>
<td>Convex Open or Closed Regions</td>
<td>A</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Three-Layer</td>
<td>Arbitrary (Complexity Limited by Number of Nodes)</td>
<td>A</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>

Figure 13: Decision Regions Formed by Various Networks (Source: Lippmann, 1987)
7.2.2 ADALINE and MADALINE networks

The ADALINE network (from ADAptive LINear Element) was developed by Widrow and Hoff (1960) and was their first contribution to neural computing. ADALINE networks are very similar to perceptron networks. Like perceptron networks, ADALINE networks are capable of classifying linearly separable patterns. The architecture of both networks is very similar (Figure 14) and both networks subtract a threshold value from the summed, weighted input as well as passing it through a threshold transfer function.

![Diagram showing ADALINE and MADALINE networks](image)

**Figure 14: Typical Adaline/Madaline Architecture (Source: Maren et al., 1990)**

The difference between ADALINE and perceptron networks is in the way they learn. ADALINE learning is more sophisticated than perceptron learning. In perceptron networks, the error is calculated using the difference between the actual and the desired outputs. In ADALINE networks, the error is calculated using the difference between the neural activation of the output node before it is passed through the transfer function and the desired output. As a result, the network is more sensitive to the real difference between activation and output.

Connection weights, which are initially set to random values, are adjusted using the following learning algorithm (NeuralWare, Inc, 1991):

1. The input is applied at the input layer and the desired output is applied at the output layer.
2. The error is computed.
3. The weights are adjusted so that the error is reduced by $1/N^w$ or $1/(N^I+1)$, where $N^w$ is the number of connection weights and $N^I$ is the number of nodes in the input layer.
4. The above procedure is repeated for the next input/output set.
This type of learning procedure is called Widrow-Hoff learning in honour of its inventors Bernard Widrow and Marcian Hoff.

MADALINES or Multiple ADALINES combine several ADALINES in parallel. MADALINES have a single output unit, which equals +1 if more than half the inputs are +1 and equals -1 otherwise. MADALINES are able to solve the linear separability problem. However, MADALINES are only capable of discriminating between two classes as they use binary inputs. This limitation can be overcome by using many independent MADALINES.

The MADALINE learning procedure is given below (NeuralWare, Inc, 1991):

1. The input is applied at the input layer and the desired output is applied at the output layer.
2. If the actual output and the desired output coincide, learning is not necessary.
3. If the actual output and the desired output do not coincide, the ADALINE whose output was closest to zero in the wrong direction is adjusted using the ADALINE learning rule outlined above.
4. The above steps are repeated until the network converges.

7.3 Multilayer Feed Forward Neural Networks II: Optimising and Random Learning Methods

The networks presented in this section are structurally very similar to the networks discussed in Section 7.2. The difference between the networks is in the way they learn. The networks in the previous section used the difference between the actual and the desired outputs to drive the learning rule. In contrast, the networks presented here use random and statistical considerations as part of their learning rule (Maren et al., 1990).

7.3.1 The Boltzmann Machine

The Boltzmann machine was developed by Ackley et al. (1985) and is discussed by NeuralWare, Inc. (1991), Maren et al. (1990) and Hecht-Nielsen (1990). The Boltzmann machine is similar to back-propagation networks in structure and performance. It incorporates hidden nodes and uses a supervised learning rule which is able to successfully adjust the weights of nodes in hidden layers. The Boltzmann machine is capable of solving non-linearly separable problems, such as implementing the Exclusive-Or function. The architecture of the Boltzmann machine is shown in Figure 15.
The Boltzmann machine is derived from the Hopfield network. Like the Hopfield network, the Boltzmann machine has an associated energy function that represents the energy states of the system. However, in contrast to the Hopfield network, the Boltzmann machine is hetero-associative rather than auto-associative, has hidden nodes and incorporates a local learning rule.

![Transfer Function Diagram](image)

Figure 15: Typical Boltzmann Machine Architecture (Source: Maren et al., 1990)

The Boltzmann machine uses a learning rule which makes use of simulated annealing. Simulated annealing enables the network to escape local minima in the error surface. The method is called simulated annealing because it is analogous to physical annealing. The analogy is between the energy state of the network and the energy of a physical solid as it is slowly being cooled. At high temperatures, the network behaves like a random model and is able to jump freely from one local minimum to another. As the temperature is reduced slowly, the energy of the system is decreased and the system can still jump out of "shallow" local minima, but not the "deeper" ones. At these intermediate temperatures, the model has a stochastic component, the magnitude of which depends on the temperature. At low temperatures, the network behaves more like a deterministic model. The simulated annealing procedure does not guarantee that the global minimum solution is found, but such a result is very likely.

An analogous example of the simulated annealing procedure is that of shaking a marble on a surface in a box (Wassermann, 1989), as shown in Figure 16. The degree of shaking in the horizontal direction represents the magnitude of the temperature in Boltzmann machines. Initially, the box is shaken violently and the marble moves around randomly. As the shaking becomes less violent, the marble will stay at points A and B for short periods of time. On further reduction of the shaking amplitude, a critical point is reached at which the shaking is just strong enough to move the marble from point A to point B,
but not strong enough to enable the marble to move up the slope from B to A. As a result, the marble reaches a global minimum in the surface.

The name Boltzmann machine derives from the fact that the individual processing elements in the network are stochastic rather than deterministic. The probability that a given processing element is active is given by the Boltzmann distribution and is a function of the temperature and the difference in the energy level between the two possible states of the processing element.

![Graph of weight](image)

**Figure 16: Marble on a Surface in a Box (Source: Wassermann, 1989)**

The detailed Boltzmann machine learning algorithm involves starting at a high artificial temperature and carrying out the following steps (Wassermann, 1989):

1. Apply the input at the input layer and the desired output at the output layer.
2. Calculate the mean squared error between the desired output and the actual output.
3. Change one of the connection weights by a random amount and repeat steps 1 and 2 above.
4. (a) If the error is reduced, retain the weight change.
   (b) If the error is not reduced, retain the weight change with a probability determined by the Boltzmann distribution (Figure 17), which is a function of the artificial temperature. If the artificial temperature is high, the probability of retaining the weight change is greater, increasing the network's chances of escaping a local minimum in the error surface. If the weight is not retained, it is returned to its previous value.
5. Repeat steps 1 to 4 gradually reducing the artificial temperature.

The problem with Boltzmann machine learning is that it is slow. However, training speed can be increased by substituting the Cauchy distribution for the Boltzmann distribution (Szu and Hartley, 1987). This is because the Cauchy
distribution has longer "tails", thereby increasing the probability of large step sizes (Figure 17).

The Boltzmann machine is generally only used for optimisation or resource constrained scheduling problems.

![Boltzmann vs Cauchy Distribution](image)

*Figure 17: Cauchy versus Boltzmann Distributions (Source: Wassermann, 1989)*

### 7.3.2 Directed random search networks (NeuralWare, Inc., 1991)

The directed random search network is based on the random optimisation method of Matyas (1965) and includes refinements proposed by Solis and Wets (1981). Unlike calculus based gradient descent methods, which move down the error surface in weight space, directed random search networks take random steps in weight space in an attempt to find the smallest error. A directed component is added to the random step so that previously successful directions are pursued.

The basic weight adjustment procedure is as follows (NeuralWare, Inc., 1991):

1. Weights in the network are assigned randomly.
2. A random step value is added to each weight.
3. The prediction error is calculated for each training sample.
4a. If the total prediction error is less than the previous best, the current weight values, which include the random step, become the new set of best weights.
   - The current prediction error is stored as the new, "best" prediction error.
4b. If the total prediction error is greater than the previous best, the same random value is subtracted from each weight, producing a "reversal" step in the direction opposite to the previous random step.
   - Steps 3 and 4a are then repeated.
If the reverse step fails to reduce the error, a completely different vector is added to the best weights and the process is repeated.

The size of the random step is specified using a variance parameter, which is adjusted in response to the current success or failure rate. If several consecutive steps result in a decrease in the error, the variance automatically increases so that larger steps are taken. The reverse applies.

Apart from the variance, the only other parameter that needs to be defined is the upper bound on the weight magnitude. This defines a compact search space resulting in guaranteed convergence.

Since only two parameters need to be adjusted, this method is easy to use. Another advantage of this method is that training is very fast.

### 7.4 Laterally-Connected, Auto Associative Networks

Single-layer, fully laterally-connected networks act as auto-associators, as they can only store one pattern at a time. These types of networks can be used to create clean patterns, as they have the ability to reproduce a noise-free, complete pattern from one that is noisy or incomplete. As with all recurrent networks, stability is a consideration, as information flows in more than one direction and oscillations can occur (Maren et al., 1990).

#### 7.4.1 Hopfield / Tank networks

Hopfield networks are discussed by many authors (NeuralWare, Inc., 1991; Lippmann, 1987; Maren et al., 1990; Hertz et al., 1991).

The Hopfield network was developed by Hopfield (1982). It evolved from research into the neuro-physiology of garden slugs. The architecture of the Hopfield network is such that the output of each node is fed back to all other nodes via weighted connections as shown in Figure 18. Positive weights represent excitatory connections while negative weights represent inhibitory connections.

The inputs to the network are binary. Each node computes the weighted sum of the inputs and passes it through a threshold or a sigmoidal transfer function. The network also has the property that all the weights are symmetrical (i.e. $w_{ij} = w_{ji}$).

Hopfield networks can be used as an associative memory or to solve optimisation problems.
Figure 18: Typical Hopfield Architecture (Source: Maren et al., 1990)

1. Hopfield Nets for Auto-association

Hopfield networks can assume one of several stable states. Each of these states is defined by a certain combination of output values and represents a particular pattern. When a pattern is presented at the input, the processing elements interact to converge to the nearest stable state. The basic Hopfield learning rule to achieve this is as follows (NeuralWare, Inc., 1991):

\[ \Delta w_{ji} = (2\ y_j - 1)\ (2\ \chi_i - 1) \]  \hspace{1cm} (13)

where
- \( y_j \) = The output of the current processing element
- \( \chi_i \) = The input to the current processing element
- \( w_{ji} \) = The connection weight between the jth and the ith processing element

In simple terms, connections are strengthened when the input and the output of a processing element are the same and they are weakened if the input and the output differ. Convergence is guaranteed if the weights are symmetric and graded nonlinearities are used.

The two major limitations of the network are:
1. The memory capacity is limited by an upper bound equal to:
\[ 0.5 \times \frac{N_{\text{TOT}}}{\log(N_{\text{TOT}})} \]
where \( N_{\text{TOT}} \) = the total number of nodes in the network (valid for large \( N_{\text{TOT}} \)).

2. The network will not converge if patterns are close to each other.

2. Hopfield Nets for Optimisation

The state of the network can be viewed as an energy surface. The energy of the system is given by (NeuralWare, Inc., 1991):

\[ E^N = -\frac{1}{2} \sum_{j \neq i} \sum_i w_{ji} x_i y_j \]  \hspace{1cm} (14)

The change in energy due to one of the processing elements changing state is given by (NeuralWare, Inc., 1991):

\[ \Delta E^N_i = -\frac{1}{2} \Delta x_i \times \sum_j (w_{ji} y_j) \]  \hspace{1cm} (15)

One element only changes its state if this results in a decrease in the network energy. As a consequence, the network is guaranteed to converge.

The Hopfield / Tank network has been used successfully to solve the much publicised travelling salesman problem. The travelling salesman problem involves minimising the total distance a salesman has to travel between a number of cities, so that he visits each city once only and also returns to the starting point.

7.4.2 Brain-State-In-A-Box networks (Maren et al., 1990)

The brain-state-in-a-box model was developed by Anderson (1983). This network has conceptual similarities to the Hopfield / Tank network. Like the Hopfield network, the brain-state-in-a-box network carries out auto-associative recall of binary valued input vectors. However, there are several differences between the two networks. The brain-state-in-a-box network is more robust, is usually not fully interconnected, has better inherent memory capacity, does not usually utilise symmetrical weights and includes feedback connections (i.e. a processing element can directly affect its own value). The architecture of the brain-state-in-a-box network is shown in Figure 19.

The brain-state-in-a-box network can either employ a Hebbian or a Widrow-Hoff (delta) learning rule. The use of different learning rules results in networks with different characteristics.
The basic operation of the brain-state-in-a-box network is based on the operation of the linear associative network described by the following matrix multiplication (NeuralWare, Inc., 1991):

\[ \mathbf{OP} = \mathbf{WN} \mathbf{IN} \]  \hspace{1cm} (16)

where
\[ \mathbf{IN} = \text{a vector containing the network inputs} \]
\[ \mathbf{WN} = \text{a matrix containing the network connection weights} \]
\[ \mathbf{OP} = \text{a vector containing the predicted network outputs} \]

In addition, the brain-state-in-a-box network incorporates a nonlinear post-processing algorithm designed to clean up spurious responses produced by linear networks. Brain-state-in-a-box networks take binary inputs and produce continuous-valued outputs bounded by -1 and 1. When an input is presented to the network, the output produced is repeatedly fed back to the input resulting in a series of interim outputs. As this procedure converges, the final output is produced. Since the output is bounded by -1 and 1, the system is forced to converge to the corners of a hyper-cube (box), resulting in the name brain-state-in-a-box.

7.5 Vector-Matching Networks

Vector-matching networks are single-layer networks that can act as auto-associators as well as optimisers. Unlike other network types, there are no connections between processing elements. Instead, pattern vectors are related to similar ones in a topological sense (Maren et al., 1990).
7.5.1 Learning Vector Quantisation (LVQ) networks

Learning vector quantisation networks are discussed by several authors (NeuralWare, Inc., 1991.; Maren et al., 1990).

The LVQ network was originally suggested by Kohonen (1988). LVQ networks consist of a single layer of neurons and are auto-associators, which assign input vectors to one of several classes. There are no explicit connections between individual processing elements. Each processing element represents an exemplar for one class.

The training procedure, which is supervised, is given below (NeuralWare, Inc., 1991):

1. An individual pattern from the training set is presented to the network, and the distance between the training vector and each processing element is calculated.
2. The processing element with the minimum distance is declared the winner.
3. (a) If the class of the winning processing element corresponds to that of the training vector, it is moved closer to the training vector.
   (b) If the class of the winning processing element does not correspond to that of the training vector, it is moved away from the training vector - this process is referred to as repulsion.

The basic learning vector quantisation procedure described above suffers from various shortcomings, and modified procedures have been developed to address these problems.

1. Learning Vector Quantisation 1

One of the shortcomings the basic learning vector quantisation method suffers from is that on occasions, some processing elements win too often, while others are not utilised. To alleviate this problem, a "conscience" mechanism was suggested by DeSieno (1988). This method gives a guilty conscience to the processing element that wins too often by assigning a distance bias to it and hence reduces its chances of winning.

2. Learning Vector Quantisation 2

This method is applicable to cases where the closest processing element is in the wrong class and the second closest processing element is in the correct class. The LVQ 2 procedure is able to refine solutions by fine tuning the boundary between regions, where misclassifications are occurring.
7.5.2 Self-Organising Maps (Lippmann, 1987; Maren et al., 1990)

The self-organising map was developed by Kohonen between 1979 and 1982 (Kohonen, 1982). Self-organising maps are biologically plausible. They assign categories to the inputs presented to them and have a two-layer architecture; an input layer and a Kohonen layer. The input layer is fully connected to the two-dimensional Kohonen layer. The Kohonen layer forms a two-dimensional feature map of the input data in a way that preserves order.

During learning, continuous-valued inputs are presented without providing the desired output. The processing elements in the Kohonen layer each measure the Euclidean distance of its weights to the input values. The processing element with the smallest distance, as well as its neighbouring processing elements, adjust their weights to become closer to the input. This is instrumental in preserving the order of the input space. Once enough training vectors have been presented, the weights form clusters that sample the input space. The point density function of the clusters approximates the probability density function of the input vectors as shown in Figure 20. The organisation of the weights is such that topologically adjacent nodes are sensitive to physically similar inputs.

![Weight Vectors During the Ordering Process](Source: NeuralWare, Inc., 1991)

During recall, the processing element in the Kohonen layer that is closest to the input is the winner and has an output of 1.0. All other processing elements in the Kohonen layer have an output of 0.0. This process is essentially an order-preserving projection of the input space onto the Kohonen layer.
If one particular processing element wins too often, it represents too much of the input space. To overcome this, a "conscience" mechanism is employed, similar to that used with LVQ networks. The conscience mechanism increases the distance of a processing element, if it has won more than $1/N^K$ times, where $N^K$ is the number of processing elements in the Kohonen layer. This decreases its chance of winning.

7.6 Feedforward/Feedback (Resonating) Hetero-Associative Networks

The networks described in this section have both feed forward and feedback connections, and may have lateral connections. As a result, stability is a consideration, as oscillations can occur that may not converge to a stable state. These networks also employ unsupervised learning rules and there is no distinction between the training phase and the operation phase (Maren et al., 1990).

7.6.1 Carpenter / Grossberg Adaptive Resonance Theory (ART) networks

ART networks are discussed by a number of authors, including NeuralWare, Inc. (1991), Lippmann (1987) and Maren et al. (1990).

Grossberg developed this type of network in an attempt to create a biologically plausible pattern classifier. ART networks, like the human brain, are able to overcome the stability-plasticity dilemma which many other networks are faced with (Wasserman, 1989). In other words, ART networks have the ability to maintain the plasticity required to learn new patterns, while preventing the corruption of patterns that have been learned previously. This is useful when a fully trained network must learn from new training data. Learning in ART networks is completely unsupervised in that the network itself determines criteria for creating different categories.

The ART network consists of two interconnected layers called F1 and F2 as shown in Figure 21. When an input is supplied to the network, the F1 layer becomes activated. This acts as a short-term memory, as this activation ceases when the input is removed. The F1 layer passes the activation to the F2 layer via weighted connections. The processing elements in the F2 layer calculate the weighted sum of their inputs and compete with each other until only the winning processing element is active. The winning processing element represents the class of the input.
The problem of network stability is addressed by having bottom-up pathways between the processing elements in F1 and F2, as well as having top-down pathways between the processing elements in F2 and F1. The top-down signal has the function of reinforcing critical features of a particular category. As a result, the processing elements in the F1 layer receive stimuli from the input presented to the network as well as from the F2 layer. During recall, information resonates between the two layers, so that critical features in F1 are reinforced and consequently have the greatest activity.

The ART network also incorporates an attentional gain control unit, which enables F1 to distinguish between external input stimuli and stimuli sent down from F2. If an external input is present, the gain is high. If the only stimulus present is that from F2, the gain is low and only subliminal activity can arise in F1.

An orienting subsystem is responsible for determining how coarse or fine categories will be. The orienting subsystem sends a reset wave to F2 whenever the activity pattern in F1 produced by the input differs significantly from that caused by the top-down signal for a particular category. A vigilance parameter ($\beta'$) determines how large the mismatch in F1 can be before the reset wave is sent. A high vigilance means that only a small mismatch is tolerated and vice versa.

Long term memory is represented by the weights of the bottom-up as well as the top-down connections. Learning in the top-down connections is as follows (NeuralWare, Inc., 1991):
• The connection weights will approach 1.0 if the connection links two active neurons.
• The strength of a connection will remain unchanged if it links two neurons that are inactive.
• The strength of a connection will decay towards 0 if it links an active F2 processing element and an inactive F1 processing element.

Learning in the bottom-up connections is similar to that in the top-down connections. The exception is the case where both neurons are active. Rather than approaching the maximum level of 1.0, the connection strength is dependent on the number of active neurons, as these compete for the resources. This learning rule is known as Weber's law.

Problems associated with ART networks include:

1. The network assumes that patterns that share a greater number of input features are more closely related and more likely to belong to the same category. This could produce erroneous results.
2. Performance is high when perfect input patterns are presented but deteriorates markedly when only a small amount of noise is present.

Possible modifications to improve performance in the presence of noise include (Lippmann, 1987):
1. Adapting weights more slowly.
2. Changing the vigilance during training and testing.

7.6.2 Bidirectional Associative Memory (BAM) networks

The BAM model is another resonating, hetero-associative network which was developed by Kosko (1987), and was inspired by the ART network. BAM networks store pattern associations so that when an input is presented to the network, the network converges to one of the known associated patterns. They also have the ability to create new categories and to learn new patterns.

Operation of this type of network is similar to that of simple linear hetero-associators, but differs in that information resonates between layers until both input and output patterns stabilise. The network incorporates two central BAM layers as shown in Figure 22.

The two BAM layers are fully interconnected with bottom-up and top-down connections. Long-term associative information is stored in the bottom-up and top-down connection weights. Connection weights are adjusted using a Hebbian learning rule. The learning rule may be viewed as creating an energy surface in which local minima represent particular associations. When noisy
input is presented to the network, the corresponding category is determined by moving down the energy surface to one of the local minima. The advantage of BAM networks have over ART networks is that they work well, even when the input is noisy.

![BAM Architecture](image)

**Figure 22: Typical BAM Architecture** *(Source: NeuralWare, Inc., 1991)*

### 7.6.3 Recirculation networks *(NeuralWare, Inc., 1991)*

Recirculation networks were developed by Hinton (1988). They are biologically plausible, have feedback connections and use an unsupervised learning rule. However, unlike other networks in this section, recirculation networks are auto-associators.

Recirculation networks typically consist of a visible layer and a hidden layer, which are fully interconnected in both directions (Figure 23).

![Recirculation Architecture](image)

**Figure 23: Typical Recirculation Architecture** *(Source: NeuralWare, Inc., 1991)*
The connections from the visible layer to the hidden layer are called bottom-up connections, whereas the connections from the hidden layer to the visible layer are called top-down connections. The purpose of the hidden layer is to form an internal representation of the data presented at the visible layer. This hidden representation is usually a compressed version of the input, as the number of processing elements in the hidden layer is less than those in the visible layer.

The learning process can be described as follows (NeuralWare, Inc., 1991): During learning, the input stimulus is passed from the visible layer to the hidden layer (time 1), recirculated back to the visible layer (time 2) and then passed back to the hidden layer (time 3). The top-down and the bottom-up weights are incremented or decremented after the second pass through the network. During training, the output of the hidden layer at time 1 is the encoded version of the input. The output of the visible layer at time 2 is the reconstruction of the original input from the encoded version. The weights are adjusted by implementing gradient descent down the surface of the squared error between the original and the reconstructed input.

7.7 Multilayer Cooperative / Competitive Networks

Multilayer cooperative / competitive networks are designed to mimic certain biological networks. The processing elements in the networks cooperate as well as compete with each other in order to perform the desired task. Usually, the cooperative and competitive processes are carried out by separate layers. This architecture results in subtle and sophisticated feature recognition and pattern recognition networks. The competitive mechanism is achieved by adding lateral, inhibitory connections to feed forward networks (Maren et al., 1990).

These networks are not discussed in detail as part of this review. Examples of multilayer cooperative / competitive networks include:
1. Competitive learning networks (Rumelhart and Zipser, 1985)
2. Masking field networks (Grossberg, 1978)
3. Boundary contour systems (Grossberg and Mingolla, 1985a, 1985b)
4. Hierarchical scene structures (Minsky and Maren, 1989)
5. The neocognitron (Fukushima, 1988)

7.8 Hybrid Networks

Human learning is a mixture of unsupervised learning ("learning by discovery") and supervised learning ("learning by instruction") (Hrycej, 1990). Typically, hybrid networks combine a competitive, or unsupervised,
layer with a supervised layer resulting in rapid computation (Burke and Ignizio, 1992). This architecture leads to two distinct phases in the learning process. The unsupervised learning component acts as a preprocessing unit which discovers features that are linear functions of the input vector. The results of this stage are passed to the supervised learning phase (Hrycej, 1990).

Hybrid networks are usually used when known architectures are inadequate to perform the task at hand and are usually formed by combining networks of known architectures. Such networks can be strongly coupled or loosely coupled (Maren et al., 1990). Strongly coupled hybrid networks are treated as a single network. They consist of two or more networks that have been fused together to form a single structure in a way that highlights the strengths and minimises the weaknesses of the component networks. Weakly coupled hybrid networks are formed by linking two or more networks that retain their individual characteristics.

7.8.1 Hamming networks (NeuralWare, Inc, 1991; Lippmann, 1987)

Hamming networks assign inputs to one of several classes represented by exemplar vectors. The Hamming distance is calculated between the input vector and each exemplar vector. The class of the input vector is deemed to be that with the smallest Hamming distance. The Hamming distance is defined as "...the number of bits in the input vector which do not match the corresponding bits in the exemplar vector." (NeuralWare, Inc., 1991)

The architecture of the Hamming net is shown in Figure 24. Hamming networks consist of two subnets (Lippmann, 1987). The lower subnet has adjustable weights and calculates the number of matching bits. The "MAXNET" has fixed weights and picks the exemplar that has the maximum number of matching bits. It has been shown that the "MAXNET" always converges.
7.8.2 Counter-propagation networks (Wassermann, 1989)

Counter-propagation networks were developed by Hecht-Nielsen (1987a, 1988). They function as a look-up table which finds the exemplar closest to the input vector by competition between exemplars, and reads out its equivalent mapping. Input vectors may be binary or continuous-valued.

Counter-propagation networks do not possess the same generalisation capabilities as back-propagation networks but can complete the training phase up to one hundred times faster. They can also produce correct outputs when presented with incomplete or partially incorrect inputs.

The counter-propagation architecture consists of self-organising maps (Kohonen, 1982) and Grossberg outstars (Grossberg, 1969, 1971). The Kohonen layer acts as a nearest neighbour classifier. The processing elements in the Kohonen layer compete with each other and the one with the highest output wins. The Grossberg outstar is used in the output layer and typically learns to produce a certain output when a particular input is applied.

Counter-propagation networks act as bi-lateral auto-associative networks. As shown in Figure 25, two input vectors, \(i_1\) and \(i_2\), are applied at the input layer. The two input vectors interact with each other to select the processing
element that is closest to the composite vector \((i_1, i_2)\). The winning processing element in the Kohonen layer activates the Grossberg layer, which provides the nearest \(o_1^p\) and \(o_2^p\).

![Diagram of Counter-Propagation Architecture](image)

Figure 25: Typical Counter-Propagation Architecture (Source: Wasserman, 1989)

7.8.3 Probabilistic neural networks
(NeuralWare, Inc., 1991; Specht and Shapiro, 1990)

Probabilistic neural networks act as pattern classifiers. They employ supervised training rules and use the training data to develop distributed functions, which are used to estimate the likelihood of an input vector belonging to a particular category. In fact, the probabilistic neural network is a neural implementation of the Bayes classifier. The Bayes classifier uses Parzen estimation (Parzen, 1962) to build a probability density function over the feature space for each category. This enables the computation of the likelihood of a given input vector belonging to a certain category. The probabilistic neural network selects the most likely category for a given input vector.

Training speeds of probabilistic neural networks are much faster than those of back-propagation networks. However, back-propagation networks are faster than probabilistic neural networks during recall. Probabilistic neural
networks also require a lot of training examples in order to perform generalisations satisfactorily.

8. BACK-PROPAGATION NETWORKS

The back-propagation network was developed by Rumelhart, Hinton and Williams (1986), and is the most widely used network today. Back-propagation networks (Figure 26) belong to the class of multilayer feedforward networks that use the delta learning rule.

![Figure 26: Typical Back-Propagation Architecture (Source: Maren et al., 1990)]

Back-propagation networks are capable of adjusting the weights in the hidden layer(s) and hence solve the credit assignment problem formulated by Minsky and Papert in their book "Perceptrons" (Minsky and Papert, 1969). This is achieved by assigning part of the blame for erroneous outputs to all processing elements (NeuralWare, Inc., 1991).

Back-propagation networks are mapping networks that approximate mathematical functions or mappings from an input space to an output space using examples of the mapping's action. Mapping networks are variants of the methods of statistical regression analysis (Hecht-Nielsen, 1990). They can be designed and trained to map many complex patterns and have the ability to generalise with the aid of the hidden layer nodes, which perform nonlinear feature extractions (Maren et al., 1990).

Broomhead and Lowe (1988) suggest that back-propagation networks may be viewed as performing simple curve fitting operations in high dimensional space. In addition, learning is seen to be synonymous with producing a best fit surface in a high dimensional space to a finite set of data points (the
training set), and generalising is seen to be equivalent to interpolating the test set on this fitting surface.

Back-propagation networks are fully interconnected. During recall, information flow is feed forward only. The back-propagation architecture is the same as that for multilayer perceptrons with the following exceptions (Maren et al., 1990):

1. Back-propagation networks use continuous transfer functions, such as the sigmoid function. These functions are asymptotic for infinitely large and small values of activation. In fact, for most input values, the output of the transfer function takes on one of the two asymptotic values, allowing the output to be roughly grouped into one of two classes.

During training, weights are adjusted in proportion to the derivative of the sigmoid function. As shown in Figure 27, the derivative of the sigmoid function is always positive, is close to zero for extreme input values and is at its maximum when the input is zero. As a result, small weight changes occur when the output is either zero or one, and large weight changes occur when the output is in the middle range. This is desirable, as outputs of zero or one represent stable states and no further changes are required. If the output is in the medium range, however, large changes are required to drive the output to one of the two stable states.

2. In contrast to perceptron networks, back-propagation networks use the generalised delta learning rule discussed in Section 8.1.2 below.

![Diagram](image)

Figure 27: The Sigmoidal Transfer Function and its Derivative (Source: Maren et al., 1990)
8.1 Learning

The most interesting aspect of the back-propagation network, and the reason for its success and popularity, is the back-propagation learning algorithm, which is based on delta rule learning for networks with no hidden nodes.

8.1.1 The delta rule

Delta rule learning is a supervised learning algorithm for networks without hidden nodes. The input is presented at the input layer and passed to the output layer. The actual output is then compared with the target output, and the difference, or delta, between the actual and the desired response of the network is calculated. The weights are then changed in accordance with the following equation (Jones and Hoskins, 1987):

\[ \Delta w_{ji} = \eta (o_k^d - o_k^p) I_i \]  \hspace{1cm} (17)

where  
\( o_k^d \) = the desired (actual, historical) output at node k  
\( o_k^p \) = the predicted output at node k  
\( \Delta w_{ji} \) = the change in weight between nodes i and j  
\( I_i \) = the activation of the input node  
\( \eta \) = the learning rate

This basically means that the credit (or blame) for the error assigned to the input units is proportional to their activation. The more active nodes are, the more they are responsible for the actual output, and hence the error produced. As a result, the weights of the most active nodes undergo the largest modification. In fact, the direction of weight change is such that it will minimise the sum of the squares of the output deltas. As learning progresses, the network cycles through the training set until a specified performance level is reached.

The delta rule works well for single layer systems but cannot be used for multilayer systems, as there is no mechanism for working out weight changes for nodes in the hidden layer(s). This problem was solved by the introduction of the back-propagation or generalised delta rule.

8.1.2 The generalised delta rule

The generalised delta rule is a supervised learning rule for adjusting weights in multilayer networks. It is widely used and is discussed by many authors (Maren et al., 1990; Hecht-Nielsen, 1990; Lippmann, 1987; Burke and
Ignizio, 1992; White, 1990; Jones and Hoskins, 1987; Minai and Williams, 1990; NeuralWare, Inc., 1991; Cheung et al., 1990).

The goal of the learning procedure is to find a set of weights that enable the network to perform the desired input/output mapping. During training, the mean square difference between the actual and the desired output is calculated as a function of the weights. A different error is obtained for each combination of weights, creating a mean squared error surface sitting above the weight space (Figure 28). This error surface has a large number of local minima because of the large number of combinatorial permutations of the weights that leave the network input/output function unchanged. The aim of the training procedure is to find a set of weights that will minimise the error function.

The weight changes are obtained by performing gradient descent in weight space. Each weight is changed in proportion to the value of the partial derivative of the error function with respect to that weight. Consequently, weight changes are made down the gradient of the error surface.

![Figure 28: A Typical Error Surface (Source: Hecht-Nielsen, 1990)](image)

The weights are adjusted in two stages. During the first stage, the weights between the hidden layer and the output layer are adjusted. In the second stage, the weights between the hidden layer and the input layer are adjusted. An iterative method is used, which propagates the error terms needed for weight adjustment back from the output layer to the input layer via the hidden layer(s). As training patterns are presented to the network and weights are adjusted gradually in order to approximate the correct input/output mapping, the network goes through the following stages of learning (Cheung et al, 1990):
1. Error-Convergent Stage:
In this stage, the errors produced by the various patterns tend to converge towards their mean.

2. Competition Stage:
The competition stage immediately follows on from the error-convergent stage. Initially, the gradients of the individual patterns are in opposing directions and tend to cancel each other out resulting in an overall gradient equal to zero. Consequently, the individual pattern gradients compete with each other in order to drive the overall gradient away from zero. The changes in the overall gradient will be such that they result in a decrease in the error function. This will cause the errors to eventually move away from their mean.

3. Domination Stage:
In this stage, some patterns are dominantly trained. The errors of these patterns decrease relatively quickly in comparison with other patterns. As learning progresses, this domination situation disappears.

In general, stage one is very short whereas the duration of stages two and three is highly problem dependent.

The detailed back-propagation algorithm is given below (NeuralWare, Inc., 1991):

1. • The input vector, $i$, is presented at the input layer and the desired output, $o^d$, is applied at the output layer.

   • The input vector is then propagated through the network producing the predicted output, $o^p$.

   • As the input propagates through the layers, the output state is obtained for each node as a function of the input activation, $I$.

$$y_j^{[s]} = F \left( \sum_i \left( w_{ji}^{[s]} x_i^{[s-1]} \right) \right) = F \left( I_j^{[s]} \right)$$ \hspace{1cm} (18)

where $x_j^{[s-1]} =$ the current input from the $i$th processing element in layer [s-1]  
$y_j^{[s]} =$ the current output state of the $j$th processing element in layer [s] 
$w_{ji}^{[s]} =$ the weight on the connection joining the $i$th processing element in layer [s-1] to the $j$th processing element in layer [s]
\[ I_{j}^{[s]} = \text{the weighted summation of inputs to the } j\text{th processing element in layer } [s] \]

- Traditionally, \( F \) is the sigmoid function, but it can be any squashing function.

2. The scaled error and the delta weight are calculated for each processing element in the output layer in accordance with the following equations.

- The global error function is given by:

\[
E^G = \frac{1}{2} \sum_{k=1}^{N^O} ((o_k^d - o_k^P)^2) \tag{19}
\]

where \( E^G \) = the global error
\( o_k^d \) = the desired output for the \( k \)th processing element
\( o_k^P \) = the predicted output for the \( k \)th processing element

- The scaled local error at each processing element in the output layer, \( E^L \), is given by:

\[
E^L^{[o]} = (o_k^d - o_k^P) \cdot F'(I_k) \tag{20}
\]

where \( F'(I_k) \) = the derivative of the transfer function with respect to its input

- The global error, \( E^G \), is minimised by modifying the weights using the following gradient descent rule:

\[
\Delta w_{ji}^{[s]} = -\eta \cdot \frac{\partial}{\partial w_{ji}^{[s]}} E^G \tag{21}
\]

where \( \eta \) = the learning rate

In other words, each weight is changed in proportion to the size and direction of the negative gradient of the error surface.
- By rearranging and substituting, the following expression for the weight change is obtained:

\[ \Delta w_{ji}^{[s]} = \eta \ E_{Lj}^{[s]} x_i^{[s-1]} \]  

(22)

where \( E_{Lj}^{[s]} \) = the scaled local error in the jth processing element in layer [s]

3. • The scaled error and the delta weight are calculated for each processing element in the layers between the layer below the output layer and the layer above the input layer.

• The local error that is passed back through the layers is given by:

\[ E_{Lj}^{[s]} = -\frac{\partial E^G}{\partial I_j^{[s]}} \]  

(23)

• The relationship between the local error at a particular processing element at level [s] and all the local errors at level [s+1] is obtained by applying the chain rule twice to Equation (23).

\[ E_{Lj}^{[s]} = F'(I_j^{[s]}) \sum_k (E_{Lk}^{[s+1]} w_{kj}^{[s+1]}) \]  

(24)

• The weight update equation is the same as that for the output layer and is given by Equation (22).

4. • All the weights in the network are updated by adding the delta weights to the corresponding previous weights.

Once training is complete, the weights are frozen. Training is the only time data is back-propagated through the network. During recall, the network is strictly feed forward.

8.1.3 Use of training data (Hecht-Nielsen, 1990; Specht and Shapiro, 1990)

The training data consist of examples of the desired input/output mapping. Generally the training data are divided into two subsets; one portion for training and one portion for testing the performance of the network. It is vital that the test data are not used during training. When a limited data set is available, the holdout method can be used (Specht and Shapiro, 1990). When using the holdout method, one pattern is held out and training is carried out using the remaining patterns. The held out pattern is then tested against the
trained network. The above procedure is repeated, holding out each pattern in turn.

Generally, the more training examples are presented to the network, the better it performs the desired input/output function. However, back-propagation networks can "overtrain". When too many training examples are presented to the network, it tends to learn the specific data set better than the general problem. This is undesirable, as the network should have the ability to generalise. When a network overtrains, training should be stopped. However, it is difficult to know when to stop training. The only way to determine whether a network is overtraining is to stop training at periodic intervals, freeze the weights temporarily and test the performance of the network using the test set. As shown in Figure 29, the mean squared error will decrease in the initial stages of training. As the network starts to overtrain, the error will increase again.

![Graph showing the approximate performance level of a network](image)

**Figure 29: The Phenomenon of Overtraining** (Source: Hecht-Nielsen, 1990)

### 8.1.4 Stopping criteria

Stopping criteria determine when the training process should be stopped. Examples of stopping criteria include the following (Patterson, 1993; Lachtermacher, 1993; NeuralWare, Inc., 1991):

1. **Learning** can be stopped when the magnitude of the derivative of the transfer function reaches a sufficiently small value for all connections.
2. **Learning** can be stopped after a fixed number of training samples have been presented to the network.
3. **Learning** can be stopped when there is no further improvement in network performance during the testing phase. This involves stopping training at regular intervals, presenting the test set to the network and comparing
network predictions with the actual values. This method is known as cross-validation.

4. Learning can be stopped when the value of the error function reaches a sufficiently small value during training. However, this can lead to overtraining.

8.2 Measuring Network Performance

When assessing the performance of a network, two factors should be taken into account: learning speed and generalisation ability. It has been shown that learning speed and generalisation ability are closely related (Le Cun, 1989). Ultimately, it is the network's ability to generalise that is most important, as it determines the amount of data needed to train the network. The network's generalisation ability is a measure of how well the network has learned to approximate the desired mapping for arbitrary inputs.

A means for assessing generalisation ability is to calculate the root mean squared error (RMSE) between the predicted and the desired output. Using the RMSE as a performance measure has the advantage that large errors receive more attention than small ones. The RMSE is also more sensitive to errors on frequent inputs (Hecht-Nielsen, 1990). Alternative methods for assessing a network's generalisation ability include the average absolute error (AAE), the average absolute percentage error (AAPE), the maximum absolute error (MAE) and the median squared error (MSE) (Hecht-Nielsen, 1990).

The RMSE, AAPE and the AAE are calculated as follows. If there are \( P_{TE} \) patterns in the testing set:

\[
RMSE = \left\{ \frac{1}{P_{TE}} \sum_{k=1}^{P_{TE}} \left( o_k^d - o_k^p \right)^2 \right\}^{1/2}
\] (25)

\[
AAPE \, (\%) = \frac{100}{P_{TE}} \sum_{k=1}^{P_{TE}} \frac{\left| o_k^d - o_k^p \right|}{o_k^d}
\] (26)

\[
AAE = \frac{1}{P_{TE}} \sum_{k=1}^{P_{TE}} \left| o_k^d - o_k^p \right|
\] (27)

8.3 Network Design

As discussed in Section 8.2, the ability to generalise is the most important network performance measure. The likelihood of correct generalisation
depends on the size of the hypothesis space (the total number of networks being considered), the size of the solution space (the number of networks that have good generalisation ability) and the number of training examples (Le Cun, 1989). If the hypothesis space is too large or the number of training examples is too small, then there exist many networks consistent with the training data, but only very few of these lie in the true solution space, resulting in poor generalisation ability. Conversely, in order to obtain good generalisation ability with a network of increased generality, more training examples are required. However, it should be noted that Wann et al. (1990) suggest that increasing the number of training samples does not guarantee better generalisation ability. Hertz et al. (1991) indicate that good generalisation ability can be achieved by building as much a priori knowledge as possible into the network and by minimising the number of free network parameters.

In an attempt to improve network performance, it is important to optimise the network geometry, the initial weight distribution and the learning rate.

8.3.1 Network geometry

Several authors give guidelines as to how many hidden layers should be used (Maren et al., 1990; Hertz et al., 1991).

As shown by Cybenko (1988), at most two hidden layers are needed to approximate a particular function to a given accuracy, provided that there are sufficient processing elements per layer. The reason why two hidden layers are enough to approximate any function is based on the argument that any "reasonable" function can be represented by a linear combination of localised bumps and that such bumps can be constructed with two hidden layers. However, it may be possible that a network using more than two hidden layers may find a solution with less units in total or may speed up learning. Another point to consider is that although some functions may be represented using an architecture with two hidden layers, they may not be capable of learning with that architecture.

As a general guide, one hidden layer is probably sufficient for classification (decision boundary) problems, where the output node with the greatest activation will determine the category of the input pattern. However, if the outputs need to be continuous functions of the inputs, two hidden layers or different transfer functions, such as radial basis functions (Section 8.5.12), should be used (Maren et al., 1990).

When choosing the number of hidden layer nodes, the following should be kept in mind (Maren et al., 1990):
1. The hidden nodes represent features in the input space and not the exact input pattern
2. Keeping the number of hidden layer nodes to a minimum reduces the number of weights that need to be adjusted, and hence reduces the computational time needed for training.

Hidden nodes detect relationships between input and output values. If there is an insufficient number of hidden nodes, it may be difficult to obtain convergence during training, as the network is unable to create adequately complex decision boundaries. If too many hidden nodes are used, the network may lose its ability to generalise. This is similar to the area of statistics and curve fitting, where too many free parameters result in overfitting as shown in Figure 30.

![Figure 30: The Phenomenon of Overfitting](Source: Hertz et al., 1991)

A curve fitted with too many parameters follows all the small details or noise but is unable to interpolate or extrapolate effectively. The same applies to artificial neural networks; too many nodes result in poor generalisation.

Guidance for choosing the correct number of hidden layer nodes is given by various authors, some of which are a function of the number of network inputs and outputs, whereas others depend on the number of training samples available.

- Hush and Horne (1993) suggest that the number of independent training samples should be approximately ten times the number of weights for the network to have good generalisation ability
• Baum and Haussler (1989) derived a theoretical relationship between the number of training samples and the network size required to achieve valid generalisation. They suggest that for a given network geometry, the number of training samples required, $P^{TR}$, is given by:

$$P^{TR} \geq \frac{N^w}{\alpha}$$

(28)

where $N^w =$ the number of weights
and $0 < \alpha \leq 1/8$

If the above relationship is satisfied, the fraction of test samples that will be correctly classified is $(1 - \alpha)$

• Rogers and Dowla (1994) suggest using the rule of thumb that the number of training samples ($P^{TR}$) should be greater or equal to the number of weights in the network ($N^w$).

• Weigend et al. (1990, 1991) suggest using the following heuristic rule for determining the initial number of hidden nodes:

$$\frac{1.1 \ P^{TR}}{10} \leq N^{HI} (N^I + 1) < \frac{3 \ P^{TR}}{10}$$

(29)

where $P^{TR} =$ the number of training samples
$N^{HI} =$ the number of hidden nodes
$N^I =$ the number of inputs

However, Chakraborty et al. (1992) found this rule to be too restrictive when limited training data are available.

• NeuralWare, Inc. (1991) give the following upper bound for the number of hidden layer nodes:

$$N^{HI} \leq \frac{P^{TR}}{\alpha (N^I + N^O)}$$

(30)

where $P^{TR} =$ the number of training samples
$N^O =$ the number of outputs
$N^I =$ the number of inputs
$N^{HI} =$ the number of hidden nodes
$\alpha =$ a constant
The constant, $\alpha$, ranges between 2 and 10; 2 is used for very clean data while 10 is used for noisy data.

- Hecht-Nielsen (1987b) gives the upper limit for the number of hidden nodes as $2N^1+1$ for a network with one hidden layer.

- Maren et al. (1990) report that for many applications, the optimum number of hidden layer nodes has been found to be less than the number of inputs.

- DeSilets et al. (1992) specified the number of hidden layer nodes as a percentage of the total number of nodes in the input and output layers. They found that the optimum percentage ranged from 20% to 50%.

Kudrycki (1988) empirically determined that for networks with two hidden layers, the optimum ratio of first to second hidden layer nodes is 3:1.

When limited data sets are available, the following methods may be used to increase generalisation ability (LeCun, 1989):
1. Several connections can be controlled by a single parameter by using a technique called "weight sharing".
2. "Useless" connections can be dynamically deleted during training.

Methods for determining the optimum network architecture include the following:

- Harp et al. (1989) applied genetic algorithms to populations of descriptions of back-propagation networks. Each description (genome) contains values of various network parameters (genes). The population converges to a near optimal architecture by using a technique that ensures that good building blocks found in one trial are refined and combined with good building blocks from other trials.

- Fahlman and Lebiere (1990) developed an algorithm which adds nodes individually as needed during training. Initially, a network without hidden layers is trained to optimum performance. Subsequently, hidden nodes are added and trained one at a time until the model performance reaches a target value.

- Weigend et al. (1990) added a term to the error function that penalises large networks, resulting in the removal of excess weights during training.

- Sietsma and Dow (1991) and Chung and Lee (1992) developed pruning methods for removing superfluous processing elements based on the examination of the node outputs.
8.3.2 Initial weight distribution

The size and distribution of the connection weights at the beginning of training is very important. If the weights are identical, the network may not learn (Rumelhart et al., 1986). If the magnitude of the initial connection weights is too large, the activation function will be large and the transfer function might saturate, leaving the network in a local minimum or in an undesirable region of the weight space. Consequently, the initial weights should be small and randomly distributed.

8.3.3 Learning rate
(DeSilets et al., 1992; Minai and Williams, 1990; White, 1990)

The learning rate determines the absolute size of the weight change during learning. It determines the magnitude of the steps taken down the error surface. The learning rate is one of the factors which determines how fast the network learns.

Error surfaces have a multitude of areas with shallow slopes. As a result, large learning rate values are needed for rapid learning. However, rapid training may not necessarily be effective. If the learning rate is too large, the network might jump to undesirable areas in weight space.

Generally, the learning rate has to be found by trial and error. Rumelhart et al. (1986) have reported successfully using values ranging from 0.05 to 0.75 for a variety of problems.

8.4 Deficiencies of the Back-Propagation Algorithm

The deficiencies of the back-propagation algorithm have been widely discussed (Cheung et al., 1990; Hrycej, 1990; Specht and Shapiro, 1990; Burke, 1991; Vemuri, 1988; Burke and Ignizio, 1992; Jones et al., 1990).

Deficiencies of back-propagation networks include that they

1. Take a long time to train
2. May suffer from "overtraining", leading to poor generalisation
3. Can be temporally unstable
4. Rely on non-local computation
5. Require retraining if new information arises
6. Tend to get stuck in local minima of the error surface and hence global minima cannot be guaranteed
7. Assume all patterns are equally effective during training
8. May suffer from "network paralysis", which occurs when some weights actually freeze and the network stops adjusting these
9. Require a great deal of training data
10. Are sensitive to the value of control parameters such as the learning rate.

Some of the deficiencies related to network training stem from the fact that the assignment of optimal weights is a high dimensional, nonlinear optimisation problem for which there is no adequate solution algorithm.

8.5 Ways of Improving the Back-Propagation Algorithm

There have been a number of attempts to address the deficiencies of the back-propagation algorithm discussed in Section 8.4, some of which are discussed below.

8.5.1 Momentum  (NeuralWare, Inc., 1991; Minai and Williams, 1990)

As discussed in Section 8.3.3, it is difficult to determine an appropriate learning rate. At points of high curvature, a small learning rate is required to avoid divergent behaviour. At points of shallow slopes, a large learning rate is required in order to avoid slow learning. To resolve this contradiction, the momentum term was introduced.

The inclusion of the momentum term modifies Equation 22 as follows (NeuralWare, Inc., 1991):

\[ \Delta w_{ji}^{[s]}(t) = \eta \sum \Delta L_{j}^{[s]} x_{i}^{[s-1]} + \mu \Delta w_{ji}^{[s]}(t-1) \]  \hspace{1cm} (31)

where \( \mu = \) momentum

The fact that a function of the previous delta weight is passed through to the current delta weight ensures that general trends are reinforced and oscillatory behaviour cancels out. A positive momentum term provides a built-in inertia, allowing a small learning rate but faster learning.

8.5.2 Modifying the derivative  (NeuralWare, Inc., 1991)

Modifying the derivative of the sigmoid transfer function can nearly double the learning speed. When the node activation value becomes saturated (0.0 or 1.0), the derivative, and hence the scaled error, becomes 0.0 and the network ceases to learn. This problem is eliminated by adding a small positive offset to the derivative.
8.5.3 Different error functions (NeuralWare, Inc., 1991; Bishop, 1990)

As discussed in Section 8.3.1, if the number of hidden nodes is insufficient, the desired accuracy may not be achieved. However, if there are too many nodes, the network overfits the data as shown in Figure 30.

The following error function has been proposed by Bishop (1990) to smooth out the output function and to eliminate the problem of overfitting (Figure 30).

\[
E^G = \frac{1}{2} \sum_k ((o_k^d - o_k^p)^2) + c E^c
\]  

(32)

where \( c \) = a constant

\[
E^c = \frac{1}{2} \int_a^b \kappa^2 \text{ dx } = \frac{1}{2} \int_a^b \frac{(z')^2}{(1 + (z')^2)^3} \text{ dx}
\]

\( \kappa \) = the curvature of the line \( z = z(x) \) - see Figure 30.

By using Equation 32, many hidden nodes can be used without overfitting the data.

Many different error functions can be used. The only requirement is that their derivatives can be calculated at the output layer. Examples of possible error functions include the cubic and the quartic error functions shown below.

\[
E^G = \frac{1}{3} \sum_k \left| (o_k^d - o_k^p)^3 \right|
\]  

(33)

\[
E^G = \frac{1}{4} \sum_k (o_k^d - o_k^p)^4
\]  

(34)

8.5.4 Cumulative update of weights (NeuralWare, Inc., 1991)

The learning speed can also be increased by updating the weights after a certain number of training input/output pairs have been presented, rather than after every presentation. The number of input/output pairs presented before the update is made is called the epoch (\( \varepsilon \)). An epoch can consist of the entire training set or a subset of the training set. Faster training times can be obtained by using the cumulative update rule because global updates will always reduce the overall error function, whereas individual updates only reduce the error for the particular pattern under consideration, which may not necessarily reduce the global error. However, it is important that the epoch
size is not too large, as many more calculations need to be carried out for a single update, and hence the benefit of using an overall error function may be lost.

8.5.5 Fast back-propagation (NeuralWare, Inc., 1991)

Fast back-propagation is a variation of the back-propagation algorithm developed by Samad (1988). In fast back-propagation, the delta weight equation is given by:

\[ \Delta w_{ji}^{[s]} = \eta \ E_j^{[s]} \ (x_i^{[s-1]} + E_i^{[s-1]}) \]  \hspace{1cm} (35)

Using this variation of Equation 22 can dramatically reduce the number of iterations required for convergence.

8.5.6 Dynamic learning rate (Schmidhuber, 1989; White, 1990)

Another way to speed up the learning process is to use a dynamically varying learning rate. Gradient descent methods attempt to find a point of zero gradient in the error surface corresponding to a local minimum in the error. However, it would be more appropriate to find the zero points of the error function. This can be achieved by using a dynamically varying learning rate that is computed during each presentation of an input/output training pair.

The learning rate is selected so that the updated weight vector points to the intersection of the weight hyperplane and the line defined by the current error and the current gradient. It is assumed that the error function can be locally approximated by its tangential hyperplane. Thus the gradient is used for linear extrapolation in order to update the weight vector, producing a zero value for the error function or a value close to zero. The above process is repeated with the updated vector as the starting point for the next iteration. One step of the process is illustrated in Figure 31 for a two-dimensional example.

The value of the learning rate for each training pair is given by (Schmidhuber, 1989):

\[ \eta = \frac{E^G}{\sum_k \left( \frac{\partial E^G}{\partial w_k} \right)^2} \]  \hspace{1cm} (36)
When the denominator of Equation 36 is equal to zero, the learning rate is set equal to zero.

The above method is based on the Newton-Raphson method for finding the roots of an equation. As the learning rate is calculated automatically, the need for parameter fine tuning and the empirical uncertainty in calculating the learning rate are eliminated. A similar method is described by White (1990).

![Graph of E^G vs w with a tangent line at w showing -\eta \frac{dE^G}{dw}](image)

**Figure 31: Illustration of One Iteration Step (Source: Schmidhuber, 1989)**

### 8.5.7 Delta-Bar-Delta algorithm

(NeuralWare, Inc., 1991; Minai and Williams, 1990)

The delta-bar-delta (dbd) algorithm was developed by Jacobs (1988) in an attempt to increase the convergence speed of back-propagation networks. The slope of the back-propagation error surface varies considerably along different weight directions. By using a constant learning rate, and hence step size, for each weight, the distance moved down the error surface will be considerably larger for an area with a steep slope than one that has a shallow slope. As a result, local minima across steep slopes can be jumped over and descent along shallow slopes is very slow.

The delta-bar-delta algorithm addresses the above problem by using the following four heuristics (Minai and Williams, 1990):

1. All weights have individual learning rates
2. The weights are varied in response to changes in the geometry of the error surface
3. When the sign of the error surface gradient remains unchanged for several iterations, the corresponding learning rate is increased, as this indicates that a minimum lies ahead.

4. When the sign of the error surface gradient changes for several consecutive time steps, the learning rate is decreased, as this indicates that a minimum is being jumped over.

Although the delta-bar-delta algorithm speeds up the learning process considerably, it suffers from the following drawbacks (Minai and Williams, 1990):

1. Using the momentum term in conjunction with the delta-bar-delta algorithm can result in divergent behaviour.
2. The learning rate can sometimes increase to such an extent that the decrease described in 4 above might not be sufficient to prevent wild jumps.

The extended-delta-bar-delta (edbd) algorithm has been developed (Minai and Williams, 1990) to overcome these shortcomings. The following changes are made to the delta-bar-delta algorithm:

1. The increase in the learning rate is changed from a constant to an exponentially decreasing function. As a result, the learning rate increases faster on flat areas than on areas of greater slope.
2. A time-varying momentum term is added to each weight. The momentum term is varied in the same way as the learning rate, resulting in an increased momentum term on plateaux and an exponentially decreasing momentum term in the vicinity of local minima.
3. A ceiling is placed on the learning rate and momentum values in a further attempt to prevent wild jumps and oscillations in weight space.
4. A memory with recovery feature is built into the algorithm. It has the function of saving the best result seen so far, and restarting the search at this point with attenuated learning rate and momentum values when the error becomes greater than a specified value.

8.5.8 Adaptive back-propagation (Silva and Almeida, 1990)

The back-propagation algorithm performs weight updates in the direction that yields the maximum error reduction. However, there is good reason to do exactly the opposite. If the gradient is small, small steps are actually taken, although the error surface has a gentle slope so that bigger steps are more appropriate. If the gradient has a large value, the back-propagation algorithm takes large steps whereas small steps are required to avoid oscillations as a local minimum is being approached.
Adaptive back-propagation makes use of the same principles as the delta-bar-
delta algorithm and the extended-delta-bar-delta algorithms. The adaptive
back-propagation algorithm adjusts the learning rate as learning progresses.
If the sign of a certain component of the gradient remains the same for several
iterations, the error surface has a smooth variation along this axis and an
increase in the learning rate is warranted. If the sign of a component changes
in several consecutive iterations, the learning rate is decreased to avoid
oscillations. The adaption of the learning rate occurs at each epoch and is
exponential in nature. The optimal learning rate values are usually
determined in a few learning cycles, resulting in a rapid reduction of the total
output error.

8.5.9 Dynamic training sets and weighting factors

Back-propagation training can be slowed down by some "poorly-trained
patterns" (Cheung et al, 1990). Such patterns usually belong to infrequent
classes, which form part of random training sets. Generally, frequent patterns
will be dominantly trained, leaving the information of the infrequent patterns
uncaptured. The back-propagation algorithm does not make any concessions
for poorly trained patterns. However, two simple modifications can be made
to the back-propagation algorithm to take poorly trained patterns into account
(Cheung et al., 1990):

1. Using a Dynamic Training Set

The training set can be enlarged by including more examples of infrequent
classes. Poorly trained patterns can be identified as they produce the largest
errors. If poorly trained patterns remain a feature, the size of the training set
is increased further. On the other hand, when the behaviour of the error
function normalises, the training set can be reduced. The training set is thus a
dynamic one, as the size and content of the set is changing as part of training.

2. Using Weighting Factors

The direction in which the system is moving down the error surface should
incorporate weighting factors to take into account the relative errors of
training patterns.

8.5.10 Functional link networks (NeuralWare, Inc, 1991)

Functional link networks differ from traditional back-propagation networks in
that they have additional nodes in the input layer that may dramatically
increase the learning rate. The additional nodes can be added in one of two ways:

1. **Outer Tensor Model**

   In this model, the additional inputs are created by multiplying each component of the input vector with the entire input pattern. For example, if the input vector is given by \( [i_j : 1 \leq j \leq N] \), then the inputs for the tensor model are given by \( [i_j, i_{ji} : 1 \leq j \leq N, j \leq i] \). In this way, no new information is added, but the representation of the input space is enhanced, making the learning process less difficult.

2. **Functional Expansion Model**

   In this model, the additional inputs consist of functions of the existing inputs. For example, in addition to \( i_j \), there might be input nodes for \( \sin (\pi i_j) \), \( \sin (2\pi i_j) \). Again, the representation of the input space is enhanced by mapping the input vectors into a larger pattern space.

**8.5.11 Supplementary learning** (Kimoto and Asakawa, 1990)

Supplementary learning was developed to cater for high-speed learning with a large volume of data. This technique has two important features:

1. Prior to commencement of training, tolerances are set for all output units. As the network learns, only errors exceeding this tolerance are propagated back through the network. As a result, the computational load drops as learning progresses because fewer errors will exceed the tolerance.

2. The learning rate parameter (\( \eta/P^{TR} \)) is adjusted automatically, and is dependent on the size of the training set. The weight update equation is given by:

\[
\Delta w(t) = \frac{\eta}{P^{TR}} \frac{\partial E^G}{\partial w} \mu \Delta w(t-1)
\]  

(37)

where \( P^{TR} = \) the number of training patterns used

As the learning rate is divided by the number of data items that actually require error back-propagation, its value does not need to be changed during the course of training. Initially, the learning rate parameter is relatively small, as the number of learning patterns is large. As learning
progresses, the learning rate parameter is increased automatically because the number of learning patterns decreases.

The result of the above measures is that the supplementary learning procedure is very fast and that there is no need to adjust the learning parameter.

8.5.12 Radial basis functions

The use of radial basis functions is discussed by a number of authors (Hertz et al., 1991; Maren et al., 1990; Broomhead and Lowe, 1988; Jones et al., 1990).

The performance of back-propagation networks can be improved by using radial basis functions as the transfer function in the hidden layer (Broomhead and Lowe, 1988). Each processing element in the hidden layer represents a radial basis function centre. The response of the hidden layer nodes is given by the following function (Jones et al., 1990):

$$ z (I_j) = \sum_{j=1}^{N_{HI}} \lambda_j f^{rb}_j (I_j) $$

(38)

where \( \lambda_j \) = a coefficient

\( N_{HI} \) = the number of basis functions (the number of hidden nodes)

\( f^{rb}_j(.) \) = a radial basis function at node \( j \)

A common radial basis function is the normalised Gaussian function given below (Hertz et al., 1991):

$$ f^{rb}_j (I_j) = \frac{\exp \left( \frac{(I_j - c_j)^2}{2 \sigma_j^2} \right)}{\sum_k \exp \left( \frac{(I_k - c_k)^2}{2 \sigma_k^2} \right)} $$

(39)

where \( c_j \) = the centre of the \( j \)th radial basis function

\( \sigma_j \) = the variance or spread around the centre, \( c_j \), of the \( j \)th function

\( I_j \) = the input to the \( j \)th neuron

An example of a radial basis function is given in Figure 32.
The use of radial basis functions can be advantageous when the network is required to perform continuous function mappings. When radial basis functions are used, the network only requires one hidden layer, as each unit has its own receptive field in the input space; a region centred on \( c_j \) (i.e. on the centre of each hidden layer processing element) with size proportional to \( \sigma_j \). The goal is to pave the input space with these receptive fields. In order to achieve that goal, appropriate receptive field centres, \( c_j \), and widths, \( \sigma_j \), need to be determined.

Hertz at al. (1991) suggest that vector quantisation techniques, such as the competitive learning algorithm, can be used to obtain suitable receptive field centres. The problem of how to adaptively set the radial basis function centres has also been addressed by Moody and Darken (1989) and Saha and Keeler (1990). As the performance of the network is not very sensitive to the variance, it is usually determined by an ad hoc choice, such as the mean distance to the neighbouring centres (Hertz et al., 1991).

Once the input space is covered by appropriate receptive fields, the problem is almost solved. If a particular input lies in the middle of a receptive field, only the corresponding processing element is activated, producing the network output. If a particular input lies somewhere between two receptive field centres, the two processing elements involved are activated to a certain degree, and the network output will be a weighted average of the individual node outputs. As a result, the network provides a sensible, smooth fit to the desired function.

Once the radial basis function centres have been defined, the connection weights from the input layer to the hidden layer nodes are effectively fixed. Consequently, only the weights from the hidden layer nodes to the nodes in the output layer need to be determined during training. These connection weights correspond to the \( \lambda_j \) coefficients in Equation 38, and can be obtained
by linear optimisation. For example, the delta rule can be used to adjust the connection weights, which will minimise the mean squared error between the actual and desired outputs. As a result of this linearity, the learning speed of radial basis function networks is higher than that of conventional back-propagation networks.

One problem associated with radial basis function nets is that they do not lend themselves to electronic implementation (Hertz et al., 1991)

9. APPLICATIONS

Traditionally, ANNs have been used to carry out cognitive tasks performed naturally by the brain including recognising a familiar face, learning to speak and understand a natural language, identifying handwritten characters, retrieving contextually appropriate information from memory and determining that a target seen from different angles is in fact the same object (Vemuri, 1988; Hecht-Nielsen, 1988). They have also been used to obtain a greater insight into the cognitive capabilities of the brain, such as searching, representation and learning. Ultimately, this understanding may be used to construct cognitive models which form the basis of artificial intelligence (Vemuri, 1988).

However, the areas in which neural networks are being used is expanding rapidly. In 1985, the world market for the commercial use of ANNs was virtually non-existent. At present, it is in the order of tens of millions of dollars and is expected to reach approximately $5 billion by the year 2000 (Hubick, 1992). In order to exploit the strengths of ANNs, the problems they are used for should have one or several of the following properties (Burke, 1991; Lippmann, 1987; Josin, 1987):

- A large amount of data is available
- The input-generating distribution is unknown and probably non-Gaussian
- It is expensive to estimate statistical parameters
- Nonlinear relationships are suspected
- The input data is noisy
- A large number of attributes describe the inputs
- On-line decision making is needed
- Possible recalibration of the system may be required
- Uncertainty exists about the rules governing the mapping
- Available models account for some, but not all of the data
- Many hypotheses are pursued in parallel
- High computation rates are required
- Algorithms are unknown or intractable
• Computations are time and resource intensive.

Neural networks provide a viable alternative to expert systems when the following conditions apply (Burke, 1991):

• Rules underlying decisions are not well understood
• Numerous examples of the decisions are available
• A large number of attributes describe the inputs.

Neural models may outperform statistical approaches when (Burke, 1991):

• Underlying distributions are unknown and assumptions of linearity or Gaussian data prove inadequate
• A good deal of noise confounds analysis
• Outliers may exist.

Different applications require the use of distinct classes of networks.

9.1 Networks for Pattern Recognition

When performing pattern recognition tasks, classification and association networks may be used. Classification networks determine which class best represents a set of input values. There are a number of networks that can be used for classification, all of which are based on Kohonen's method of self-organisation. Learning vector quantisation networks are the most popular of these (Lippmann, 1987).

Data association is very similar to classification. Association networks learn associations for error free or ideal patterns and then recognise noisy input patterns as one of the learned patterns. Networks that can be used for this application include the bi-directional associative memory network, the Boltzmann machine, the Hamming network and the Hopfield network (NeuralWare, Inc., 1991).

In the field of pattern recognition, ANNs have been successfully used for:

• recognising handwritten Chinese characters (Yeung and Fong, 1994)
• alphabetical letter recognition (Omatu et al., 1990)
• recognising music symbols (Ezhov and Sala, 1994)
• recognising stock price patterns (Kamijo and Tanigawa, 1990)
• target recognition (Mertzanis and Austin, 1991; Gilmore and Czuchry Jr., 1990)
• contour recognition (Rangarajan et al., 1991; Loncelle, 1990; Oja et al., 1990; Thiaville et al., 1990; Lampinen and Oja, 1990)
• texture recognition (Khotanzad and Lu, 1991; Gosh and Bovik, 1991; Rangarajan et al., 1991, Tirakis et al., 1990; Lampinen and Oja, 1990)
• face recognition (Lampinen and Oja, 1990; Cottrell and Fleming, 1990)
• classifying fingerprints (Mitra et al., 1994)
• classifying two-dimensional shapes (Khotanzad and Lu, 1991)

9.2 Networks for Conceptualisation

Conceptualisation networks analyse input data in order to determine conceptual relationships. Adaptive resonance theory networks and self-organising maps can be used for this application (NeuralWare, Inc., 1991).

9.3 Networks for Filtering

Filtering encompasses the smoothing of an input signal. Recirculation networks can be used for this purpose (NeuralWare, Inc., 1991).

9.4 Networks for Optimisation


Artificial neural networks have the ability to represent and solve difficult combinatorial optimisation problems. Networks with non-adaptive learning rules are used for such tasks. The most widely used of these networks is the Hopfield/Tank network (Hopfield and Tank, 1985).

In optimisation, energy functions resemble cost functions and have to be structured so that they encompass the objective function as well as the relevant constraints. The objective is to minimise this energy function in order to find the best solution to the original problem. To map a problem onto a neural network, the following steps need to be taken (Krovvidy and Wee, 1990):

1. A representation scheme has to be chosen which enables the decoding of the outputs into a solution to the problem
2. An energy function has to be derived, the minimum value of which corresponds to the "best" solution to the problem
3. Values have to be assigned to the parameters of the energy function
4. Connectivities and input bias currents should be derived such that they appropriately represent the instance of the specific problem to be solved.

Neural network methods are capable of computing good, although not necessarily optimal, solutions. They provide viable alternatives to classical optimisation techniques for solving certain optimisation problems.

Reasons for using artificial neural network methods include (Looi, 1992; Wang and Changkong, 1992):

1. Their operation speed is high as a result of massively parallel computation
2. They may be implemented by optical devices, which operate at higher speeds than conventional electronic circuits
3. They have improved fault tolerance
4. In contrast to methods based on the simplex algorithm, the solution procedure is not based on the sequential enumeration of extreme points.

As a result, the use of ANN methods is most appropriate for large-scale, linear programming problems (Wang and Changkong, 1992).

The major drawback with using ANN methods is that they tend to get stuck at local optima. One way to overcome this problem is to find many solutions, starting from different initial states, in an attempt to find the global optimum. However, this method is not very satisfactory and stochastic procedures, such as simulated annealing, can be used to "jolt" the network out of local minima.

Neural network methods are suited to solving resource constrained scheduling problems. In such problems the aim is to achieve a certain degree of scheduling effectiveness while satisfying the constraints imposed by one or more limited resources. Efficient scheduling is often of great importance in the business world. Recently, ANNs have been applied to a variety of scheduling tasks, including:

- integrated scheduling of manufacturing systems (Dagli and Lammers, 1989)
- planning and scheduling in aerospace projects (Ali, 1990)
- space mission scheduling (Gaspin, 1989)
- delivery truck scheduling (Daws et al, 1988)
- large scale plant construction scheduling (Kobayashi and Nonaka, 1990)
- scheduling of astronomical observations for the orbiting Hubble space telescope (Johnston and Adorf, 1992).
9.5 Networks for Prediction

The use of ANNs for prediction has been discussed by various authors (NeuralWare, Inc., 1991; Varfis and Versino, 1990; Jones et al., 1990; Windsor and Harker, 1990; Kimoto and Asakawa, 1990; Walter et al., 1990).

"Prediction" involves the estimation of output values given a set of input values. For example, evaporation rate can be predicted using temperature, humidity and wind velocity as inputs. Primarily, feed-forward networks with nonlinear transfer functions are used for prediction, of which back-propagation networks are the most common. Neural networks have been used successfully for complicated nonlinear forecasting tasks (Varfis and Versino, 1990) and time series prediction (Jones et al., 1990).

Hybrid methods combining regression techniques with neural network techniques can also be used with great success (Walter et al., 1990). Adaptive linear regression methods are used to calculate the predicted value at the next time step as a linear combination of previous values. However, in many cases the actual relation is nonlinear, so that a fixed set of linear prediction coefficients will not produce accurate results. To overcome this problem, a Kohonen network can be used to adaptively discretise the input data. If these discretisations are small enough, linear relations will be a good approximation in each cell, and a separate set of linear prediction coefficients can be estimated for each cell.

Examples of the use of artificial neural networks for prediction are given below.

9.5.1 General applications

9.5.1.1 Economic time series prediction

Varfis and Versino (1990) used a multilayer feed forward network to predict manufacturing industries and energy production indices. Training was carried out using an on-line back-propagation algorithm. The training data consisted of daily manufacturing industries and energy production indices from January 1971 to August 1988. The results obtained were comparable with those obtained using ARMA type models.

Chakraborty et al. (1992) used a simple feedforward network for the prediction of flour prices for the cities of Buffalo, Minneapolis and Kansas City. The neural network model performed better than a well-known time series model described by Tiao and Tsay (1989).
Windsor and Harker (1990) used a standard back-propagation network for predicting the UK ordinary share index. The inputs used include the UK share index, interest rate, the M1 money supply, gross national product, personal disposable income, the balance of payments, the conservative party majority, the savings ratio, the birth rate, the unemployment rate, the US balance of payments and the Dow Jones index. Data from the previous 20 to 25 years was used for training, which was considered enough to capture all typical movement patterns. The back-propagation network was able to perform better than any of the alternative methods tried.

Kimoto and Asakawa (1990) used a back-propagation network with one hidden layer to learn the relationship between various technical and economical indices and the timing for when to buy and sell stocks. The network was used to predict the best time to buy and sell for one month in advance. A standard sigmoid function was employed. The indices that were used as inputs include the vector curve, the turnover, the interest rate, the foreign exchange rate, the New-York Dow Jones average and others. The neural network prediction system made an excellent profit in a simulation exercise, and obtained better results than a multiple regression analysis model.

9.5.1.2 Chaotic time series prediction

Lapedes and Farber (1988) used back-propagation networks to accurately predict future values of a chaotic time series, which frequently occur in nature. In this method, past values of the series are used to predict future values, \( x(t + g) \), where \( g \) is some prediction time step in the future. As \( g \) increases, the prediction accuracy decreases. At large prediction time steps, \( g \), neural network methods can be orders of magnitude more accurate than conventional methods (Lapedes and Farber, 1988).

The neural network determined the following input/output relationship:

\[
o(t + g) = f(i_1(t), i_2(t - \Delta), ..., i_{N_I}(t - N_I \Delta))
\]

(40)

where

- \( o(t+g) \) = the output at discrete time \( t + g \)
- \( i_1, ..., i_{N_I} \) = the network inputs
- \( \Delta \) = a time delay

The time series used by Lapedes and Farber was generated by integrating the Glass-Mackey equation shown below.
\[ X = \frac{ax(t - \tau)}{1 + x^{10}(t - \tau)} - b \cdot x(t) \]  

where \( \tau \) = the width of a time strip  
\( a, b \) = constants

Training was carried out using 500 input/output training sets. Initially, values of \( N^1 = 4 \) and \( \Delta = 6 \) were chosen, but it was found that various choices of \( N^1 \) and \( \Delta \) can affect the prediction accuracy by a factor of two.

Lapedes and Farber (1988) also determined that an iterative method is by far the most accurate method to use for large prediction steps. This involves predicting for a small value of \( g \) (say \( g/k \), \( k = \) integer) initially and using the result of this prediction as the input to the next prediction step (ie. at \( t + 2g/k \)). The iterative process is repeated until the predicted value at time \( t+g \) is obtained.

### 9.5.1.3 Predicting electrical power system loads

Atlas et al. (1990) used a multilayer perceptron network to successfully predict electrical power system loads from an hour to several days in the future. This is useful in the efficient scheduling and utilisation of power generation. The training set used consisted of 523 days of hourly temperature and load data from November 1, 1988 to January 27, 1989. The test set consisted of four days of hourly temperature and load data from January 28 to January 31, 1989. The network inputs included the hour of the day, the two previous temperature readings and the two previous load readings. The output consisted of current load. The predictions obtained were very accurate, with the average percentage error of 1.39%.

### 9.5.2 Applications in water engineering

The stochastic nature of many hydrological data sets, where a particular output is dependent on many unmeasurable actions, makes them suitable for analysis using artificial neural networks (Daniell, 1991). The following are potential application areas of artificial neural networks in the field of water engineering (Daniell, 1991):

- Predicting flows, water consumption and evaporation
- Filtering noisy data
- Optimising water supply systems
- Mapping radar rainfall images
- Classifying data
- Developing flood forecasting models
- Developing flood warning systems
- Developing stream flow models
- Developing regional flood estimates
- Evaluating water quality data

ANNs are particularly suited to the prediction of water quantity and water quality parameters as

- The prediction of one parameter is usually dependent on many other parameters
- The exact relationship between parameters is generally unknown and is likely to be non-linear
- Extensive data sets exist
- The data is usually noisy as a result of measurement and sampling errors.

Examples where ANNs have been used in the field of water engineering include:

**9.5.2.1 Estimation of water consumption / demand**

Daniell (1991), compared the performance of a three-layer back-propagation network with that of an existing linear regression model used to estimate the monthly water consumption for Canberra, Australia. Network inputs included monthly rainfall, the number of rain days per month, monthly evaporation and monthly average temperature. The output consisted of the average daily per capita water consumption for the month under consideration. Training data was taken from 1975 to 1984 and the test set consisted of data from 1985 and 1986. The network consisted of one output node, four input nodes and three hidden layer nodes. The results obtained using the ANN model compared favourably with those obtained from the linear regression model.

Fleming (1994) used an ANN model to forecast monthly water demands for the Northern Adelaide Plains, South Australia. The input variables that were found to be significant include the month of the year, the rainfall in the 'current' month and the rainfall in previous months. The results obtained compared favourably with multiple regression and time series (Box-Jenkins) models.

Zhang et al. (1994) found that ANN models clearly outperformed multiple regression models, ARIMA models and Kalman filtering models for predicting daily water demands. The following five inputs were found to be dominant: daily maximum temperature, the weather condition (i.e. whether it was sunny, cloudy or rainy), precipitation, whether it was a weekday or a Sunday (including public holidays) and the last day's deliveries.
Canu et al. (1990) developed an ANN model to forecast daily water demand and compared the results obtained with those produced by stochastic and statistical methods. They found that ANNs were able to overcome the following limitations of statistical and stochastic methods:

- Stochastic models cannot accommodate inputs such as weather information or the day of the week
- Both the stochastic and statistical models are linear in nature and hence cannot treat nonlinear phenomena such as steep consumption increases.

The ANN model used consisted of a fully-connected multi-layer perceptron network. The inputs used include the past seven consumption values, three pluviometer values, three temperature values and seven values representing the day of the week. The single output value consisted of the forecasted demand. The training set contained 12 years of data from 1976 to 1987 and data from 1988 was used as the test set. The best value for the learning rate was determined by trial and error. The average prediction error was found to be very similar for all methods tested. However, the maximum prediction error is considered to be a better measure of the success of a forecasting tool. When considering this as the criterion, the ANN outperformed all other methods used. It should be noted, however, that the results obtained using the neural network method were highly dependent on the learning rate. It was found that using small learning rates produced better results than using large learning rates.

9.5.2.2 Regional flood estimation

Daniell (1991) used a three-layer back-propagation network for regional flood estimation in the Australian Capital Territory, and compared the results with those obtained using multiple regression analysis. The ten catchment parameters that were used as inputs include area, slope, length, fall, precipitation, fractions of the catchment under urban, rural and forest use, annual series skew and partial series skew. There were seven nodes in the hidden layer and five nodes in the output layer representing the 2, 5, 10, 50 and 100 ARI flood flows. The hyperbolic tangent transfer function was used. The results obtained using the ANN model were very impressive and compared favourably with those obtained using a linear regression model.

9.5.2.3 Predicting salinity

DeSilets et al. (1992) developed a neural network model for predicting salinity in the Chesapeake Bay, U.S.A., and have compared its performance with that of a regression model. Predicting salinity values is important, as salinity affects the oyster populations in the area.
A total of ten neural network models were developed; five using bottom-of-the-bay-only data and five using data from all depths. Each of the five models corresponded to various regions of the bay; the upper bay, the middle bay, the lower bay, the tributaries and the entire bay. The regression model was used as a measure of comparison and to gain an understanding of the factors affecting salinity. It was found that location, depth and time of the year had a direct impact on salinity, and were used as inputs to the network. The solitary network output was the predicted salinity value. Training and testing data were collected from 34 stations throughout the bay, and a total number of 36,258 observations were available. The network used a back-propagation algorithm, a sigmoid transfer function, was fully connected and the number of nodes in the hidden layer varied from one to three. The optimum learning rate for the various models varied from 0.2 to 0.8 and the optimum momentum value varied from 0.1 to 0.9. These values were obtained by trial and error.

A stopping rule was used to determine the length of the training period. Training was stopped when one of the following occurred:
- The number of passes through the training set was at least 100 and the worst absolute error at the output node, once all patterns were considered, was less than 20%.
- The number of iterations reached 2000.

Three main results were obtained:
1. The ANN model produced a lower mean absolute error than the regression model in 18 out of 20 cases.
2. The tails of the prediction error distribution produced by the neural network were shorter than those of the regression model. Hence the maximum errors obtained from the model were less.
3. The neural network model had more errors in the smallest error range and fewer errors in the largest error range.

Maier and Dandy (1995a) developed a univariate ANN model for forecasting salinity in the River Murray at Murray Bridge, South Australia, up to fourteen days in advance, and compared its performance with that of a univariate time series (ARIMA) model. They found that the the ARIMA model outperformed the ANN model for short-term forecasts. However, the reverse was true for longer term forecasts.

Maier and Dandy (1993a, 1993b) extended the univariate ANN model to include upstream salinities, flows and river levels, resulting in a considerable improvement in the 14 day forecast. A comparison with a multivariate time series (VARIMA) model (Maier and Dandy, 1994, 1995b) showed that the multivariate ANN model performed slightly worse than the VARIMA model
for short-term forecasts, but performed significantly better when a forecasting period of 14 days was used.

9.5.2.4 Flow prediction

Lachtermacher and Fuller (1994) used ANN methods to obtain univariate flow predictions for the Gota River. The one-step ahead predictions were found to be slightly better than those obtained using other time series models, including ARMA models.

Karunanithi et al. (1994) developed a neural network model to predict the flow at an ungauged site in the Huron River, Michigan, using known flows at upstream and downstream locations. The predictions obtained using the neural network model were more accurate and less susceptible to noisy data than those obtained using a more conventional power model.

9.5.2.5 Runoff prediction

Zhu and Fujita (1994) successfully used ANN methods to obtain on-line and off-line runoff forecasts. The relationship between rainfall and runoff is dependent on many variables that interact with each other in a complex, usually non-linear, manner. Consequently, ANNs are well suited to this application.

Halff and Halff (1993) developed an ANN model for predicting runoff hydrographs given the rainfall at the 25 preceding time steps. The resolution used was five minutes. A three-layer network was used, with five nodes in the hidden layer and one node in the output layer. The data from four storms were used for training, while testing was carried out using the data for one independent storm event. The results obtained were encouraging.

9.5.2.6 Prediction of nutrient concentrations

Daniell and Wundke (1993) used ANNs to successfully derive continuous records of total nitrogen and total phosphorus using continuous flow, conductivity, turbidity, temperature, oxidised nitrogen and soluble phosphorus records as inputs, in order to calculate nutrient loads of total nitrogen and total phosphorus. The neural network was found to perform better than a simple regression model.

9.5.2.7 Forecasting minimum temperature

Schizas et al. (1994) used a backpropagation network to forecast daily minimum temperatures at Larnaca Airport, Cyprus. The inputs used include
the total cloud cover, wind direction, wind speed, visibility, the present weather conditions, atmospheric pressure, dry-bulb temperature, wet bulb temperature, the low cloud amount, astronomical day length and the observed minimum temperature of the previous night. The results obtained indicate that the neural network approach appears to be a viable alternative to traditional statistical techniques when limited historical data are available.

9.5.2.8 Prediction of blue-green algae concentrations

Maier and Dandy (1995c) developed a multivariate ANN model which was able to successfully predict the relative magnitude and timing of incidence of a specific genus of blue-green algae in the River Murray at Morgan, South Australia. The model inputs included turbidity, colour, temperature, flow, total phosphorus, soluble phosphorus and total nitrogen at various lags.

9.5.2.9 Design of a reliable groundwater remediation strategy

Ranjithan et al. (1993) used ANNs as a screening tool to assist in the design of reliable groundwater remediation strategies, which are determined with the aid of a management model. In many instances, the spatial variation of hydraulic conductivity is unknown, so that the management model has to be run for a large number of equally probable hydraulic conductivity distributions. In order to minimise the number of scenarios that have to be evaluated, a neural network model can be trained to predict the level of criticalness of a particular distribution of hydraulic conductivities, so that the management model only has to be run for the cases that are most critical.

10. SUMMARY

Artificial neural networks (ANNs) are loosely based on the structure of the brain. They consist of a network of processing elements, which are joined by weighted connections. Typically, the input of one such processing element is the weighted sum of the outputs of the nodes connected to it. This input is generally passed through a non-linear transfer function, producing the node output, which in turn provides the input to many other processing elements. Such networks have the ability to determine relationships between sets of input data and, depending on the learning rule used, corresponding output data. This relationship is "learned" by adjusting the connection weights on presentation of the training data.

There are many different types of networks, which mainly differ in the way they learn and the way the individual nodes are connected. There are two major types of learning; supervised learning and unsupervised learning.
When supervised learning rules are used, the inputs, as well as the corresponding outputs, are presented to the network during training. When unsupervised learning rules are used, the network is only presented with input stimuli. The network then discovers patterns, features, regularities or categories in the input data. Node connectivity affects the way information flows through the network. Connections can be feedforward, feedback and lateral. Networks can also be fully or partially connected.

Back-propagation networks (which are the type of network most widely used today) are multi-layer, feedforward networks which use a supervised learning rule. The weights are adjusted by performing gradient descent in weight space. One of the problems users of back-propagation networks are faced with is the optimisation of the network geometry and the internal network parameters, such as the learning rate and the momentum, although some guidance is available.

ANNs have been successfully used for a number of applications, including pattern recognition, optimisation and forecasting. They are particularly suited to the prediction of environmental and hydrological variables, and have been applied in a number of areas, including the estimation of water consumption / demand, regional flood estimation, the prediction of salinity, flow, runoff, nutrient concentrations, minimum temperatures, blue-green algae concentrations and the design of reliable groundwater remediation strategies.
11. REFERENCES


Cybenko, G. (1988) "Continuous Valued Neural Networks with Two Hidden Layers are Sufficient", Technical Report, Computer Science Department, Tufts University, Medford, MA.


of Civil and Environmental Engineering, The University of Adelaide, February.


Maier, H.R. and Dandy, G.C. (1995c) "Modelling Cyanobacteria (Blue-Green Algae) in the River Murray Using Artificial Neural Networks", paper accepted for Modsim '95, Newcastle, Australia, November 27-30.


APPENDIX A: NOTATION

a A constant
EG(t) The global error function at discrete time t
EL The local error at a particular processing element
EN An energy function
f^rb_j(.) A radial basis function at node j
F(.) The transfer function
g The forecasting period
i A network input
i A vector containing the network inputs
I_j The activation level of node j
NH1 The number of nodes in the first hidden layer
NH2 The number of nodes in the second hidden layer
NI The number of nodes in the input layer
NK The number of nodes in the Kohonen layer of an LVQ network
NO The number of nodes in the output layer
NP The number of nodes in the previous layer
NTOT The total number of nodes in the entire network
NW The number of connection weights
o^d_k(t) The desired (actual, historical) output at discrete time t
o^p_k(t) The predicted output at discrete time t
o^p A vector containing the predicted network outputs
PTE The number of patterns (samples) in the testing set
PTR The number of patterns (samples) in the training set
T_j The threshold for node j
w_{ji} The connection weight between nodes i and j
WN A matrix containing the network connection weights
x_i The input from node i
y_j The output of node j
α A parameter
β^v A vigilance parameter
Δw_{ji} The amount the connection weight between nodes i and j is changed during one weight update
ε The epoch size
η The learning rate
μ The momentum value
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAE</td>
<td>Average absolute error</td>
</tr>
<tr>
<td>AAPE</td>
<td>Average absolute percentage error</td>
</tr>
<tr>
<td>ADALINE</td>
<td>Adaptive linear element</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial neural network</td>
</tr>
<tr>
<td>ART</td>
<td>Adaptive resonance theory</td>
</tr>
<tr>
<td>BAM</td>
<td>Bi-directional associative memory</td>
</tr>
<tr>
<td>CPU</td>
<td>Central processing unit</td>
</tr>
<tr>
<td>dbd</td>
<td>Delta-bar-delta</td>
</tr>
<tr>
<td>edbd</td>
<td>Extended delta-bar-delta</td>
</tr>
<tr>
<td>LVQ</td>
<td>Learning vector quantisation</td>
</tr>
<tr>
<td>MADALINE</td>
<td>Multiple adaptive linear element</td>
</tr>
<tr>
<td>MAE</td>
<td>Maximum absolute error</td>
</tr>
<tr>
<td>M-P</td>
<td>McCulloch - Pitts</td>
</tr>
<tr>
<td>NEP</td>
<td>Network emulation processor</td>
</tr>
<tr>
<td>NNN</td>
<td>Natural neural network</td>
</tr>
<tr>
<td>RMSE</td>
<td>Root mean squared error</td>
</tr>
<tr>
<td>VLSI</td>
<td>Very large scale integration</td>
</tr>
</tbody>
</table>
UNIVERSITY OF ADELAIDE

Reports from the Department of Civil and Environmental Engineering

R - Series (Research)

R1 Cheung, Y.K., Yeo, M.F., Cumming, D.A.,
Three dimensional analysis of flexible pavements with special reference to edge loads, July 1975.

R2 Bodley, W.E.,
Energy dissipation in mammalian arteries - An assessment of the distribution of energy dissipation between the blood and the vessel wall, November 1975.

R3 Kasemset, C., Cheung, Y.K., Khatua, T.P.,

R4 Melksham, M.H., Yeo, M.F., Cheung, Y.K.,
The initial effects of water jet cleaning on superheater tubes, 1976.

R5 Kay, J.N., Abel, J.F.,
A design approach for circular buried conduits, December 1976.

R6 Delcourt, C., Cheung, Y.K.,

R7 Brooks, D.S., Hirst, M.J.S.,
The low cycle fatigue behaviour of joints in reinforced concrete frames, April 1977

R8 Moxham, K.E., Dwight, J.B.,

R9 Cheung, Y.K., Swaddiwudhipong, S.
Static analysis of frame shear wall structures, June 1977.

R10 Kay, J.N.,
Factor of safety for piles in cohesive soils, July 1977.

R11 Wylie, E.B., Bodley, W.E.,
Control of unsteady free-surface flow with gates, December 1977.

R12 Bodley, W.E., Wylie, E.B.,
Control of transients in series channel with gates, December 1977.

R13 Kay, J.N., Qamar, M.I.,

R14 Hirst, M.J.S.,
The structural behaviour of composite walls and beams, June 1978.

R15 Hirst, M.J.S.,
Application of the finite-stringer theory of the interaction of walls and their supporting structures, June 1978.
R16  Hirst, M.J.S., Kay, J.N.,  
_A design method for shear wall and pile interaction_, June 1978.

R17  Kay, J.N.,  
_Incorporation of probability in the design of buried concrete pipe_, August 1978.

R18  Kay, J.N., Flint, R.C.L.,  
_Design charts for large span metal arch culverts_, August 1978.

R19  Hutton, S.G., Sved, G.,  
_The determination of critical loads and natural frequencies of non-uniform beams_, September 1978.

R20  Brooks, D.S., Sved, G., Payne, D.C.,  
_The stiffness of partially cracked brick walls_, March 1979.

R21  Kay, J.N., Hain, S.J.,  
_A design method for buried concrete pipe_, October 1979.

R22  Kay, J.N., Mitchell, P.W.,  
_A down hole plate load test for in situ properties of stiff clays_, October 1979.

_Instrumentation of a corrugated steel-soil arch overpass at Leigh Creek, South Australia_, November 1979.

R24  Warner, R.F.,  
_Serviceability of cracked partially prestressed concrete members: Test and analysis_, January 1980.

R25  Payne, D.C., Sved, G., Brooks, D.S.,  
_Numerical analysis of brick columns subject to axial and lateral loads_, May 1980.

R26  Hirst, M.J.S.,  
_Solar heating of bridges_, April 1980.

R27  Bodley, W.E., McIntyre, A.K.,  
_Depth control during demand changes in open channels_, August 1980.

R28  Kay, J.N.,  
_A systematic approach to safety margins in pile foundation design_, July 1980.

R29  Warner, R.F.,  
_Design criteria for the new structures code_, August 1980.

R30  Brooks, D.S.,  
_Strength and stability of brick masonry walls_, October 1980.

R31  Sved, G., Brooks, D.S., Yeo, M.F.,  
_Investigations on cruciform welded joints with root gaps_, August 1980.

R32  Crawley, D.B.,  
_An evaluation of creep and shrinkage of high strength concrete in South Australia_, December 1980.
R33 Warner, R.F.,

R34 Kay, J.N., Avalle, D.L.,
*Application of the Screw Plate to stiff clays*, March 1981.

R35 Hirst, M.J.S.,
*Designing Slender Webs for Steel Bridge Beams*, March 1981.

R36 Flint, R.C.L.,
*Design Chart for Large-Span Corrugated Metal Arch Structures*, September 1981.

R37 Sved, G., Yeo, M.F., Brooks, D.S.,

R38 Dandy, G.C., Neil, R.H.,

R39 Payne, D.C., Brooks, D.S.,

R40 Brooks, D.S., Payne, D.C.,

R41 Flint, R.C.L., Kay, J.N.,

R42 Kay, J.N., Cavagnaro, R.L.,

R43 Kay, J.N., Flint, R.C.L.,

R44 Kay, J.N., Parry, R.H.G.,

R45 Dandy, G.C.,

R46 Ewers, J. R.,

R47 Arnold, M.,

R48 Hirst, M.J.S.,
*Thermal Loading of Concrete Bridges*, August 1982.

R49 Hirst, M.J.S.,
*Thermal Loading of Concrete Roofs*, August 1982.

R50 Hirst, M.J.S.,
R51 Hirst, M.J.S.

R52 Sved, G., Payne, D.C. and Brooks, D.S.

R53 Moxham, K. E., Brooks, D.S.
*A Dual Servo-Controlled Test Rig*, May 1983.

R54 Mitchell, Peter W., and Arnold, Maurice,

R55 Arnold, M.

*Recent Developments in Screw Plate Testing in Adelaide*, October 1983.

R57 Dandy, G. C.

R58 Kay, J.N. Kokkinakis, T., Tallidira, M.C.
*Compressible Conduits Under Deep Fill Loads - A Model Study*, October 1983

R59 Avalle, D.L., Flint, R.C.L., Kay, J.N.
*Large Scale Test Facility for Model Conduits and Arches Buried in Soil*, October 1983.

R60 Ferreira, L.J.A.
*Urban Transport Management (UTM) and Energy Consumption - a review of evidence*, November 1983

R61 Warner, R.F. and Yeo, M.F.
*Collapse Behaviour of Concrete Structures with Limited Ductility*, January 1984

R62 Ahmad, Azman and Warner, R.F.
*Ductility Requirements for Continuous Reinforced Concrete Structures*, January 1984

R63 Behan, J.E. and Warner, R.F.
*Fatigue Properties of Temcore Reinforcing Bars*, January 1984

R64 Foo, M.H. and Warner, R.F.
*Fatigue Tests on Partially Prestressed Concrete Beams*, January 1984

R65 Hirst, M.J.S.
*Measuring the Temperature of Concrete Structures*, February 1984

R66 Ferreira, L.J.A.
*Modelling Urban Fuel Consumption*, February 1984

R67 Ferreira, L.J.A.
*Traffic Signal Coordination - a survey of drivers in Adelaide*, April 1984
Ferreira, L.J.A.  
*The Effects of traffic signal coordination along an arterial in Adelaide*, April 1984

Warner, R.F. and Dandy, G.C.  
*Multistage Decision Models for the Assessment of Structural Defects*, May 1984

Warner, R.F.  
*Design of Concrete Structures for Fatigue and Reliability*, April 1985

Crawley, D.B.  
*Civil Engineering Design Management - Teaching by Project*, April 1985

Crawley, D.B.  
*Using a Project Design to Integrate Civil Engineering Studies*, August 1985

Crawley, D.B.  
*Preliminary Design for a Project: The design brief (for building and other structural projects)*, August 1985

Kay, J.N.  
*An Efficient Soils Exploration and Testing Program*, October 1986

Kay, J.N.  
*Examination of a New Proposal for Determination of Atterberg Limits*, August 1986

Varga, J.  
*Change of Flow Pattern in Model Bin (Silo Pressure Abatement Study - SACBH)*, October 1986

Dandy, G.C.  
*Factors which Affect Residential Water Consumption in Adelaide*, November 1986

Kay, J.N. and Rigon, C.  
*Instrumentation of a Bebo Arch Constructed at Byron Bay, New South Wales*, November 1986

Dandy, G.C.  

Wong, K.W., Yeo, M.F. and Warner, R.F.  
*Collapse Behaviour of Reinforced Concrete Frames*, August 1987

Kgloboko, K., Wyche, P.J. and Warner, R.F.  
*Collapse Behaviour and Ductility Requirements in Partially Prestressed Concrete Bridge Girders*, April 1988

Oehlers, D.J.,  
*Reinforced Concrete Beams with Steel Plates Glued to their Soffits; Prevention of Plate Separation Induced by Flexural Peeling*, August 1988
R81 Oehlers, D.J.
*The Derivation of the Fatigue Endurance of Stud Shear Connections in Composite Steel and Concrete Beams*, November 1988

R82 Oehlers, D.J.
*A New Approach to the Design of Stud Shear Connectors in Composite Bridge Beams*, June 1989

R83 Hua, W.G., Griffith, M.C. and Warner, R.F.
*Diagnosis and Assessment of Defective Concrete Structures*, August 1989

R84 Oehlers, D.J.
*Shear Strength of Longitudinally Cracked Composite Beams*, August 1991

R85 Oehlers, D.J.
*Design of Shear Connectors in Composite Steel and Concrete Beams*, August 1991

R86 Griffith, M.C.
*Performance of Unreinforced Masonry Buildings during the Newcastle Earthquake, Australia* August 1991

R87 Oehlers, D.J.
*Uni-Directional Fatigue Tests on Stud Shear Connectors* January 1992

R88 Oehlers, D.J.
*Experimental Tests on Plated Beams* January 1992

R89 Oehlers, D.J.
*Profiled Beam Construction* January 1992

R90 Warner, R.F. and Smith C.A.
*Design of Circular Concrete Columns using Spreadsheets* February 1992

R91 McPheat, J., Simpson, A.R. and Gibbard, M.

R92 Simpson, A.R. and Zhu, Yin Xiong
*An Expert System for Turbidity Control Advice in Water Treatment Plant Operations* May 1992

R93 Murphy, L. and Simpson, A.R.
*Pipe Optimisation using Genetic Algorithms* June 1992

R94 Connarty, M.C. and Dandy, G.C.

R95 Dandy, G.C. and Crawley, P.
*Application of Linear Programming to Planning and Operation of Reservoir Systems with Water Quality Considerations* April 1992
R96 Connarty, M.C. and Dandy, G.C.
*A Methodology for Optimum Sequencing of Water Supply Projects*  
October 1992

R97 Kenyon, J.M. and Warner, R.F.
*Refined Analysis of Non-Linear Behaviour of Concrete Structures*  
August 1992

R98 Oehlers, D.J.
*Stud Shear Connectors in Composite Beams with Sloping Haunches*  
October 1992

R99 Bergant, A and Simpson, A R  
*Range of Validity of the Discrete Vapour Cavity Model for Pipeline WaterHammer*  
March 1994

R100 Mitani, I and Warner, R F  
*Design of Slender Concrete-filled Steel Tube Columns*  
February 1993

R101 Maier, H and Dandy, G C  
*Application of Artificial Neural Networks to the Prediction of Salinity*  
February 1993

R102 Griffith, M C  
*Seismic Performance of Australian Designed Reinforced Concrete Framed Buildings*  
February 1993

R103 Oehlers, D J  
*Stiffening Existing Reinforced Concrete Slabs by Bonding Steel Plates to Their Tension Face*  
March 1993

R104 Oehlers, D J  
*Strengthening Existing Reinforced Concrete Slabs by Bonding Steel Plates to Their Tension Face*  
March 1993

R105 Oehlers, D J  
*A New Fatigue Methodology for the Design and Assessment of Stud Shear Connectors in Composite Bridge Beams*  
April 1993

R106 Oehlers, D J  
*Detailing Rules for the Side Cover to Stud Shear Connectors to Prevent Splitting*  
August 1993

R107 Oehlers, D J and Sved, G  
*Fracture of Stud Shear Connectors in Composite Steel and Concrete Beams*  
September 1993

R108 Ghosh, A, Oehlers, D J and Wahab, M A  
*A New Fatigue Design and Analysis Approach Based on Residual Strength*  
October 1993

R109 Murphy, L J, Simpson, A R and Dandy, G C  
*Pipe Network Optimization Using an Improved Genetic Algorithm*  
December 1993

R110 Hassanli, A M and Dandy, G C  
*Optimum Design of Drip/Trickle Irrigation Systems on Flat Terrain*  
December 1993
R111  Fleming, N S and Daniell, T M  
_Sustainable Water Resources Management: an Australian Perspective_ January 1994

R112  Sved, G  
_The Effect of Yielding in the Outer Fibres of a Beam on the Shearing Stresses at the Neutral Axis_ March 1994.

R113  Kawano, A and Warner, R F  
_Effect of Model Formulation on Numerical Errors in Step-by-Step Creep Calculations for Concrete_ March 1994

R114  Oehlerl, D J, Ghosh, A and Wahab, M A  
_Fatigue Design and Analysis of Structural Components Based on Residual Strength_ March 1994

R115  Fleming, N S  
_Forecasting Water Consumption in the Northern Adelaide Plains, SA, using Artificial Neural Networks, Regression and Time Series Methods_ March 1994

R116  Jaksa, M B and Kaggwa, W S  
_A Micro-Computer Based Data Acquisition System for the Cone Penetration Test_ May 1994

_Modelling the Lateral Spatial Variation of the Undrained Shear Strength of a Stiff, Overconsolidated Clay using an Horizontal Cone Penetration Test_ September, 1994

R118  Morgan, P J and Kaggwa, W S  
_Review of the Behaviour and Constitutive Modelling of Expansive Clay Soils_ September, 1994

R119  Fleming, N S.  
_An Investigation Into Rainfall-Runoff Relationships_ September, 1994

R120  Aesche, B G and Simpson, A R  
_Optimisation of Pipe Networks using an Evolution Strategy_ November 1994

R121  Maier, H R and Dandy, G C  
_Use of Neural Networks for Determining the Inputs for Multivariate Time Series Models_ November 1994

R122  Jaksa, M B, Brooker, P I and Kaggwa, W S  

R123  Hassani, A M, Dandy, G C  

R124  Kawano, A and Warner, R F  
_Model of Stress-Induced Shrinkage for the Analysis of Concrete Structures_ January 1995
R125 Kawano, A and Warner, R F
*Nonlinear Analysis of the Time-Dependent Behaviour of Reinforced Concrete Frames* January 1995

R126 Davies, C M and Dandy, G C
*Modelling Residential Water Demand in Adelaide Using Regression Analysis* May 1995

R127 Maier, H R and Dandy, G C
*Comparison of the Box-Jenkins procedure with artificial neural network methods for univariate time series modelling* June 1995

R128 Bergant, A and Simpson, A R
*Water Hammer and Column Separation Measurements in an Experimental Apparatus* June 1995

R129 Maier, H R and Dandy, G C
*Application of Multivariate Time Series Modelling to the Prediction of Salinity* June 1995

R130 Hassanli, A M and Dandy, G C
*An Optimisation Model for Design of Multiple Sub-unit Pressure Irrigation Systems* July 1995

R131 Maier, H R
*A Review of Artificial Neural Networks* August 1995
UNIVERSITY OF ADELAIDE

Reports from the Department of Civil and Environmental Engineering

G - Series (General)

G1 Brooks, D.S., Crawley, D.B., Moxham, K.E., Fargher, P.J. and Miles, R.T.
"Education in Civil Engineering Design," July 1976

G2 Arnold, M.A.

G3 Arnold, M.A.
"Salt-damp - Overseas Research and Practice," March 1978

G4 Brooks, D.S.
"Glass Reinforced Cement - A New Material from Old Ingredients," April 1978

G5 Crawley, D.B.
"Report on Institution of Engineers Conference - Engineering Education 1978,
July 1978

G6 Cumming, D.B.
"Australia's Engineering Heritage," June 1979

G7 Cumming, D.B.
"Engineering Heritage," February 1981

G8 Warner, R.F.
"Structural Assessment of Existing Concrete Buildings," January 1981

G9 Bridgeland, R.J.
"Civil Engineering at The University of Adelaide 1878--1974," December 1981

G10 Cumming, D.A.
"Processing of Copper Ores in South Australia," May 1982

G11 Cumming, D.A.
"The Kapunda Mine 1842 to 1906," May 1982

G12 Cumming, D.A.
"Identification and Evaluation of Items of Engineering Heritage," May 1982

Obsolete G13-R
see G17
Cumming, D.A.
"Notes on Engineers, Technicians, Manufacturers and Contractors who have
Served South Australia, (Part I)," March 1983

Obsolete G14
see G19
Cumming, D.A.
"Notes on Engineers, Technicians, Manufacturers and Contractors who have Served
South Australia, (Part II)," March 1983

G15 Dillon, P.J.
"Artificial Groundwater Recharge: A Case Study in Maharashtra, India," June
1983

G16 Cumming, D.A.
"The Burra Mine and Smelter," May 1984

G17 Cumming, D.A.
"Notes on Engineers, Technicians, Manufacturers and Contractors who have Served
South Australia, (Part I)" July 1984
G18  Cumming, D.A.  
*Notes on Engineers, Technicians, Manufacturers and Contractors who have Served South Australia, (Part III)* July 1984

G19  Cumming, D.A.  
*Notes on Engineers, Technicians, Manufacturers and Contractors who have Served South Australia, (Part II)* November 1984

G20  Crawley, D.B., McPherson, I.  
*A Strategy for Building Structure Design in Moderate Risk Earthquake Regions, Applied in South Australia.* November 1985

G21  Kay, J.N.  
*Workshop on the Use of Statistical Methods for Highway Construction,* November 1987

G22  Moxham, K.E. and Warner, R.F.  
*Structural Engineering into the Next Millenium,* February 1991

G23  Daniell, T.M.  
*Flexibility in Management - Specialized Training for Engineering Undergraduates* July, 1991

G24  Fleming, N.S.  
*Assigning Weights to Multiple Alternative by Personal Preferences - Evaluation of postgraduate priorities for working environment improvement,* September 1992

G25  Maier, H R and Dandy, G C  
*Blue Green Algae in the River Murray* December 1994