INTEGRATING THE KEY APPROACHES OF NEURAL NETWORKS

by

Beverley Robin Howard

Submitted in fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In the subject

OPERATIONS RESEARCH

At the

UNIVERSITY OF SOUTH AFRICA

SUPERVISOR: PROFESSOR WOLVAARDT

December 1999

Acknowledgements

I record my sincere thanks to Professor Wolvaardt for his guidance in the presentation of this thesis.

My thanks to my wife Pauline and daughter Melissa for their tolerance throughout my neglect of family duties in pursuit of my studies.

I am indebted to Miss Herculine Retief for the encouragement by carrying an additional work load, enabling time for the completion of this work and for help with the final reproduction.

For their help in communication and submission of work to the University of South Africa I owe thanks to Miss Boshoff, Mrs Pretorius and Mrs Swanepoel.

I would like also to thank the members of Professors Swanepoel's "Department of Operations Research" that have befriended their oldest student on his visits to the seat of learning.

Table of contents Page

List of Figures

List of Tables

Summary

The thesis is written in chapter form. Chapter 1 describes some of the history of neural networks and its place in the field of artificial intelligence. It indicates the biological basis from which neural network approximation are made.

Chapter 2 describes the properties of neural networks and their uses. It introduces the concepts of training and learning.

Chapters 3, 4, 5 and 6 show the perceptron and adaline in feedforward and recurrent networks particular reference is made to regression substitution by "group method data handling. Networks are chosen that explain the application of neural networks in classification, association, optimization and self organization.

Chapter 7 addresses the subject of practical inputs to neural networks.
Chapter 8 reviews some interesting recent developments.
Chapter 9 reviews some ideas on the future technology for neural networks.
Chapter 10 gives a listing of some neural network types and their uses.
Appendix A gives some of the ideas used in portfolio selection for the Johannesburg Stock Exchange.

Chapter 1.

FOUNDATIONS OF NEURAL NETWORKS

Summary

This chapter offers some history and a brief description of the field of artificial intelligence as the domain in which neural networks function. It further describes other artificial intelligence subjects that are related, or interface with neural networks.

The biological basis from which neural network approximations stem are described, together with simple artificial neurons to model aspects of these cells.

1.1 Artificial intelligence

Definition: Artificial Intelligence, is the science of making machines do things that would require intelligence if done by human beings. Minsky (1968).

For many years man has sought to model functions of the brain activities, from simple animal forms to the extremely complex human brain. The field of artificial intelligence (AI) has developed with increasing knowledge of the way in which the brain functions and increasing ability to partially emulate those functions through advances in technology.

John von Neumann (1903 to 1957) pioneered the development of the computer to a level at which simple AI models became possible. His work evolved the design and application of computing machines. It is he whom, in 1946, is credited with the proposal for computer working using the binary system, and the concept that operating instructions for the machine could be stored in memory. Sometime later his team at Princeton University built the first computer in which information was processed a 'word' at a time rather than serially.

Traditional AI systems make the important assumption of the physical symbol system hypothesis. Newell and Simon (1976) contributed to the understanding of the physical symbol system.

Definition: A set of entities, called symbols, which are physical patterns that can occur as components of another type of entity, called an expression or symbol structure. A physical symbol system is a machine that produces through time an evolving collection of symbol structures.

Newell, Shaw and Simon (1963) used their early AI experiences with a logic theory machine to prove theorems from the first chapter of Whitehead and Russell (1950) Principia Mathematica. Subsequent attempts to use the theorem proving technique to build a general solution system for complex problems have not proved successful.

Gilling and Brightwell, (1982) indicate the vast gap that exists between human intelligence and AI. They note that since the human brain has an estimated 10^{11} nerve cells, it follows that state of the art super computers can only display intelligent behaviour in a narrow domain.

The logic approach to AI thinking moved in the 1970 decade to a knowledge approach, human knowledge being represented symbolically. LiMin Fu (1994) suggested that it is knowledge that makes the system intelligent and not the inference mechanism.

The artificial intelligence field has inspired several areas of development that include, expert systems, optimization, fuzzy logic, and neural networks.

1.1.1 Expert systems

Feigenbaum (1982) offers the following definition for an expert system.
Definition: An expert system is an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution.

Expert systems are a significant part of artificial intelligence. Its uses differing widely from financial loan appraisal, through to equipment replacement and medical diagnosis.

The system may have a self-learning approach in that feedback of the results from an action may influence the later decisions. Most systems consist of two principal parts, the first being the experts knowledge base which provides the guiding rules or judgments based on the chosen expert opinion. The second part interprets the knowledge base in relation to the particular problem being presented.

1.1.2 Optimization

In the field of "operations research" many methods are used that provide reasoned answers to posed problems. The methods attempt to obtain an optimal solution to a problem under given constraints. The practical problem is modelled mathematically and the model forecasts indicate an optimal solution. The system performance is usually measured in terms of a minimization or maximization criterion.

The basic problem of optimization is to arrive at the best decision for a given set of circumstances. There may be difficulties in deciding the best decision, in the light of the different perspectives that may be viewed of the problem.

Optimization may use the algorithmic approach of the computer, as demonstrated in linear and dynamic programming. It may also be achieved by the use of neural networks in such varied applications as partial pattern recognition and the travelling salesman problem. There is a difference in that the neural network optimum would not be a proven optimum.
Note that the knowledge base is in the determination of objectives and constraints. If the wrong inputs are given algorithms will not provide optimal solutions to the problem.

1.1.3 Fuzzy logic

Zadeh (1965) states that the theory of fuzzy logic is primarily concerned with clarifying ambiguity in natural language. It has evolved from the work done in the theory of fuzzy sets.

Giarratano and Riley (1994) offer this guidance "Conventional set theory defines set membership in terms of characteristic functions (CF) in which the CF is 1 if the object is a member and 0 if the object is not a member. An object may have partial membership of a fuzzy set, the degree of membership being given by the compatibility function".

Fuzzy logic has been practically applied in many situations from camera tracking to environmental decision. Maier and Sherif (1985) supply more than 450 references of fuzzy logic applications.

1.1.4 Neural networks

This is the aspect of artificial intelligence that is to be the core work of this thesis. Neural networks may be described as interconnected networks of simple processing units (artificial neurons) that exhibit some properties of the biological nervous system. Each processing element (neuron) receives a part of the problem to be solved. This gives the power of the network which is achieved by a technique known in neural networks as parallel processing in which computations are spread simultaneously between several processing elements.

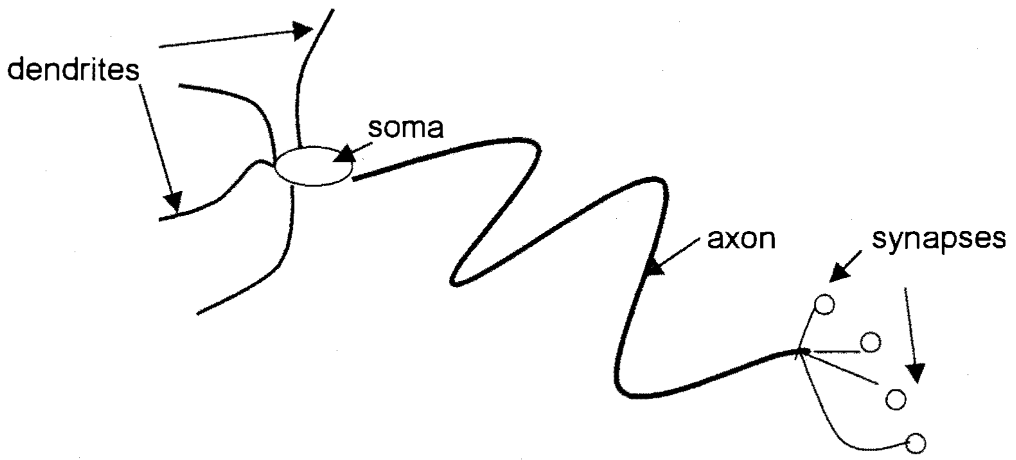
Neural networks have demonstrated an ability to produce good results when applied to the tasks of artificial intelligence. Examples include pattern recognition in literary styles Aston University (1993), and DARPA (1988) list a wide variety of applications ranging from Aerospace to Telecommunications.

1.2 Biological origins

Neural network models have evolved from the perceptions of biological behaviour, some researchers believing that neurons are the computational units of the brain.

Anderson (1995) explains that knowledge of neuroscience is helpful in understanding why models are founded in certain approximations about biological neurons and the relative merits of those approximations.

There are many forms in real neurons following the biological need. The following diagram illustrates the classic generic neuron, modelled after spinal motor neurons.



A classic generic neuron.

Figure 1.1

Neurons are cells with inputs from the dendrites, named because of their tree like appearance. The cell body is called the soma, from which the axon serves as a transmission line, sometimes with several branches, to the terminal arborization. The special terminals are known as synapses and influence the activities of other cells.

Current neural network theory assumes that the varying synaptic strengths of several interconnected neurons are the key to the computations performed. Anderson (1995) notes that there is a shortage of practical supporting evidence for the theory, except for studies such as those on the eye of the horseshoe crab (Limulus polephemus) and the abdominal ganglion of the gastropod mollusc (Aplysia californica).

Axoplasmic flow, in which enzymes and nutrients are moved towards the synapses was demonstrated more than 50 years ago by the simple process of constricting the axon and observing bulging on the soma side of the constriction. Reverse transmission from synapses to soma has also been shown to occur.

Transmission between cells occurs at the synaptic cleft. The transmissions may be electrical or chemical. Invertebrates mainly use electrical synapses, whilst in the higher vertebrates the transmission is mainly chemical.

The special structures of chemical synapses make use of molecules called neurotransmitters. These synapses are of special significance to the modelling of neural networks since they are the information transmitters.

Attempts to model neuron behaviour are complicated by the complexity of the chemical synapse. The many types differ in ion exchange mechanism, time constant, change of strength with activity, and environmental influence.

The two sides of the synaptic junction are known respectively as pre-synaptic and post-synaptic sides. The former relates to the input side that is driven by the action potential of its cell, whilst the latter refers to the driven cell.

Chemical synapses that have been meticulously studied by several researchers, according to Anderson, (1995), are those on the spinal motor neuron. They have been adopted as classical models for the mathematical approximations made for artificial neural networks.

1.3 Computer models

Due to the great complexity of the neuron, modelling may take place at many levels of understanding. Current computers would be unable to cope with the full model complexity of the biological neuron at our present state of knowledge. The degree of complexity of the neural network is based on the particular purpose it is intended to serve, since an artificial neural network is structured from several modelled neurons.

It may be relevant at this time to contrast conventional computer application with its use in neural networks.

1.3.1 Conventional computer behaviour

A basic conventional computer model due to von Neumann.

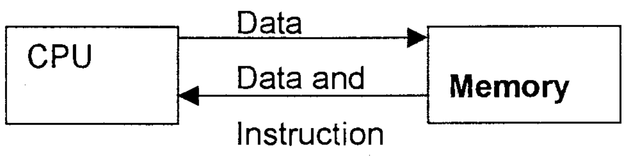


Figure 1.2

The activated computer performs the following sequence of events:

1. Receive an instruction from memory.
2. Retrieve all data required by the instruction from memory.
3. Execute the instruction.
4. Store the outcome in memory.
5. Go to step 1.

The algorithm is formulated to consist of a set of simple statements, which may be processed to the instructions that the central processor unit executes.

Strings of symbols that obey the rules of some formal system, may be interpreted conceptually. The dream for artificial intelligence was that all knowledge could be structured to permit symbol manipulation on a Von Neumann machine.

The following are some essential characteristics of the Von Neumann machine for comparison with those of neural networks.

The algorithm to solve the problem must be found, then the machine must be given detailed instructions on the exact sequence of steps (computer program) required to perform the algorithm.

Data is required in a precise format; noisy data upsets the machine operation. Hardware degradation occurs when a few key memory locations are disrupted.

Objects being processed, such as words or numbers, correspond to memory blocks in the machine hardware.

A conventional database would not have the capacity to account for the diversity of recognition that the human brain can make.

1.3.2 Artificial neural network behaviour

The main properties of an artificial neural network are:

a) Topology, organisation, number of layers and the manner of connections (structure).
b) Learning, the storage of information in the network (behaviour).
c) Recall, the way in which information is retrieved from the network (interpretation).

The neurons act independently, each one's output depending only on the inputs received from the connecting neurons.

The neuron does not need knowledge of the state of any other neuron from which there is no explicit connection. The large number of connections provide several redundancies in facilitating a distributed representation.

?
Simpson (1993) notes that neural networks are advantageous in three primary situations.

a) When a few decisions are required from large amounts of data such as pattern and speech processing.
b) Where non-linear mappings are to be automatically acquired such as loan evaluations and robot control.
c) Where a quick response is needed to a combinatorial optimization problem, as encountered in telecommunication and airline scheduling, a near optimal solution is usually provided by the neural network.

1.4 Artificial neural networks

A definition of a neural network (ANN) might be:
A set of simple processing elements interconnected in a manner that permits modelling, at a user-specified level, of the biological neuron.

Neural networks develop under the premise that, by modelling the physical architecture of the brain, within knowledge and capacity constraints, we may emulate its decision capability. This brain metaphor suggests that intelligence may emerge through the interconnection of several processing elements, each of which performs a simple computation, as a part of the total problem.

The processing capability of this connected network is dependent on the connection strengths (weights), the manner of connections, and the learning mode.

1.5 The role of artificial neural networks

The artificial neural network's role may be of greatest significance where the data to be processed is noisy, in the fields of:

1) Classification, in which data is categorized.
2) Association, in which a memorized pattern or object is retrieved based on recognition of part of the object.

3) Optimization, in which the solution found is thought to be the best.
4) Prediction, in which a future value or values are forecast.
5) Self-organizing, in which the best selection is found from input data through self-learning.

Activity	Network type	Potential Application
Classification	Learning vector quantization (LVQ) Counterpropagation Probabilistic neural networks (PNN)	Any best selection from scanned data that may be clustered.
Association	Hamming Hopfield Boltzmann Bi-directional associative memory (BAM)	Although a form of classification, these nets identify data that is close to the requirement but contains errors.
Optimisation	Hopfield Boltzmann	The travelling salesman problem.
Prediction	Back propagation General regression neural networks (GRNN) Group method data handling (GMDH)	For a given set of inputs the outputs may be grouped for a decision such as stock selection by predicted progress.
Self Organization	Kohonen networks LVQ	Winner selection from given data or clustering outputs

Activities and network types

Table 1.1

The above table lists the activity, possible network and potential application gleaned from numerous sources. It is worth noting that the feed forward backpropagation network has been used for most tasks.

1.6 Analysis of neural network limitations

Analytical approaches to the understanding of neural networks are inhibited by the variations of connecting pathways between neurons.

To progress in the understanding of artificial neural networks, it is clear that the natural behavioural basis of the nervous system must be simplified. Even the most sophisticated modern computers will not cope with the astronomical number of calculations required to behave as a biological network in a simple organism.

1.7 Neuron models

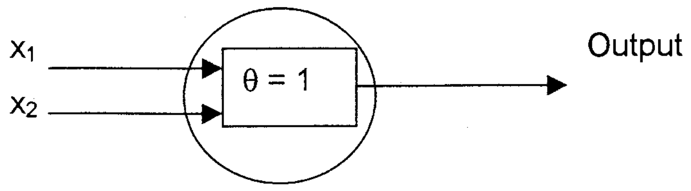
The processing elements of the neural network are the neurons, which are interconnected to form a neural network.

1.7.1 The two state neuron

The concept of the two-state neuron was introduced by McCulloch, and Pitts, (1943). This was an early attempt to model the physiological properties of the neuron and its connections, using neural computing elements, based on their assumptions of the biological neuron's behaviour.

After describing the all or none characteristics of quiescent or excited neurons in higher animals, as well as the synaptic delay between the receipt of a stimulus and the resultant output stimulus, Von Neumann (1945) explained: "Following W Pitts and W S McCulloch we ignore the more complex aspects of neuron functioning. It can easily be seen that these simplified neuron functions can be imitated by telegraph relays or vacuum tubes".

The McCulloch-Pitts neuron is of historical value, and not thought to be an effective model in the light of modern biological knowledge. The following diagram is of a two input, x_1, x_2 neuron, with threshold function $\theta = 1$.



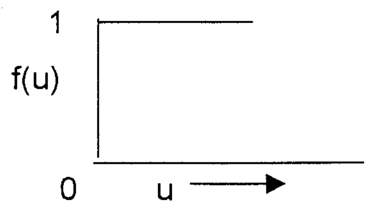
A simple McCulloch – Pitts neuron.

Figure 1.3

This artificial neuron was constructed under the false assumption that the brain functions as a logic and computational device. The inputs are the outputs of preceding cells. In the absence of inhibitory synapses the cell sums its inputs, if this sum exceeds a threshold value the neuron becomes active, else it remains in a passive state.

The McCulloch-Pitts neuron has been known as a threshold logic unit (TLU) where the information is processed as follows:

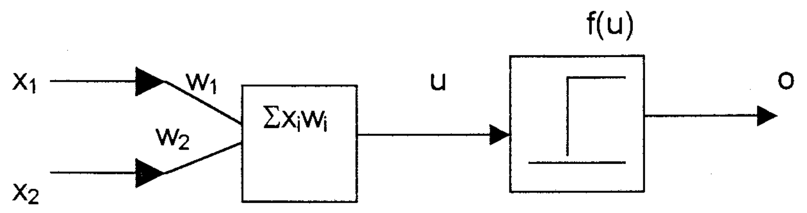
Each input signal represents the output of a preceding cell, this signal being multiplied by a weight representing the strength of the connection (synaptic strength). These weighted signals are now summed to produce a total processing element activation. If this activation exceeds a certain threshold the unit produces an output response.



Hard limiting threshold function

Figure 1.4

The original McCulloch-Pitts model of the neuron produced this binary response (hard limiting function), in which the output was 0 or 1.



A digital neuron model

Figure 1.5

Inputs x_i are the outputs of preceding cells,
Weights w_i are the strengths of synaptic connections,
$u = \Sigma x_i w_i$ is the sum of the weighted inputs,,
o = output of the threshold function f(u),
This single unit is capable of simulating the OR function, $S = x_1$ OR x_2.

x_1	x_2	S
0	0	0
1	0	1
0	1	1
1	1	1

A two input OR table.

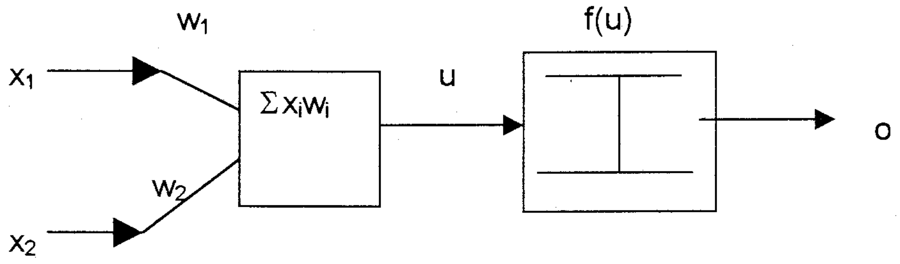
Table 1.2

This artificial neuron behaves as a digital-processing unit in that the inputs and outputs are of a binary nature. There is no mention of time, the unit being assumed to respond instantaneously to its input whereas the biological neuron integrates over time and space.
The logic function produced for a threshold value (bias) 2 is AND.

1.7.2 A non-linear threshold neuron

If in the McCulloch-Pitts neuron we replace the hard limiter function with a non-linear function, we may have a more general neuron that can have a discrete output (-1 or 1), or a continuous output varying between y_{min} and y_{max}.

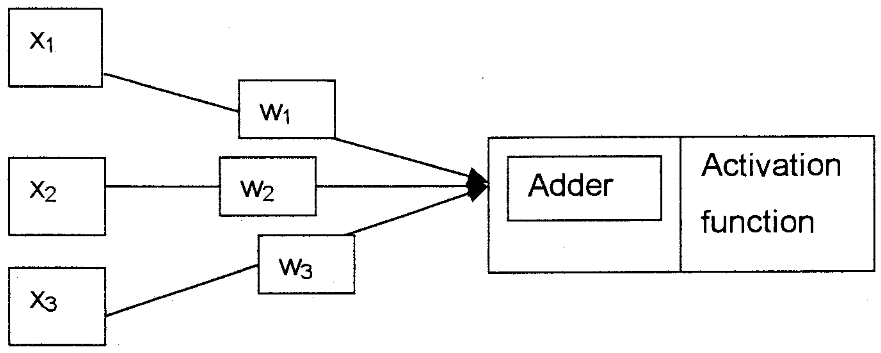
If the neuron fires, the output y_j is at a high (active) level, if the neuron does not fire it is said to be at the low, (quiescent) level.



A general two input neuron.

Figure 1.6

1.7.3 The generic connectionist neuron



The generic connectionist neuron.

Figure 1.7

For inputs x_i and connection weights w_i the initial artificial neuron function is to sum the products of inputs x_i and associated weights w_i.

In the biological neuron, activity at the synaptic cleft may be inhibitory, simulated by a negative weight or excitatory, modelled by a positive weight, this may be simulated in our artificial neurons.

Each input to a neuron has exactly one associated weight, so for the above model we may represent the inputs and weights as three-dimensional vectors **x** and **w**.

$$y = \sum_{i=1}^{3} x_i w_i = \square x_1 w_1 + x_2 w_2 + x_3 w_3 \square$$

We are sometimes required to **bias** (provide a threshold weight θ for) the summation effect, to achieve a particular result, then to simplify later computation we have:

$y = \Sigma x_i w_i + \theta$ which if we consider $\theta = x_4 w_4$ and $x_4 = 1$ whilst w_4 is the bias quantity then:

$$\sum_{l=1}^{4} x_i w_i = \square x_1 w_1 + x_2 w_2 + x_3 w_3 + \theta \square$$

In general:

$$\sum_{i=1}^{n+1} x_i w_i = \square x_1 w_1 + x_2 w_2 + \ldots\ldots + x_n w_n + \theta \square$$

where $\theta = x_{n+1} w_{n+1}$ and $x_{n+1} = 1$ whilst w_{n+1} is the bias quantity.

An appropriate activation function interprets the sum of these products to an output or no output state, dependent on the sum achieving a threshold level.

1.8 Activation functions

Activation levels of neurons can be discrete or continuous. Discrete examples which are used to classify inputs into two categories are the symmetric hard limiting function illustrated in (Figure 1.8), (or the hard limiting function 0 or 1).

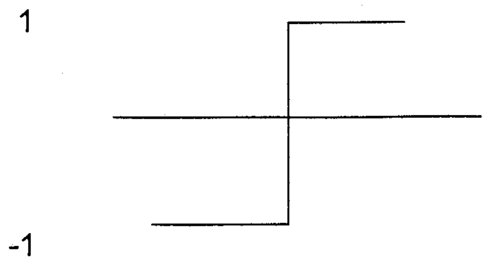


Figure 1.8

$$f(y) = \begin{cases} 1 \text{ for } y > 0 \\ \text{previous state for } y = 0 \\ -1 \text{ for } y < 0 \end{cases}$$

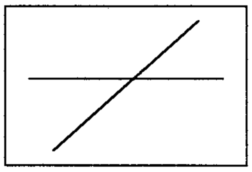
Where $y = \Sigma_{i=1} x_i w_i$.

1.8.1 Continuous examples

The common continuous examples are the linear, saturating linear, symmetrical linear, sigmoid and hyperbolic tangent, (tanh) functions.

(1) The linear transfer function has an output that is equal to its input. $f(y) = y$ (Figure 1.9a).

The saturating linear function (Figure 1.9b) $f(y) = \begin{cases} 0 \text{ for } y < 0 \\ y \text{ for } 0<y<1 \\ 1 \text{ for } y >1 \end{cases}$



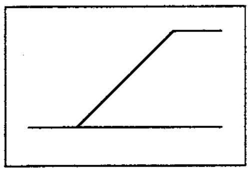


Figure 1.9a **Figure 1.9b**

(2) $f(y) = a + by$ (Figure 1.10)

The symmetrical linear function $f(y) = \begin{cases} -1 \text{ for } y < -1 \\ y \text{ for } -1<y<1 \\ 1 \text{ for } y >1 \end{cases}$

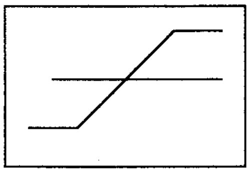
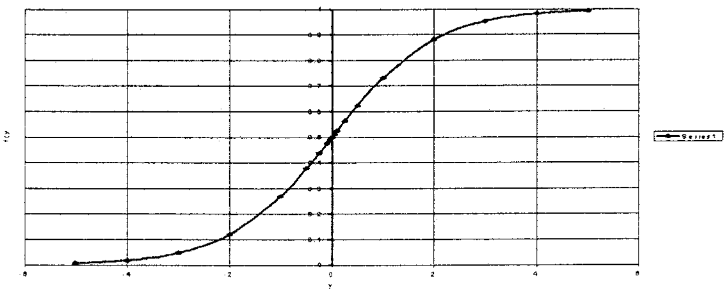


Figure 1.10

(3) A Sigmoid function typically $f(y) = 1/(1 + e^{-y})$ where the function $f(y)$ ranges from 0 to 1, as y ranges from $-\infty$ to ∞.(Figure 1.11)



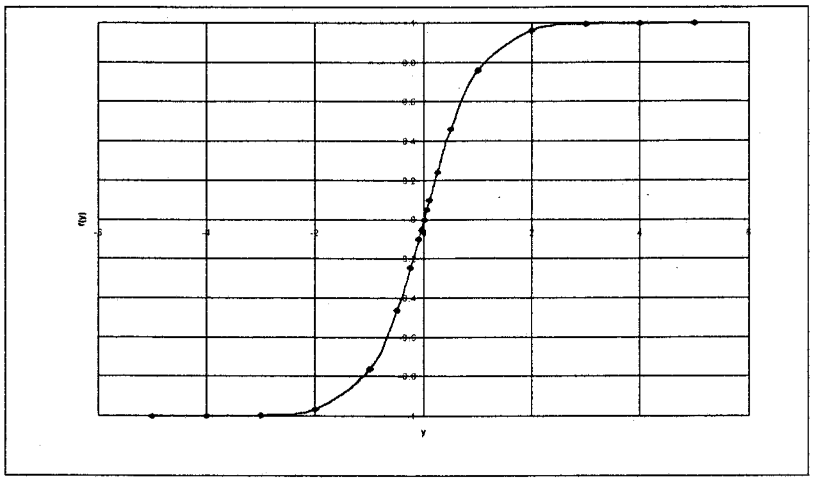
A sigmoid function

Figure 1.11

The function is valuable in backpropagation because of it's simple derivative.

(4) $$f(y) = \tanh(y) = \frac{1 - e^{-y}}{1 + e^{-y}}$$

where $f(y)$ ranges from -1 to 1, as y ranges from $-\infty$ to ∞. This is also of sigmoid form.



A tanh (y) function

Figure 1.12

1.9 Single neuron computation

Consider the two input neuron, Figure 1.4, the primary activity is to sum the weighted inputs. For given inputs from preceding cells, with weights representative of connection strengths, this summation results in a net stimulus in which the relative strength of individual weighted inputs is lost.

A secondary step converts the net stimulus to an activation level for the neuron, through the activation function. The tertiary step relates the relationship of the activation level to the threshold into an output. If the inputs are both 1 and the respective weights are 0.3 and 0.25 with a sigmoid activation function, with threshold 0.5 then the output will be 1 if the activation level equals or exceeds the threshold value, else output 0.

Step 1.	$1 \times 0.3 + 1 \times 0.25 = 0.55$
Step 2.	$O = 1/(1 + e^{-0.55}) = 0.6341$
Step 3	Since $0.6341 > 0.5$, output is 1.

1.10 Requirements for a practical circuit

To achieve a practical neural network some form of sensory detector will transmit a suitable signal(s), as an input to one or more artificial neurons. The chosen network will process the signals through it's structure to produce an output(s). When comparisons are made between output(s) and expected output(s), the circuit may be self-moderating to move closer to the desired result(s).

There are many different networks in use for a wide variety of practical applications varying through such diverse fields as drug detection sniffers to weather forecasting.

Recent documentation suggests an exciting prospect of neuron modulation by nitric oxide, to more closely simulate synaptic activity in the human brain. (Davidson, C.1998) (See chapter 9 on recent advances)

References

[1] Anderson, J. *An Introduction to Neural Networks* pp.7-14 (1995).

[2] Davidson, C. Gas on the Brain. *New Scientist.* 3rd October (1998).

[3] DARPA (Defence Advanced Research Projects Agency) (1996).

[4] Feigenbaum, E. A. *The art of artificial intelligence*: Themes and case studies of knowledge engineering. Proceedings of IJCAI-77 1014-1029 (1977).

[5] Gilling, D. & Brightwell, R. *The Human Brain* pp 17-18 (1982).

[6] Giarratano, J & Riley, G. *Expert Systems* pp 8-9 (1994).

[7] Li Min Fu, Neural Networks in Computer Intelligence. (1994).

[8] Maier, J. & Sherif, Y.S. Applications of fuzzy set theory. *IEE Transactions on Systems, Man and Cybernetics* (1985).

[9] Minsky, M. & Papert, S. *Perceptrons.* Cambridge Ma. MIT Press (1969).

[10] McCulloch, W. & Pitts, W. A logical calculus of the ideas imminent in nervous activity. *Bulletin of Mathematical Biophysics*, 5 pp.115-133.(1943).

[11] Newell, A., Shaw, J. C. & Simon, H. A. Empirical exploration with the logic theory machine: A case study in heuristics. *Computers and thought*, McGraw-Hill, New York.(1963).

[12] Newell, A, and Simon, H. A. Computer science as empirical enquiry: Symbols and search. *Communications of the ACM*, 19 (3), pp.113-126.(1976).

[13] Whitehead, A.N. & Russell, B. *Principia Mathematica* 2nd edition Cambridge University Press. (1950).

[14] Zadeh, L. A Fuzzy sets. *Information and Control* 8 pp.338-353 (1993).

Chapter 2.

NEURAL NETWORKS STRUCTURES

Summary.

This chapter describes in depth artificial neural network properties and the basic construction of networks. Instars and outstars are described as useful elements in many networks. The broad classifications of feedforward and feedback networks are illustrated and a simplified diagram method is introduced.

The concepts of training a network and its learning abilities are illustrated.

2.1 Artificial neural network properties

The networks are a limited model of the biological brain, with each artificial neuron attempting a small partial solution, for a sensory perception problem. The neural network is made up of many simple processing elements called artificial neurons. They are interconnected by direct links that act to perform parallel distributed processing, so that a given computational task may be solved.

Human beings are estimated to function with 10 billion or more neurons, differing in their structure dependent on the function they perform. The simulation of a small part of these functions is currently being attempted with at most a few thousand artificial neurons.

Artificial neural networks generally share the following features:

(1) Information processing and memory are distributed among the structure creating difficulty in dividing the structure into hardware and software. (Networks are trained not programmed).

(2) The high level of interconnection is in a form such that the state of an artificial neuron affects the potential of others to which it is connected in relation to the strengths (weights) of the connections.

(3) The connection weights, which are a comparable to the real neuron synaptic strength, are usually adaptive. Adaptation may occur anywhere within the connection, weight adjustment is by some algorithm, the result being distributed memory within the network.

(4) The artificial neurons, now called neurons, contain typically non-linear activation functions. The output of a particular neuron is a non-linear function of the input signals of the other fired neurons.

Neural networks often have elements with high variability that are not reliable. The highly redundant distributed structure renders the network insensitive to parameter variation (input noise) over a wide range, it is also relatively unaffected by individual element failure.

The total information of the network is distributed among the many artificial neurons, so that the losses of a few processing elements, or alteration to connections, cause relatively slight performance loss. Computer programmed systems are made unreliable by small amounts of memory failure, by contrast neural networks are robust.

Increasing degradation in a network depletes performance but does not cause sudden termination. The robust character gives an advantage to neural networks where complete failure is a critical hazard, whilst reduced performance is major in that it permits some recovery time. In the operation of nuclear plant, a changed pattern of behaviour, with regard to the many inputs of pressure, temperature, fuel element displacement, coolant flow and radiation levels, would still trigger a response in the absence of a few network elements.

In multi-layer networks the activation function must be non-linear, or its computational capability will be equivalent to that of a single layer network. I.e. it cannot learn non-linear mappings.

Suppose a linear activation function is used in a two layer network, input vector **x**, layer matrices $\mathbf{W}_a$ and $\mathbf{W}_b$, then the output would be $(\mathbf{x}\mathbf{W}_a)\mathbf{W}_b$. Since matrix multiplication is associative we may write $\mathbf{x}(\mathbf{W}_a\mathbf{W}_b)$ which is the equivalent of a single layer with weight matrix $\mathbf{W}_c = \mathbf{W}_a\mathbf{W}_b$.

2.2 Building the network

The fundamental tasks of network construction are:

a) The determination of the neural framework, and the pattern of connections.

b) The activation functions and range for the neurons.

c) The system dynamics in terms of initial weighting, momentum and learning rule.

2.2.1 The framework

The structure is by the number of layers, and the neurons per layer, a group of neurons within a layer is referred to in some commercial programmes as a slab. Typically there is an input layer, one or more hidden layers and an output layer.

The input layer neurons function as distributors to other neurons. The hidden layer provides the network non-linearities; the output layer presents the classified result, which may be used for comparison to an expected result. The network can be feedforward, or recurrent, with connections between neurons being symmetric or asymmetric.

Connection types:

Interlayer between neurons in different layers

Intralayer connections between neurons in the same layer

Self-connection from a neuron to itself

Supralayer connection is between neurons in non-adjacent layers.

Jump connections each layer is connected to all other layers.

2.2.2 Instars and Outstars

Instars and outstars were the Grossberg (1974) models for certain biological functions, they have been interconnected to form many complex networks.

Their description arises from the original form of representation.

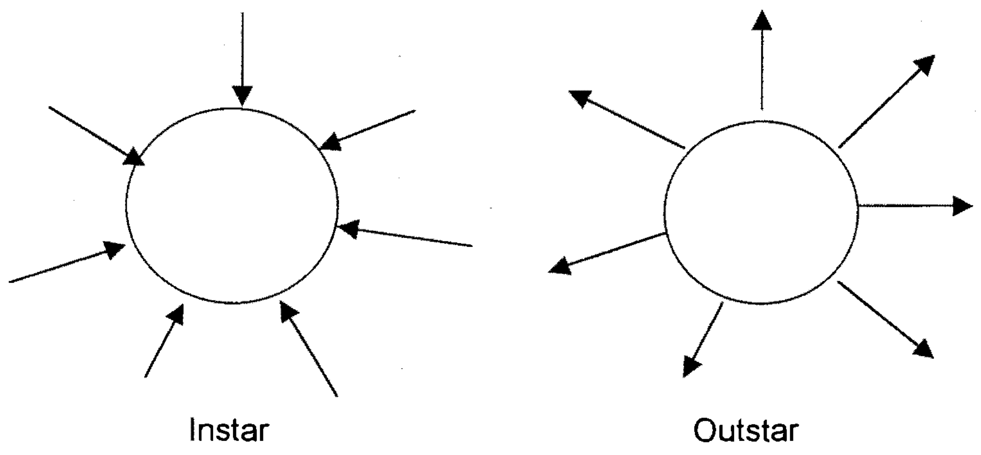


Figure 2.1

An instar is a pattern recognition device that is trained to respond to a specific input vector **x**, it does not respond to any other vector. In training, weights are adjusted to yield likeness to the input vector.

For inputs x_i and connecting weights w_i the output is $\Sigma x_i w_i$ from which the neuron output responds strongly to the input for which it was trained.

Wasserman (1989) describes the instar and outstar performance in the following manner.

Instar training modifies individual weights w_i at time t+1 from the state at time t according to the formula: $w_i (t + 1) = w_i (t) + \alpha(x_i - w_i (t))$, α is the training coefficient, starting at some small value about 0.1 and reducing as training progresses. The instar is triggered in response to a specific input pattern.

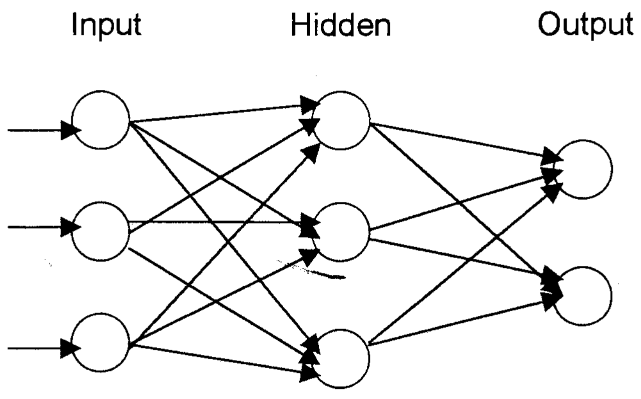
The outstar has a complementary function to the instar in that, when triggered, it transmits a desired excitation pattern to other neurons. Outstar training modifies individual weights w_i at time t+1 from the state at time t according to the formula: $w_i (t + 1) = w_i (t) + \beta(y_i - w_i (t))$, β is the training coefficient, starting at value near 1 and reducing to zero as training progresses.

2.3 Classification of Neural Networks

Artificial neuron connection sets are called architecture or circuit structures, the two major types being:

2.3.1 Feed forward (multi-layer) networks

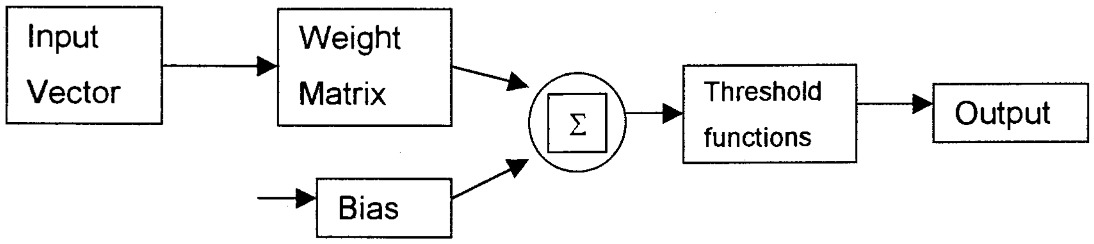
Networks in which each neuron receives inputs from other neurons or from external inputs, all connections are forward, there are no reverse (feedback) connections.



A feedforward network.

Figure 2.2

A simplified block diagram of the feedforward network shows the unidirectional nature of the layers.



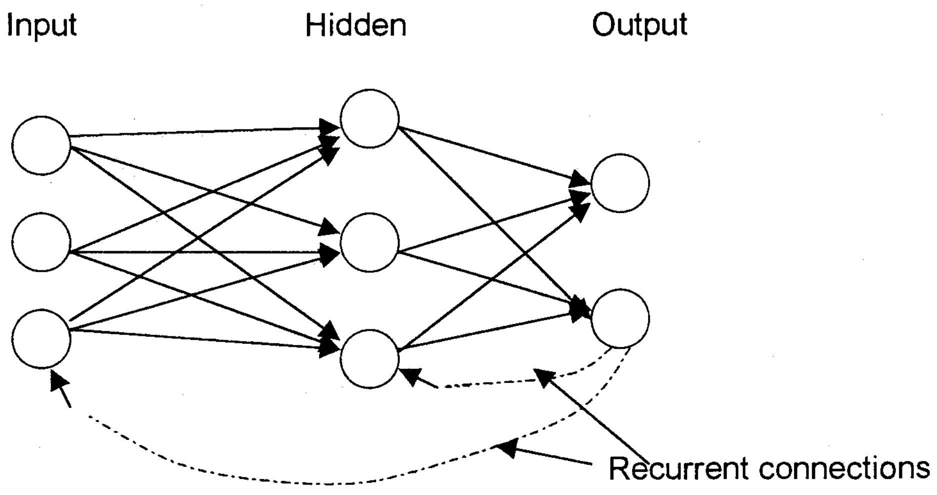
A feedforward network block diagram.

Figure 2.3

The input layer is a distributive vector, its weighted inputs matrix being summed and by application of the transfer functions resulting in the output pattern from the hidden layer. Where there are two or more hidden layers their inputs are the outputs of the preceding layer.

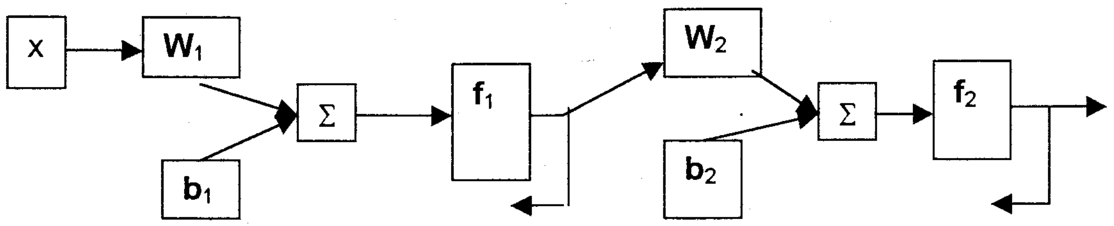
2.3.2 Feedback (recurrent) networks

Networks that have dynamic processing units in the form of integrators, or unit delays, that feedback to preceding summation units.



A recurrent (feedback) network.

Figure 2.4



A block diagram of a recurrent network.

Figure 2.5

2.3.3 The delay unit

For units in which time events occur in discrete steps, this unit may be used to delay the feedback signal that modifies the weights. The delay function causes the output to lag the input by one time unit.

a(t) = u(t-1)

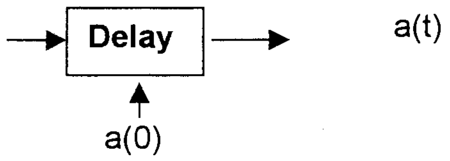
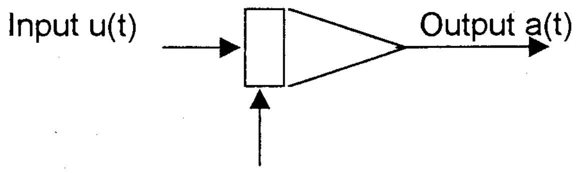


Figure 2.6

The outputs initial condition is t = 0 as indicated by the a(0) condition.

2.3.4 The integrator

The integrator in continuous time recurrent networks has an output that is dependent on its initial state, the input signal strength and the signal application time.



Initial state a(o), $a(t) = \int_0 u(t) + a(0)$.

Figure 2.7

2.4 Architecture selection

The problem specification, as stated by Hagan, Demuth, Beale, (1996), guides the network determination by:

Problem inputs = network inputs,

problem outputs = network outputs.

The transfer function selection may be partially guided by the output required.

2.5 Training and learning

Training is the application of a series of patterns at the input to the artificial neural network. Learning is the neural network ability to resolve those patterns. I.e. the internal activities that produce the end result of the training.

2.5.1 Learning

Adjustment of the interconnection weights between layers is the mechanism by which learning occurs. As a network result is produced it is compared with the expected result, and the connecting weights are adjusted towards the expected results. If learning is possible in finite time, a set of weights emerge that will produce acceptable responses for sample decisions or predictions.

The cycles of training are extremely important, since too little training does not permit the network to learn the patterns. Too much training results in the network learning noise or memorizing the training patterns with consequent inability to generalize well with new patterns.

2.5.2 Supervised learning

This requires a set of input patterns and the expected output patterns. Supervised networks build models, which classify patterns, make predictions, or make decisions. The results are in accord with other patterns of inputs and outputs they have "learned." They give the neural networks deduced answer based upon the variety of learned patterns.

In a supervised network, you show the network how to make predictions, classifications, or decisions by giving it a large number of correct classifications or predictions from which it can learn. The historical data of relative success will provide output patterns.

There are many supervised network types such as backpropagation, general regression neural network (GRNN), and probabilistic neural network (PNN).

2.5.3 Unsupervised learning

Sets of training patterns are provided, but no target is given, or forward path is provided, the network determines it's own output.

Unsupervised networks can classify a set of training patterns into a specified number of categories without being shown in advance how to categorize. The network does this by clustering patterns; it clusters them by their proximity in n dimensional space where n is the number of inputs. The user tells the network the maximum number of categories and it usually clusters the data into that number of categories. Occasionally the network may not be able to separate the patterns into that many distinct categories.

Kohonen networks are unsupervised. Kohonen networks in some proprietary networks (e.g. Neuroshell 2 Ward systems group, inc.) the winning neuron is set to 1, all others 0, or we may take the option of actual neuron values which will provide a neuron ranking. Pattern feed options are the may be in rotation, or random. The relative clustering distance may be Euclidean or normalized.

Euclidean distance, used in this analysis, is the square of the distance between pattern and weight vector per neuron. The winning neuron is selected as the one with minimum activation.

No type of network is guaranteed to always give an absolutely "correct" answer, especially if patterns are in some way incomplete or conflicting. Results should be evaluated in terms of the percentage of correct answers that result from the model. In this regard, the technology is similar to biological neural functioning after which it was designed, and differs significantly from all other conventional computer software.

Neural networks may not work at all with some applications. Some problems are well suited for the pattern recognition capabilities of a neural network and others are best solved with more traditional methods.

2.6 Layer designation

A single layer network is one in which there are only input and output layers, since the input vector is distributive only it is not counted in this study. (Some texts designate the inputs as a layer).

A network having an input vector, two hidden layers and an output layer, would be conventionally designated a three layer network.

Examples

Q1. What do you perceive as the potential difficulties in the simulation of human intelligence by the use of artificial neural networks?

A1 The first and most important factor is the current level of understanding of human intelligence. Simulation of a process depends primarily on the understanding of that process. Gaseous, chemical and electrical effects and their interfaces are still the subject of much study.

The second factor is the limitation of scale, in that the human brain is estimated to operate with 10^{11} neurons whilst the most sophisticated artificial neural networks operate with a few thousand neurons.

A third factor is the diversity of connections in the biological neuron, the connections being inhibitory or excitatory. Their strength being dependent on their usage.

Q2. When a network has been structured, explain the differences between supervised and unsupervised learning?

A2 For supervised learning the network is given a set of answers that are the expected results, A comparison is made between expectation and actual outputs. In a typical classification problem the network may be trained by being given a series of scripted letters to be correctly identified as the appropriate member of the alphabet. The success of the network is ability to classify correctly units not used for training.

The unsupervised network is not given expected results, its learning is not defined in terms of particular correct examples. In a Kohonen network the number of outputs describes the expected clusters. The network decides which is the correct group for a given vector of inputs.

Q3 The following input-weight pairs relate to the connectionist neuron shown in figure 1.7,

(0.1, -0.5); (0.6, -0.3); (0.4, 2.4)

If a bias of 0.5 is decided, determine the output of this neuron for activation function:

$y = f(\mathbf{u}) = 1/(1 + e^{-\mathbf{u}})$ where $\mathbf{u} = \Sigma x_i w_i + \text{bias}$

A3 $\mathbf{u} = (0.1)(-0.5) + (0.6)(-0.3) + (0.4)(2.4) + 0.5 = 1.23$

$y = f(\mathbf{u}) = 1/(1 + e^{-1.23}) = 0.77382.$

Q4 What would be the effect for question three if a bias weight of -0.7 was given?

$\mathbf{u} = (0.1)(-0.5) + (0.6)(-0.3) + (0.4)(2.4) - 0.7 = 0.03$

$y = f(\mathbf{u}) = 1/(1 + e^{-0.03}) = 0.5075.$

Q5 A neuron has inputs $(x_1, x_2) = (2, 3)$, with weights $(w_1, w_2)^T = (-4, 2)$ and bias 0.8, determine the output for:

a) A linear function

b) A symmetrical hard limit function

c) A log sigmoid function

d) A hyperbolic tangent function

A5 a) $\mathbf{u} = 2(-4) + 3(2) + 0.8 = -1.2$

$y = f(-1.2) = -1.2$

$y = f(-1.2) = -1$

$y = f(\mathbf{u}) = 1/(1 + e^{-(-1.2)}) = 0.2315.$

$y = f(\mathbf{u}) = (e^{(-1.2)} - e^{-(-1.2)})/(e^{(-1.2)} + e^{-(-1.2)}) = -0.8336.$

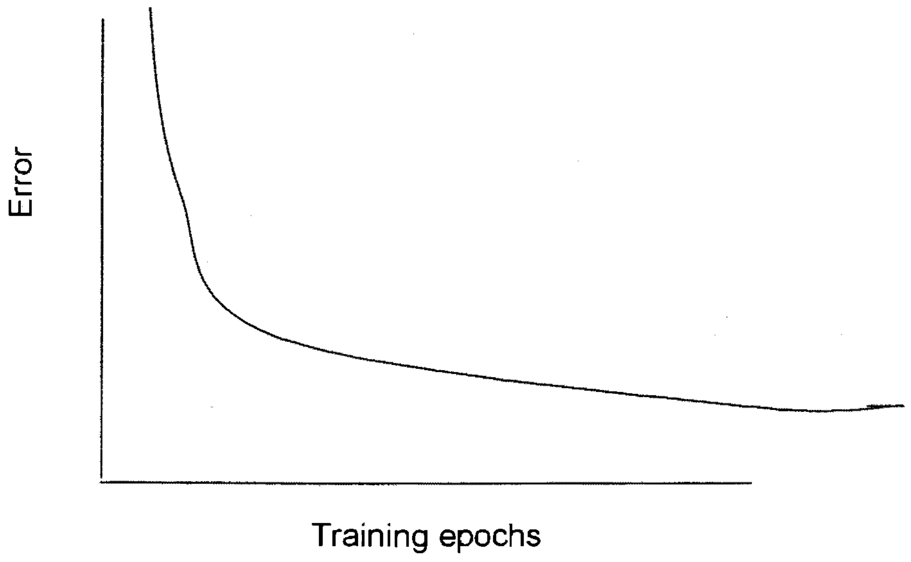
Q6 Herbert Simon (1983), made the following statement about machine learning:

"Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task, or tasks drawn from the same population, more efficiently and more effectively the next time".

Comment on the above statement in the context of artificial neural networks.

A6 From the input pattern set provided the artificial neural network learns, so that it may classify future unseen data. The learning usually takes place over several epochs. The network may be trained to a state in which the recognition is within an expected error. Over training may result in pattern memorizing that reduces the network capacity to classify previously unseen data, Herbert Simon has not introduced this aspect of learning.

In general a plot of error against training epochs in a supervised network shows the following behavioural pattern.



The following question is similar to a question in Hagan, Demuth and Beale (1996).

Q7 A single layer neural network has 5 inputs and 3 outputs, which are continuous in the range (0, 1). Answer the following questions about the network architecture:

a) How many neurons are required?

b) What are the dimensions of the weight matrix **W**?

c) What would be the output **Wx**?

d) What transfer function would you select?

e) Would a bias be required?

A7 a) Three neurons would be required, one per output.

The three neurons would each require one matrix row, whilst each input would need a matrix column. The weight matrix has three rows and 5 columns.

Wx is a three element vector.

A log sigmoid function is suggested because of the continuity requirement in the range 0 to 1.

Additional information would be required to determine the need for a bias.

References

[1] Anderson, J. *An introduction to neural networks* Massachusetts Institute of Technology, pp 214-230, (1995).

[2] Grossberg, S. Classical and instrumental learning by neural networks. *Progress in theoretical biology* vol 3, pp 51-141(1974).

[3] Hagan, M. T. Demuth, H. B. Beale, M *Neural network design* PWS publishing, Boston Ma. pp 2, 18-19, (1995).

[4] Wasserman, P.D. *Neural Computing Theory and Practice*, Van Nostrand Reinhold, pp 214-216, (1989).

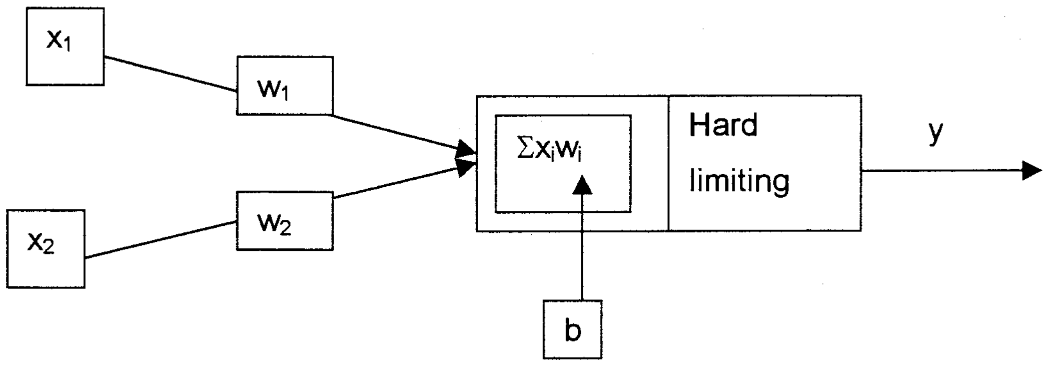
Chapter 3.

BASIC LINEAR NETWORKS

Summary

The basic building blocks of most modern networks, the perceptron and adaline are introduced. They are only capable of solving linearly separable classification problems. The perceptron has a further severe limitation, in that it's potential weights increase is unbounded. The perceptron, Hebbian and Widrow-Hoff learning rules associated with these networks are explained.

3.1 The single neuron perceptron and linear separation



A single neuron perceptron.

Figure 3.1

The output y for the simple network in Figure 3.1 is given by $y = f(\mathbf{w}^T\mathbf{x} + b)$ with the decision boundary being given by $\mathbf{w}^T\mathbf{x} + b = 0$. Above the boundary $y = 1$. On, or below the boundary $y = 0$.

For any point on the boundary line, the inner product of the input and weight vectors is constant, hence input vector **x** is orthogonal to the given weight vector w.

Vectors above the boundary line have an inner product $\mathbf{w}^T\mathbf{x} > b$, whilst those below the line have an inner product $\mathbf{w}^T \mathbf{x} < b$.
For a given weight vector **w** and input vector **x**, the bias may be calculated from $\mathbf{w}^T \mathbf{x} + b = 0$, using values from the decision boundary line.

Example

Suppose we have $\mathbf{x}^T = (x_1, x_2)$, $\mathbf{w}^T = (1, 1)$ and $b = -1$ then the decision boundary is $x_1 + x_2 - 1 = 0$ as shown in Figure 3.2.

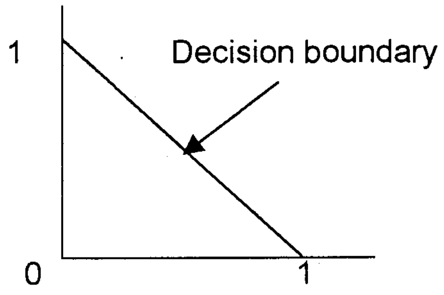


Figure 3.2

The region to the right of the line will give $y = 1$, to the left or on the line $y = 0$.

Neural networks built of perceptrons were the first to interest researchers in the possibilities of a true learning machine, and from it several complex learning networks have evolved.

Definition:

Linear separation occurs when a hyper-plane provides the decision surface that separates two pattern classes.

Suppose we have two pattern classes, Rosenblatt (1958) showed the ability of the perceptron to be able to learn to separate given patterns into the two classes. For linear separation it is possible to give rules for changing the connection strengths so that, in finite time, a set of weights are arrived at that will correctly classify the patterns.

3.1.1 Selection of weights and bias

Hagan, Demuth and Beale (1996) have given a method of deciding suitable weights and bias values for the single-neuron perceptron. For a chosen decision boundary they state that the weight vector will be orthogonal to the weight boundary. It must point to the class to have the value 1.

First the points are drawn on a suitable grid. A line is drawn that separates two classes. Class **a** is shown as light circles of value 1. Class **b** is shown as dark circles of value 0. The class point locations are shown in Figure 3.3a. The arrow representing the weight vector may be of any length. In Figure 3.3a weight vector $\mathbf{w}^T = (3, 2)$ is given as one of a set of choices.

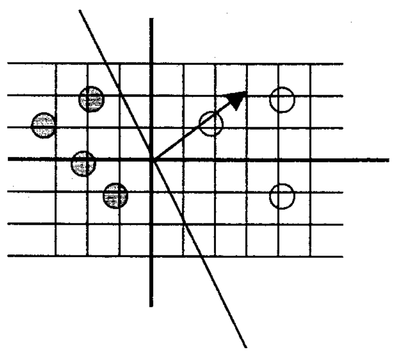


Figure 3.3a

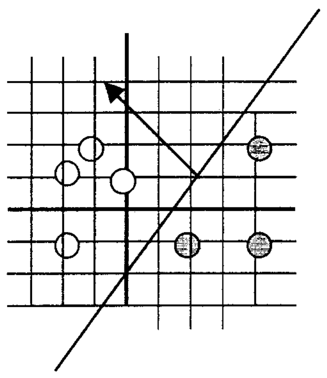


Figure 3.3b

To determine the bias we pick a point on the decision boundary. We then use the equation $\mathbf{w}^T\mathbf{x} + b = 0$, whence $b = -\mathbf{w}^T\mathbf{x}$. The values of **x** are given by the relationship of the point on the decision boundary to the weight vector start

In Figure 3.3a, $\mathbf{x} = \mathbf{0}$, then $b = -(3, 2)\begin{matrix}0\\0\end{matrix} = 0$

In Figure 3.3b, $\mathbf{x} = \begin{matrix}2\\1\end{matrix}$, then $b = (-3, 3)\begin{matrix}2\\1\end{matrix} = -3$

The results may be checked for each class point.

For dark point $\begin{pmatrix} -2 \\ 2 \end{pmatrix}$ and a hard limiting activation function,

$$f(\mathbf{w}^T\mathbf{x} + b) = f\left((3, 2)\begin{pmatrix} -2 \\ 2 \end{pmatrix} - 0\right) = f(-2) = 0,$$

For light point $\begin{pmatrix} 4 \\ 2 \end{pmatrix}$ and a hard limiting activation function,

$$f(\mathbf{w}^T\mathbf{x} + b) = f\left((3, 2)\begin{pmatrix} 4 \\ 2 \end{pmatrix} - 0\right) = f(16) = 1,$$

3.1.2 A training algorithm

We have two pattern classes **a** and **b**, and we have samples representative of each class. We shall submit these samples sequentially to the perceptron. As the supervisor we know the correct response for each pattern submitted.

If the system makes a correct response nothing is done. If it gives an incorrect response we modify the weights. If the output is 1 when it should be –1, we decrease the weights. If it is -1 when it should be 1, we increase the weights.

The concept of training a perceptron to classify patterns can be illustrated graphically by the example following the training algorithm.

This simple training algorithm (based on the work of Rosenblatt 1958) forms a new vector of connection weights from the old weights vector, based on the response to the training pattern, correctness of response and the input vector.

To start, all network weights are randomized and the bias b is set. When no starting values are given, it is recommended that the weights are set to small random values.

Procedure

1) Initialize connection weights.

2) Apply the input pattern (training vector **x**).

3) Compute the perceptron output $f(\mathbf{w}^T\mathbf{x} + b)$.

The response level of output O is given by:

$O = f(\mathbf{w}^T\mathbf{x} + b)$

where b = bias, f is the hard limiting function.

$$f(\mathbf{w}^T\mathbf{x} + b) = \begin{cases} 1 & \text{for } \mathbf{w}^T\mathbf{x} + b > 0 \\ 0 \text{ (can also be -1)} & \text{for } \mathbf{w}^T\mathbf{x} + b \leq 0 \end{cases}$$

4) Compare the actual output with the expected output

5) Compute new weights using the following table

$\mathbf{w}_{n+1} = \mathbf{w}_n + c\mathbf{x}$ where $\mathbf{w}_{n+1}$ is the new weight vector, $\mathbf{w}_n$ is the previous weight vector .

Actual Output	Expected output	C
1	1	0
1	-1	-1
-1	-1	0
-1	1	1

Weight constant table

Table 3

6) Repeat from (2) for the complete pattern set.

Example

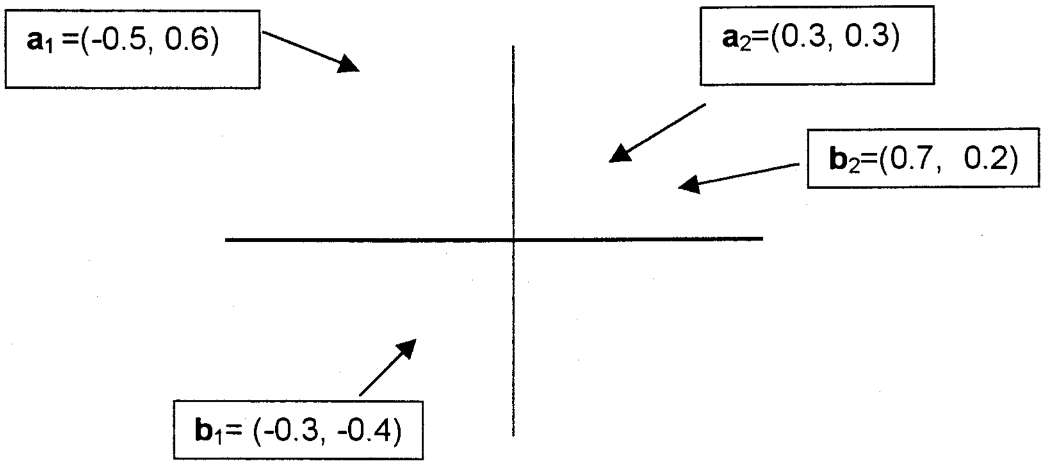
Suppose we have a series of points capable of representation in two classes **a** and **b**. We use the perceptron to classify them by assigning the value 1 to points in the **a** class and -1 to the points in the **b** class. The hard limiting function (-1 and 1) with bias b = 0 is used.

Initial weight vector:

w = (w_1, w_2) = (0.4, 0.2)

Training sets $\quad$ $\mathbf{a}_1$ = (-0.5, 0.6); $\mathbf{a}_2$ = (0.3, 0.3);

$\mathbf{b}_1$ = (-0.3, -0.4); $\mathbf{b}_2$ = (0.7, 0.2);



$\mathbf{a}_1$ =(-0.5, 0.6)
$\mathbf{a}_2$=(0.3, 0.3)
$\mathbf{b}_2$=(0.7, 0.2)
$\mathbf{b}_1$= (-0.3, -0.4)

Figure 3.4

The original points are plotted in the above diagram:

Apply the training pattern $\mathbf{a}_1$= (-0.5, 0.6)

$\mathbf{w}^T\mathbf{x} + b$ = (0.4)(-0.5) + (0.2)(0.6) + 0 = -0.08.

f(-0.08) = -1, expected +1.

$\mathbf{w}_{n+1} = \mathbf{w}_n$ + 1(input pattern vector).

Add corresponding components

w_1 = 0.4 + (-0.5) = -0.1

w_2 = 0.2 + 0.6 = 0.8.

Apply the next pattern

$\mathbf{b}_1$ = (-0.3, -0.4)

O = $\mathbf{w}^T\mathbf{x}$ = (-0.1)(-0.3) + (0.8)(-0.4) = -0.29

f(-0.29) = -1.

Since pattern b_1 response should be –1,

$\mathbf{w}_{n+1} = \mathbf{w}_n +$ 0(input pattern vector).

(w_1, w_2) unchanged.

Apply the next pattern $\mathbf{a}_2 = (0.3, 0.3)$

$\mathbf{w}^T\mathbf{x} - b = (-0.1)(0.3) + (0.8)(0.3) = 0.21$

$f(0.21) = +1$.

Since pattern a_2 response should be +1,

$\mathbf{w}_{n+1} = \mathbf{w}_n +$ 0(input pattern vector).

(w_1, w_2) unchanged.

Submit pattern $\mathbf{b}_2$ (0.7, 0.2)

$\mathbf{w}^T\mathbf{x} - b = (-0.1)(0.7) + (0.8)(0.2) = 0.09$.

$f(0.09) = +1$

Since pattern $\mathbf{b}_2$ response should be -1,

$\mathbf{w}_{n+1} = \mathbf{w}_n -$ 1(input pattern vector).

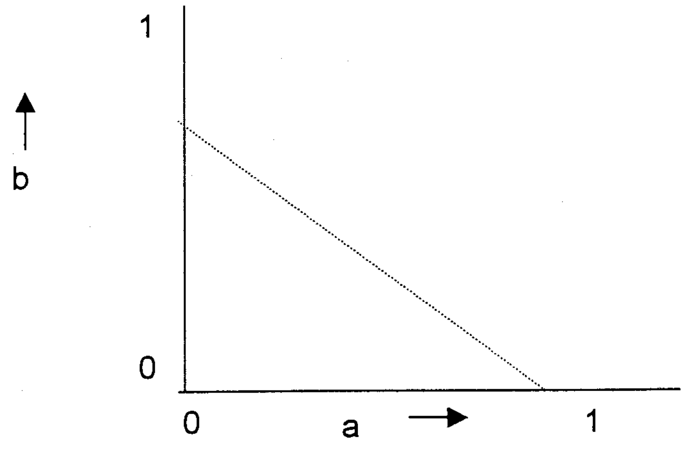
$w_1 = (-0.1) - (0.7) = -0.8$

$w_1 = 0.8 - 0.2 = 0.6$.

3.1.3 Logic functions by perceptron (linear separation).

Consider the logic OR function: S = a OR b.

a	B	S
0	0	0
1	0	1
0	1	1
1	1	1



Logic OR

Figure 3.5

The dotted line is one of many that could be used to separate 0 and 1.

A	B	C	D	E	F	G	H	I	
1	A perceptron program to perform the logic OR function								
2									
3	A hard limit function is used								
4									
5	Inputs	Connection weights		Product		Summation		Hard limit function	
6	1	0.5		B6*C6					
7	1	0.5		B7*C7		E6+E7		IF(G7<0.5,"0","1")	

A logic OR formula spreadsheet.

Table 3.2

The Excel spreadsheet gives the following output if formulae are preceded by an equals sign (=).

A	B	C	D	E	F	G	H	I	
1	A perceptron program to perform the logic OR function								
2									
3	A hard limit function is used								
4									
5	Inputs	Connection weights		Product		Summation		Hard limit function	
6	1	0.5		0.5					
7	1	0.5		0.5		1		1	

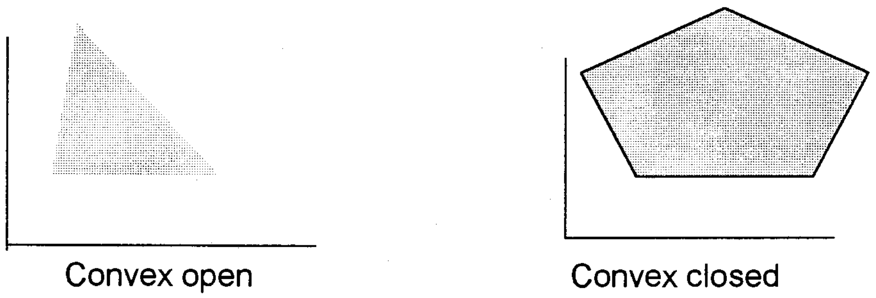
A logic OR calculations spreadsheet.

Table 3.3

For binary inputs in B6 and B7, I7 gives the OR result.

Simple spreadsheet perceptrons, like the above model, may be used to model the logic functions: OR, NOR, AND, NAND.

Each neuron may be used to produce a straight line. When two or more neurons form a network layer, the regions between the straight lines will be convex. Any convex region can be described by a (large enough)set of straight lines. By definition a convex region is one in which any two points within the region may be joined by a straight line that does not leave the region. The region may be open or closed, a closed region being completely bounded. (see Figure 3.5).



Convex regions

Figure 3.6

3.1.4 Delta rule training for linear separation

This is a generalization of the perceptron training algorithm of Rosenblatt (1958) that extends the technique to continuous inputs and outputs. In perceptron training, sets of inputs are provided, the weights being adjusted until the required output, or an output within the permitted error is achieved. The procedure is repeated for a set of inputs, which may be submitted sequentially or in a random manner.

The delta rule applied for single layer perceptron weight adjustment, is a simple rule for learning that will converge to a solution in a finite number of steps regardless of the initial choice of weights. The knowledge that a result will be achieved in a finite number of steps is offset by not knowing how many steps.

The delta algorithm for weight adjustment is:

$\Delta w_{ji} = \eta\delta x_i$.

Where Δw_{ji} is the change in the weight of x_i as used by neuron j,

η is the learning rate ($0 < \eta < 1$),

δ is the difference between target T and output value O and x_i is the input value.

Output is given by $O = f(\mathbf{w}^T\mathbf{x} + b)$ where b = bias for neuron j and f is the hard limiting function.

$$f = \begin{cases} 1 & O > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Weight adjustment $\mathbf{w}_{n+1} = \mathbf{w}_n + \eta\delta\mathbf{x}$.

Example

A single layer perceptron as shown in Figure 2.1 has a learning rate 0.35; inputs $x_1 = 1$; $x_2 = 1$; initially chosen weights are $w_1 = -4$ (inhibitory); $w_2 = 3$ (stimulatory) and the bias b = 1.

If the target value is T = 1 then $O = f((-4)(1) + (3)(1) - 1) = f(-2) = 0$,

$\delta = 1 - 0 = 1$,

$w_{3,1} = -4 + (0.35)(1)(1) = -3.65$,

$w_{3,2} = 3 + (0.35)(1)(1) = 3.35$.

2nd output with corrected weights,

$O = f((-3.65)(1) + (3.35)(1) - 1) = f(-1.3) = 0$,

$\delta = 1 - 0 = 1$,

$w_{3,1} = -3.65 + (0.35)(1)(1) = -3.3$,

$w_{3,2} = 3.35 + (0.35)(1)(1) = 3.7$.

3rd output with corrected weights,

$O = f((-3.3)(1) + (3.7)(1) - 1) = f(-0.6) = 0$,

$\delta = 1 - 0 = 1$,

$w_{3,1} = -3.3 + (0.35)(1)(1) = -2.95$,

$w_{3,2} = 3.7 + (0.35)(1)(1) = 4.05$.

4th output with corrected weights,

$O = f((-2.95)(1) + (4.05)(1) - 1) = f(0.1) = 1,$

$\delta = 1 - 1 = 0,$

$w_{3,1} = -2.95 + (0.35)(0)(1) = -2.95,$

$w_{3,2} = 4.05 + (0.35)(0)(1) = 4.05.$

The first spreadsheet abstract shows the weight updating formulae used in Excel. The preceding equals sign (=) has been omitted so that formulae may be shown.

The second spreadsheet shows the results that confirm the previous calculations.

	A	B	C	D	E	F	G	
1	Delta rule application							
2								
3	A hard limit function is used							
4								
5	Inputs	Weights	Product	Function	δ	α	Delta rule	
6	1	-4	A6*B6		1-D7	0.35	B6+E6*F6*A6	
7	1	3	A7*B7	IF((C8-1)>0,"1","0")			B7+E6*F6*A7	
8			SUM(C6:C7)					
9	1	G6	A9*B9		1-D10	0.35	B9+E9*F9*A9	
10	1	G7	A10*B10	IF((C11-1)>0,"1","0")			B10+E9*F9*A10	
11			SUM(C9:C10)					
12	1	G9	A12*B12		1-D13	0.35	B12+E12*F12*A12	
13	1	G10	A13*B13	IF((C14-1)>0,"1","0")			B13+E12*F12*A13	
14			SUM(C12:C13)					
15	1	G12	A15*B15		1-D16	0.35	B15+E15*F15*A15	
16	1	G13	A16*B16	IF((C17-1)>0,"1","0")			B16+E15*F15*A16	
			SUM(C15:C16)					

A formula spreadsheet for updating weights.

Table 3.4

	A	B	C	D	E	F	G
1	Delta rule application						
2							
3	A hard limit function is used						
4							
5	Inputs	Weights	Product	Function	δ	α	Delta rule
6	1	-4	-4		1	0.35	-3.65
7	1	3	3	0			3.35
8			-1				
9	1	-3.65	-3.65		1	0.35	-3.3
10	1	3.35	3.35	0			3.7
11			-0.3				
12	1	-3.3	-3.3		1	0.35	-2.95
13	1	3.7	3.7	0			4.05
14			0.4				
15	1	-2.95	-2.95		0	0.35	-2.95
16	1	4.05	4.05	1			4.05
			1.1				

Spreadsheet confirmation of example results.

Table 3.5

Example

This example is based on an idea from Wasserman (1989). A simple 5 row x 4-column grid of light transmitters may be constructed, so that the lit squares are seen as one of the numbers 0 to 9. We need to train a perceptron to recognise even number patterns by yielding output 1, odd patterns giving no output.

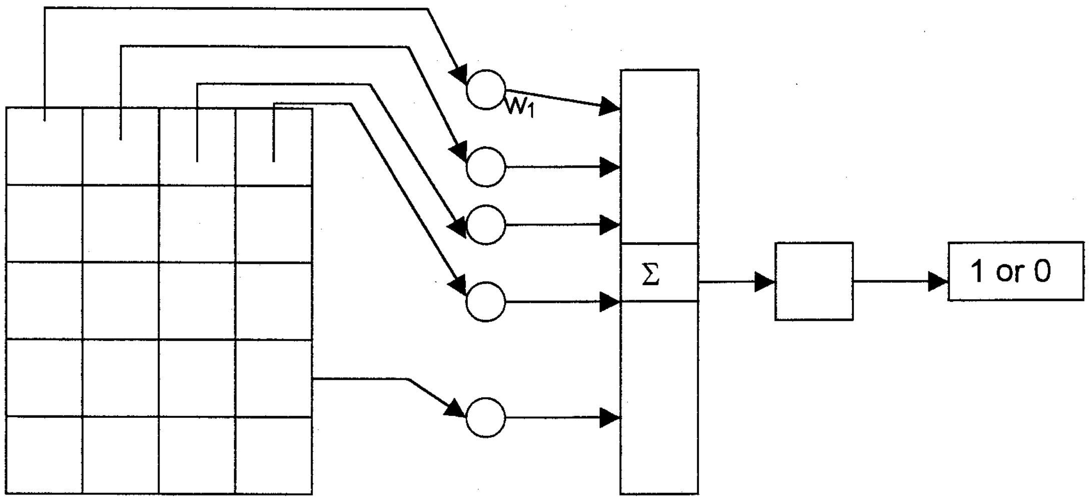
Picture the lit squares in the following diagram to see the figure 8.

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

A grid to display 8.

Table 3.6

The diagrammatic arrangement is shown below.



Number grid signals.

Figure 3.7

A lit square feeds the corresponding neuron an input of 1; otherwise, that neuron has input 0. Reading by row, left to right, the number 8 would be coded 0110 1001 0110 1001 0110 giving the perceptron response as 1.

With the appropriate summation of inputs multiplied by weights and bias added the correct response should be given. The correct weights are determined from a delta rule training algorithm.

A global training method over a complete number set would be used.

In practice the training process starts with all of the weights initialized to small random numbers. This protects the network against saturation by large values of the weights, and prevents certain other training pathologies. For example, if the desired performance is attained by unequal values of the weights and weights start at equal values, the network will not learn.

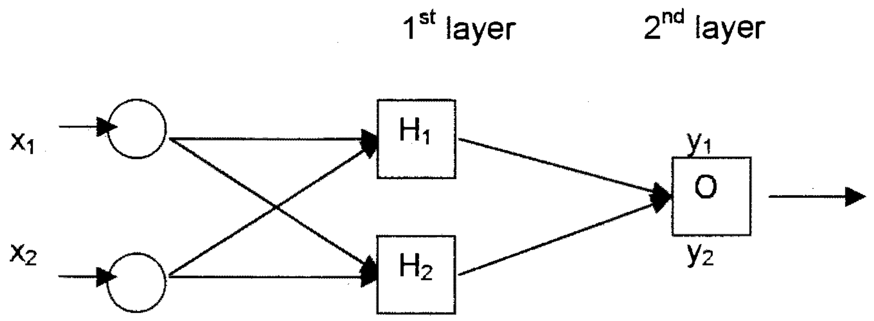
Note that a computer program would be more effective for this type of problem since it would be easy to store all binary patterns and their correct responses.

A similar technique could be used to represent the letters of the alphabet, or perhaps another set of ones and zeros that could be used to produce an output pattern. If one wished to train the network to recognize all the letters of the alphabet, 26 training vectors would be required.

3.1.5 "n" layer networks

The use of more than one layer in a network permits the linear separation of more than two classes. In an "n" layer network, selection of appropriate weights and activation functions can define convex regions.

We may examine a two-layer network in which two neurons in the first layer (hidden layer) feed a single neuron in the second layer (output layer), as shown in Figure 3.8.



A two-layer network.

Figure 3.8

Suppose for the output neuron (O) the connection weights from the hidden neurons (H_1, H_2) are (0.5, 0.5) with a hard limiting function. The output neuron bias activation is at 0.75. for output 1. The hidden neurons would need to satisfy;

$(y_1)(0.5) + (y_2)(0.5) > 0.75$, where ($y_1$, y_2) are the outputs of (H_1, H_2),

with $y_1 = f(x_1 w_{11} + x_2 w_{21}) + b_1$ and

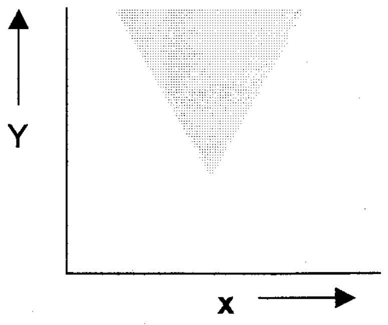
$y_2 = f(x_1 w_{12.} + x_2 w_{22}) + b_{2,}$

Therefore $f(x_1 w_{11} + x_2 w_{21} + b_1) = 1$ and $f(x_1 w_{12.} + x_2 w_{22} + b_2) = 1$ (Since f is a hard limiting function).

Therefore $x_1 w_{11} + x_2 w_{21} + b_1 > 0$ and $x_1 w_{12.} + x_2 w_{22} + b_2 > 0$ where b_1 and b_2 are biases for the hidden neurons.

Each neuron in the hidden layer will produce a straight line, the output neuron providing an AND function.

Suppose the first hidden neuron subdivided the xy plane to give 1 to the right of the line. Suppose the second hidden neuron gave 1 to the left of the line. (See Figure 3.8). A typical open convex region would be formed as the output neuron gives the logic AND function.



An open convex region.

Figure 3.9

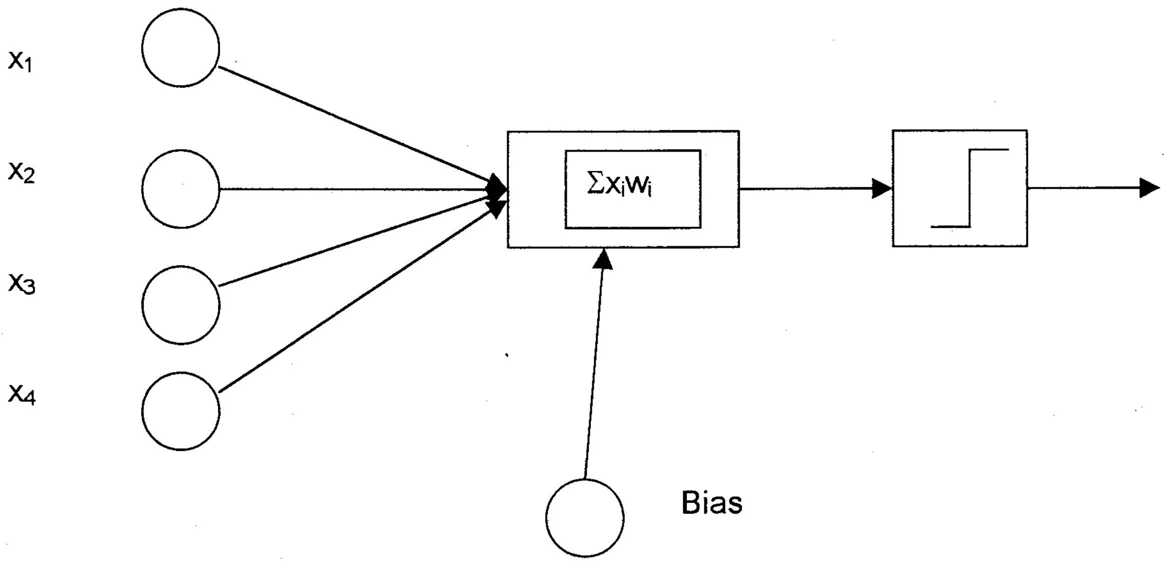
3.2 The adaline

The adaptive linear element (adaline) was developed by Widrow (1959). It has found practical applications in filters, telecommunications, and adaptive antennas (Widrow and Winter 1988).

To see the adaline in perspective it is useful to contrast it with its predecessor the perceptron. It differs from the perceptron in that the perceptron feedback is only for correctness of classification whilst the adaline continuously measures the error and progressively acts to minimize it.

The perceptron learns only from wrong answers, the adaline learns continuously. The perceptron changes its weights in response to the truth of its last classification. The learning process may take a long time to converge to a set of weights that classify correctly. A perceptron's set of weights that satisfy pattern requirements is rarely unique and may not be optimal. In contrast the adaline learns with speed and accuracy.

Initially the adaline was fed an input pattern, from which it produced a corresponding output classification. Early work assumed binary inputs 0 or 1 and output values of 1 or - 1.



An adaline neuron.

Figure 3.10

The adaline determines interconnecting weights that minimize the error between the output and the expected output. The method used to achieve this is the Widrow-Hoff rule, which is a least mean square (LMS) application. The rule differs from the delta rule in the manner of error calculation for updating the weights.

For a single neuron the inner product between the input vector and the related set of synaptic weights is computed. Computation is simplified if a bias weight is added, as explained in section 1.2.3. The output of the sum stage feeds the hard limiting activation function with output states, + 1 and - l, decision point 0. If at the sum stage the yield is positive output is 1 if not the output is - 1. The difference between the adaline and the perceptron is that the supervisor has access to the output of the sum stage and to the final classification.

The perceptron gives the output state, + 1 or -1, from which the perceptron learns only correctness of the final state. The internal state of the computing unit is not known. If the output of the summation stage was available, a supervisor could compute the error signal between the expected output and the computed output. This technique may yield a non-zero error, even when correct classification is achieved. Changes of weights may continue with correct classification, because the output of the sum stage, the inner product, is rarely exactly + 1 or - l.

3.2.1 Widrow-Hoff (LMS learning rule):

$\Delta w_{ji} = \alpha\delta x_i$ where $\delta_j = y_i - \mathbf{w}^T\mathbf{x}$

The rule application

For a given vector input $\mathbf{x}$ and required scalar output y_i, assume actual output y'_i.

Since $y'_i = \mathbf{w}^T\mathbf{x}$ where $\mathbf{w}$ is the weight vector, the LMS error minimises the square of the difference between the required and actual outputs.

$$e^2(\mathbf{w}) = \Sigma[(y_i - y'_i)^2] = \Sigma[(y_i - \mathbf{w}^T\mathbf{x})^2] = \Sigma[(y_i^2 - 2\, y_i\, \mathbf{w}^T\mathbf{x} + \mathbf{w}^T\mathbf{x}\, \mathbf{x}^T\mathbf{w})]$$
$$= \Sigma\, [y_i^2] - 2\, \Sigma\, [y_i\, \mathbf{w}^T\mathbf{x}] + \mathbf{w}^T\Sigma\, [\mathbf{x}\mathbf{x}^T]\, \mathbf{w}]$$

Set $\Sigma [y_i^2] = a$, $\Sigma [y_i \mathbf{x}] = \mathbf{b}$, and $\Sigma [\mathbf{x}\mathbf{x}^T] = \mathbf{C}$ (input correlation matrix).
Then $e^2 (\mathbf{w}) = a - 2\mathbf{b}\mathbf{w}^T + \mathbf{w}^T\mathbf{C}\mathbf{w}$

The LMS rule converges to a minimum error (it is a gradient descent rule), we may determine its optimal value at the turning point by differentiating the above expression and equating to zero.
$\delta e^2 (\mathbf{w}) / \delta \mathbf{w} = - 2\mathbf{b} + \mathbf{w}\mathbf{C} = 0$ then $\mathbf{w} = \mathbf{C}^{-1}\mathbf{b}$

For a given learning rate α the LMS weight-adjusting rule is:
$\Delta \mathbf{w} = \alpha(y_i - \mathbf{w}^T\mathbf{x})\mathbf{x}^T$

Examples

Q1) Give a brief description of the perceptron and the adaline neurons, indicating their essential differences.

A1) The key property of a single neuron perceptron is that it is capable of classifying an input vector into one of two categories. The category decision boundary is determined by the bias (b) in the equation **Wx** + b. Where **W** is the weight matrix and **x** the input vector. Since the boundary is linear the perceptron can only classify linearly separable patterns. Its activation function is the hard limiting function. In the perceptron feedback is only to determine if the network made the appropriate classification.

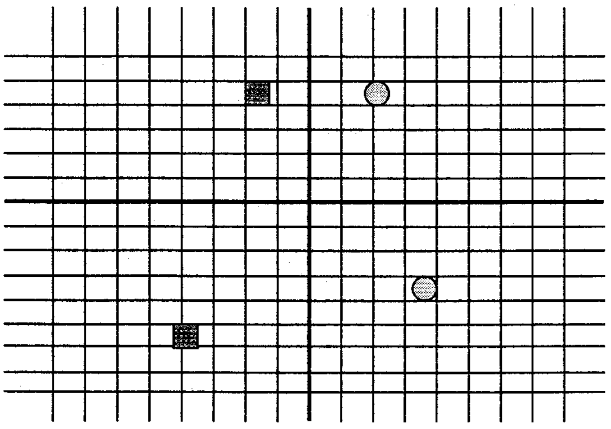
The adaline has the same basic structure as the perceptron, except that it uses the linear transfer function. It has the same linear separable constraint.

One major advantage is that of its LMS training algorithm. The perceptron training algorithm converges to a pattern classification solution although near boundary decisions may be sensitive to "noise". For the adaline the LMS training algorithm moves decision boundaries from the training patterns by minimizing the mean square error, reducing the risk of "noise" sensitivity.

Q2 Illustrate the following points on a graph. If a perceptron is used to separate the groups, **a** classified 1 and **b** classified -1, draw a suitable dividing line. For the chosen dividing line determine weights and a bias value.

Training sets ● $a_1 = (0.7, -0.6)$, $a_2 = (0.4, 0.8)$,
▨ $b_1 = (-0.3, 0.8)$, $b_2 = (-0.7, -1.2)$.

A2 Scale 1 square = 0.2 units



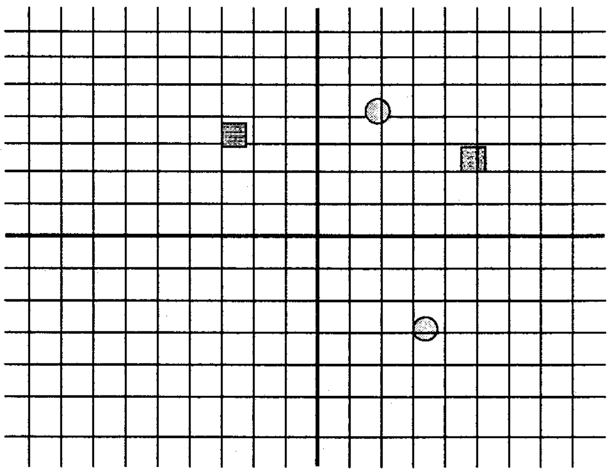
There are several lines that could be chosen with resulting bias selections

Q3 What would happen, using a single neuron perceptron, if for the same training sets **a**, the **b** sets were:

$b_1 = (-0.5, 0.6)$, $b_2 = (1, 0.2)$.

A3 A solution would not be possible with a single neuron perceptron since the points are not linearly separable.

Scale 1 square = 0.2 units



Q4 Do the final set of weights given satisfy all conditions for the worked example on pages 52 and 53?

A4 Final weight set: $\mathbf{w}^T = (-0.8, 0.6)$, with a hard limiting function (-1, 1) and bias (0).

Pattern $a_1 = (-0.5, 0.6)$:

$f(\mathbf{w}^T\mathbf{x} + b) = f((-0.8)(-0.5) + (0.6)(0.6) + 0) = f(0.76)$, $O = 1$ (correct).

Pattern $a_2 = (0.3, 0.3)$:

$f(\mathbf{w}^T\mathbf{x} + b) = f((-0.8)(0.3) + (0.6)(0.3) + 0) = f(-0.06)$, $O = -1$ (not correct).

$b_1 = (-0.3, -0.4)$:

$f(\mathbf{w}^T\mathbf{x} + b) = f((-0.8)(-0.3) + (0.6)(-0.4) + 0) = f(0)$, $O = -1$ (correct).

$b_2 = (0.7, 0.2)$:

$f(\mathbf{w}^T\mathbf{x} + b) = f((-0.8)(0.7) + (0.6)(0.2) + 0) = f(-0.44)$, $O = -1$ (correct).

Q5 Draw a simple spreadsheet program to compute the final weights that satisfy the required classification.

A4 The previous weight vector did not satisfy all patterns. A further epoch or epochs should produce the desired result.

At the second pattern submission:

Pattern $a_1 = (-0.5, 0.6)$:

$f(\mathbf{w}^T\mathbf{x} + b) = f((-0.8)(-0.5) + (0.6)(0.6) + 0) = f(0.76)$, $O = 1$ (correct).

Pattern $a_2 = (0.3, 0.3)$:

$f(\mathbf{w}^T\mathbf{x} + b) = f((-0.8)(0.3) + (0.6)(0.3) + 0) = f(-0.06)$, $O = -1$ (not correct).

Modified weights

$w_1 = -0.8 + 0.3 = -0.5$

$w_1 = 0.6 + 0.3 = 0.9$

$b_1 = (-0.3, -0.4)$:

$f(\mathbf{w}^T\mathbf{x} + b) = f((-0.5)(-0.3) + (0.9)(-0.4) + 0) = f(-0.21)$, $O = -1$ (correct).

$b_2 = (0.7, 0.2)$:

$f(\mathbf{w}^T\mathbf{x} + b) = f((-0.5)(0.7) + (0.9)(0.2) + 0) = f(-0.17)$, $O = -1$ (correct).

Note the new final weights (-0.5, 0.9) do yield correct results with all patterns.

A spreadsheet for pattern classification

Pattern	x1	x2	w1	w2	bias	#VALUE!	#VALUE!	Expected	#VALUE!
a1	-0.5	0.6	0.4	0.2	0	-0.08	-1	1	1
a2	0.3	0.3	-0.1	0.8	0	0.21	1	1	0
b1	-0.3	-0.4	-0.1	0.8	0	-0.29	-1	-1	0
b2	0.7	0.2	-0.1	0.8	0	0.09	1	-1	-1
a1	-0.5	0.6	-0.8	0.6	0	0.76	1	1	0
a2	0.3	0.3	-0.8	0.6	0	-0.06	-1	1	1
b1	-0.3	-0.4	-0.5	0.9	0	-0.21	-1	-1	0
b2	0.7	0.2	-0.5	0.9	0	-0.17	-1	-1	0

Formulae for pattern classification					Note spreadsheet values preceded by an equals sign				
Pattern	x1	x2	w1	w2	bias	B3*D3+C3*E3+F3	IF(G3>0,1,-1)	Expd	-0.5*(h3-I3)
a1	-0.5	0.6	0.4	0.2	0	-0.08	IF(G4>0,1,-1)	1	-0.5*(H4-I4)
a2	0.3	0.3	D4+J4*B4	E4+J4*C4	0	B5*D5+C5*E5+F5	IF(G5>0,1,-1)	1	-0.5*(H5-I5)
b1	-0.3	-0.4	D5+J5*B5	E5+J5*C5	0	B6*D6+C6*E6+F6	IF(G6>0,1,-1)	-1	-0.5*(H6-I6)
b2	0.7	0.2	D6+J6*B6	E6+J6*C6	0	B7*D7+C7*E7+F7	IF(G7>0,1,-1)	-1	-0.5*(H7-I7)
a1	-0.5	0.6	D7+J7*B7	E7+J7*C7	0	B8*D8+C8*E8+F8	IF(G8>0,1,-1)	1	-0.5*(H8-I8)
a2	0.3	0.3	D8+J8*B8	E8+J8*C8	0	B9*D9+C9*E9+F9	IF(G9>0,1,-1)	1	-0.5*(H9-I9)
b1	-0.3	-0.4	D9+J9*B9	E9+J9*C9	0	B10*D10+C10*E10+F10	IF(G10>0,1,-1)	-1	-0.5*(H10-I10)
b2	0.7	0.2	D10+J10*B10	E10+J10*C10	0	B11*D11+C11*E11+F11	IF(G11>0,1,-1)	-1	-0.5*(H11-I11)

Q6 A two layer perceptron has an input vector **x** = [2, 0, 3], the matrices of weights for the hidden layer and output layer respectively are:

$$\mathbf{w_h} = \begin{bmatrix} 0 & 1 & 7 \\ 3 & 5 & -2 \\ 2 & 3 & 0 \end{bmatrix} \qquad \mathbf{w_O} = \begin{bmatrix} 2 \\ 7 \\ 3 \end{bmatrix}$$

Assume that the hidden layer biases are $\mathbf{b}^T_{\mathbf{h}}$ =[1, 0, 1], the output layer bias is b_O = [1], Calculate the output.

A6 Let **h** and **o** be the hidden and output vectors respectively from the activation functions.

$\mathbf{h} = f(\mathbf{x}\mathbf{W_h} + \mathbf{b}_h)$ where $\mathbf{W_h}$ is the hidden layer weight matrix

$$[2, 0, 3] \begin{bmatrix} 0 & 1 & 7 \\ 3 & 5 & -2 \\ 2 & 3 & 0 \end{bmatrix} + [1, 0, 1] = [7, 11, 15]$$

Q7 Suggest a strategy for determination of the optimal number of hidden layer neurons.

A7 Plot the test performance against the number of hidden neurons and select the number offering the best performance.
For large networks, pre-selection may be made of a practical range of hidden neurons for the test.

Q8 Is the delta rule suitable for application to a multi-layer perceptron network?

A8 The delta rule may not be used if the error rate from a hidden neuron is not known, however in a backpropagation network this limitation may be overcome.

Q9 Can the spreadsheet approach of page 51 be used to model the exclusive OR (XOR) logic function?

A9 Because of the linear separation requirement with a single perceptron that spreadsheet approach may not be used.

Q10 With reference to the example on pages 58 and 59, what is the response for: 0110 1001 0010 1001 0110 ?

A10 The number is 3, with response 0.

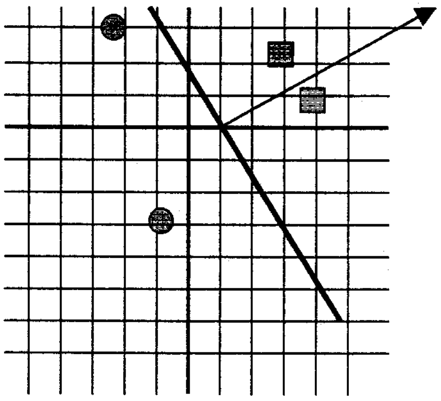
Q11 Draw a diagram for the following three cases, insert a suitable boundary line to separate the square and round points. Decide suitable weights and biases that satisfy the separation. Hard limit function (1, 0).

a) Square $(3, 2)^T$ and $(4, 1)^T$, round $(-2, 3)^T$ and $(-1, -3)^T$,

b) Square $(2, -1)^T$ and $(1, -3)^T$, round $(-1, 1)^T$ and $(-2, 1.5)^T$,

c) Square $(0, 2)^T$ and $(-1, 1)^T$, round $(0, -1)^T$ and $(1, -2)^T$.

A11 a)



The separation line is one of many that could be selected.

One set of weights are: $\mathbf{w}^T = (3, 2)$, then using $\mathbf{w}^T\mathbf{x} + b = 0$

For a chosen point on the decision boundary line $\mathbf{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $b = -\mathbf{w}^T\mathbf{x}$

Whence $b = -(3, 2)\begin{bmatrix} 1 \\ 1 \end{bmatrix} = -5$

Checking the solution for a) ▨ $(3, 2)^T$ and $(4, 1)^T$, ● $(-2, 3)^T$ and $(-1, -3)^T$,

$(3, 2)\begin{bmatrix} 3 \\ 2 \end{bmatrix} -5 = 8$ for the hard limiting function $f(8) = 1$ correct

$(3, 2)\begin{bmatrix} 4 \\ 1 \end{bmatrix} -5 = 9$ for the hard limiting function $f(9) = 1$ correct

$(3, 2)\begin{bmatrix} -2 \\ 3 \end{bmatrix} -5 = -5$ for the hard limiting function $f(-5) = 0$ correct

$$(3,\ 2)\begin{bmatrix} -1 \\ -3 \end{bmatrix} - 5 = -16 \text{ for the hard limiting function } f(-16) = 0 \text{ correct}$$

b) and c) similar approaches.

References

[1] Hagan M T, Demuth H B, and Beale M. *Neural network design*, PWS publishing, Boston Ma. pp 4, 20-21 (1996).

[2] LiMin Fu *Neural Networks in Computer Intelligence* McGraw Hill Series in computer science, pp 28-32.(1994).

[3] Rosenblatt, F *Psychological review* 65 (1958).

[4] Widrow, B. Adaptive sampled-data systems, a statistical theory of adaptation. *IRE WESCON Convention record*, part 4. New York: Institute of Radio Engineers, (1959).

[5] Widrow, B. & Winter, R Neural nets for Adaptive Filtering and Adaptive Pattern Recognition *Computer* (1988).

[6] Wasserman, P. D. *Neural computing* Van Nostrand Reinhold, New York, pp 48-49, (1989).

Chapter 4.

FEEDFORWARD NETWORK DESCRIPTIONS

Summary

Networks may be described by the direction of their connections, or by the functions they perform. The directions of connections use the terminology feedforward or recurrent, whilst the function is the description of neural network use, categories being:
classification, association, optimization and self learning.

This chapter uses practical examples of feedforward networks, in addition it introduces the categories which describe the functions of a neural network.

4.1 Directional classification

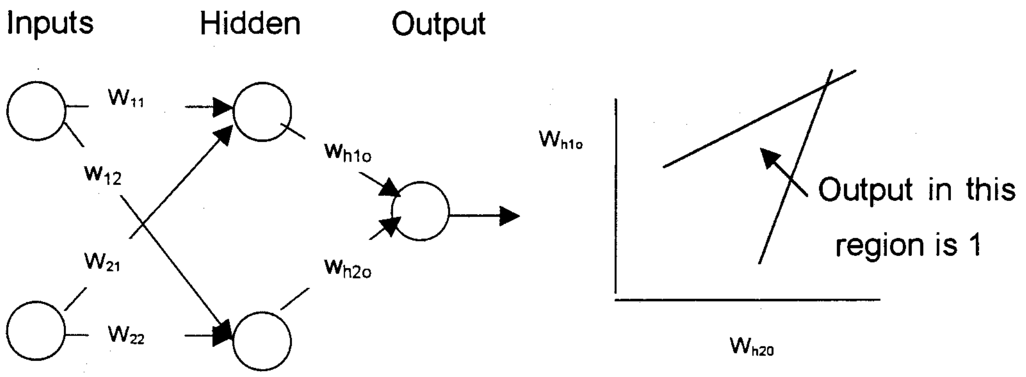
Feedforward neural networks process data from inputs to output(s). These network types have no feedback, as is the case with recurrent networks.

Commonly used feedforward networks include;
Multi-layer perceptron (MLP).
Kohonen.
Learning vector quantization (LVQ).
Cerebellar model articulation control (CMAC).
Group method data handling (GMDH). This is a special case, which although classified as feedforward, will be the subject of a separate chapter.

4.1.1 The multi-layer perceptron

The multi-layer perceptron is a feedforward neural network whose structure has input neurons, one or more hidden layers and an output layer. It may be used for non-linear classification problems.

As mentioned in chapter 3, each neuron in the first hidden layer is capable of producing a hyper-plane. Each neuron in the output layer is capable of combining hyper-planes to achieve convex decision regions. Depending on the bias, an output neuron can act as a logic OR , alternatively as a logic AND. A convex region is one in which points may be joined by straight lines that do not leave the region. Figure 4.1 shows a convex decision region produced by a two layer perceptron with two neurons in the hidden layer.



Production of convex regions

Figure 4.1

Following the ideas of weight and bias determination based on the work of Hagan, Demuth and Beale (1996) in chapter 3. We may choose suitable lines to divide three, or more, classes. A line is generated by each perceptron in the hidden layer. The resulting collection of lines separating the various classes.

Example

For four classes of input vectors:

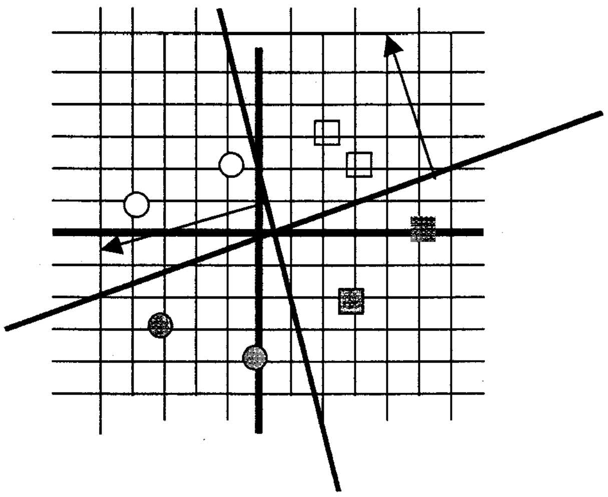
Class 1 □ $\mathbf{x}_1 = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$, $\mathbf{x}_2 = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$, Class 2 ▨ $\mathbf{x}_3 = \begin{pmatrix} 5 \\ 0 \end{pmatrix}$, $\mathbf{x}_4 = \begin{pmatrix} 3 \\ -2 \end{pmatrix}$,

Class 3 ● $\mathbf{x}_5 = \begin{pmatrix} 0 \\ -4 \end{pmatrix}$, $\mathbf{x}_6 = \begin{pmatrix} -3 \\ -3 \end{pmatrix}$, Class 4 ○ $\mathbf{x}_7 = \begin{pmatrix} -4 \\ 1 \end{pmatrix}$, $\mathbf{x}_8 = \begin{pmatrix} -1 \\ 2 \end{pmatrix}$,

Design the perceptron network to solve this problem.

An n neuron perceptron can categorize 2^n classes. The four classes will require a layer with 2 neurons.

Figure 4.2 gives the classes and their positions. Two decision lines are drawn that offer possible class separation. The first decision line separates the light and dark classes. The second decision line separates the circles and squares.



Class separation.

Figure 4.2

Weights are chosen that are orthogonal to the decision lines. The first decision line points in the direction of the light units which should produce output 1. The dark units produce output 0. The second decision line points in the direction of the circles to produce output 1. The squares produce 0.

The output layer neuron can offer selection of any set. I.e. $\mathbf{w}_1$ AND NOT $\mathbf{w}_2$ would select the light squares.

Selecting suitable weights, two possible sets are:

$$\mathbf{w}_1^T = (-1.5, 4) \quad \mathbf{w}_2^T = (-5, 1)$$

Selecting the biases:

$$b_1 = -(-1.5, 4)\begin{matrix} 5.5 \\ 2 \end{matrix} = 0.25 \quad , \qquad b_2 = -(-5, 1)\begin{matrix} -5 \\ -1.5 \end{matrix} = -23.5 \quad ,$$

For the light square $\mathbf{w}_1^T \mathbf{x}_1 + b_1 = (-1.5, 4)\begin{pmatrix}2\\3\end{pmatrix} + 0.25 = 9.25$,

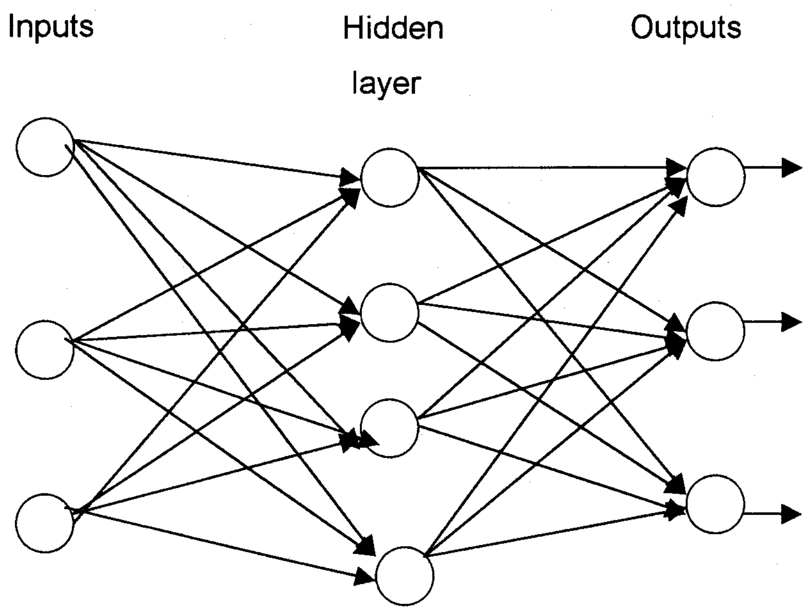
$f(9.25) = 1$.

For the dark circle $(-1.5, 4)\begin{pmatrix}0\\-4\end{pmatrix} + 0.25 = -15.75$,

$f(-15.75) = 0$.

This network has been designed with graphically defined decision boundaries that gave clear-cut decisions, in practical circuits this often not the case.

This single layer perceptron example uses a linear decision boundary to separate input vectors, there are however many problems in which categorisation is not linear.



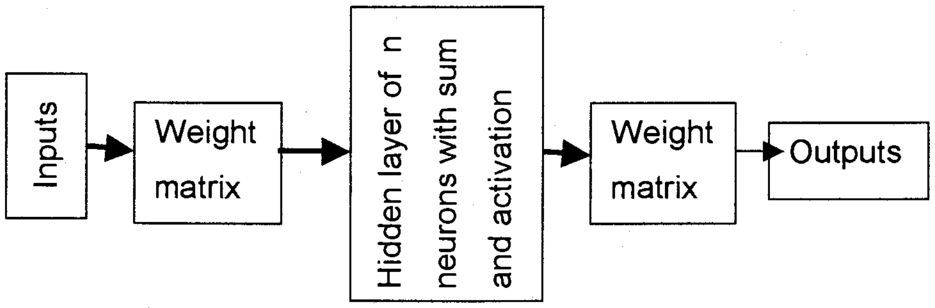
A perceptron network with a hidden layer.

Figure 4.3

In Figure 4.5 we have a two-layer representation, with four neurons in the hidden layer and three in the output layer. The input neurons merely distribute data and may be represented by a vector. Each hidden layer neuron will receive a vector of weights, the combined effect for the hidden layer will be a matrix in which the columns are the layer vectors.

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix}$$

Each input neuron is connected to all neurons of the two adjacent layers and to no other neurons. Note that connections within a layer or from higher to lower layers are not permitted.



A block diagram of a feedforward network.

Figure 4.4

Generally the multi-layer perceptron has a different number of neurons and different weights for different layers. Each neuron of the multi-layer perceptron is characterised by one output which may be fanned out to other neurons. There may be many inputs, which may be the outputs of the neurons in a preceding layer.

Persons building neural networks are faced with the problem of deciding the number of layers to be used and the number of neurons required in the hidden layers. Work by commercial packages suggests that most networks require 3 or less hidden layers. There is still work to be done on the determination of the number of neurons to be efficient within a hidden layer. An empirical approach would involve plotting test performance against the number of hidden units.

The following example shows the use of vectors and matrices in a two layer perceptron application.

Example:

LiMin Fu, (1994) provided the basis for the following work. Suppose we have a two layer perceptron with input vector $\mathbf{x}^T = (4, 2, 3)$, weight matrix $\mathbf{W_h}$ for the connections from inputs to hidden layers is:

$$\mathbf{W}_h = \begin{bmatrix} 3 & 2 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

and $\mathbf{w_o}$ for the connections from hidden layer to output layer is:

$$\begin{bmatrix} 3 \\ 1 \\ 5 \end{bmatrix}$$

$\mathbf{b_h} = (-1,0,-1)$, the thresholds for the hidden layer, $b_o = (-1)$ the threshold for the output layer.

Using $\mathbf{H} = f(\mathbf{x}\ \mathbf{W_h} + \mathbf{b_h})$ and $\mathbf{O} = f(\mathbf{H}\ \mathbf{w_o} + b_o)$ calculate the output

$\mathbf{H} = f[(17, 11, 9) + (-1, 0, -1)] = f(16, 11, 8) = [1, 1, 1]$
$\mathbf{O} = f(9 - 1) = f(8) = 1$

Multi-layer perceptrons are popular feedforward networks. Figure 4.5 shows a multi-layer perceptron with input distributing neurons and two layers: a hidden layer and an output layer. Neurons associated with inputs act as buffers for distributing the input signals x_i to neurons in the hidden layer. Each neuron j in the hidden layer sums up its input signals x_i after weighting them with the strengths of the respective connections and computes $y = f(\mathbf{w}^T \mathbf{x})$.

The function 'f' may be linear, sigmoid, hyperbolic, or some other activation function, the output of neurons in the output layer is computed in the same way.

LiMin Fu (1994) explains, adjustment is given by $w_{ji}(t+1) = w_{ji}(t) + \Delta w_{ji}$
The change of weight Δw_{ji} of a connection between neurons i and j is given as:
$\Delta w_{ji} = \eta \delta_j O_i$
where η is a parameter known as the learning rate, in the range 0 to 1, δ_j is the error gradient factor depending on whether neuron j is a hidden neuron or an output neuron and O_i is the output from a preceding neuron.

For hidden neurons $\delta_j = O_j(1 - O_i)\Sigma_k \delta_k w_{kj}$
where δ_k is the error gradient at neuron k with a connection from a hidden unit .

For output neurons $\delta_j = O_j(1 - O_i)(T_j - O_j)$ where T_j is the target output and O_j is the actual output at unit j .

When training a neural network, a set of patterns provide the input vectors. A training epoch is said to have been completed when all training patterns have been presented once to the multi-layer perceptron.
For all but the most trivial problems, several epochs are required for the multi-layer perceptron to be properly trained. A commonly adopted method to speed up the training uses a "momentum" term $(0 < \alpha < 1)$

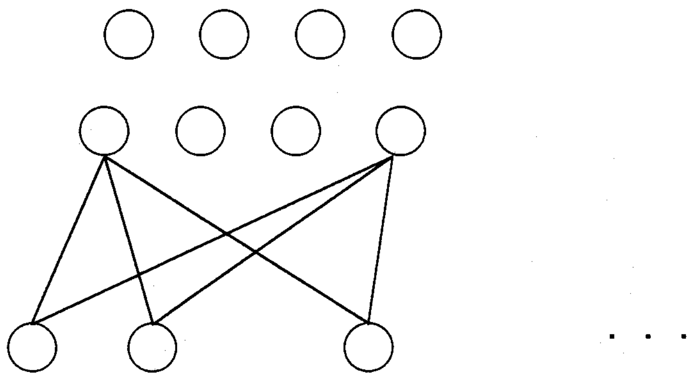
added to the weight adjusting equation that effectively lets the previous weight change influence the new weight change.

$$w_{ji}(t+1) = w_{ji}(t) + \Delta wji + \alpha[wji(t) + w_{ji}(t-1)].$$

4.1.2 The first Kohonen network

The first Kohonen (1982) network discussed in this thesis is a one layer architecture. The distributive input vector feeds an output layer which has one neuron for each possible output category. Neurons in the output layer are commonly arranged as a two dimensional array.

Each output neuron is connected to all input neurons and connection weights form the elements of the reference vector for a particular output neuron.



An early Kohonen network

Figure 4.5

The training patterns are presented to the input layer, then fed forward to the output layer and evaluated. A lone output neuron is the "winner". The network weights are adjusted during training. This process is repeated for all patterns for a number of epochs chosen in advance. Learning rate affects the network, and a commercial program (Ward systems, NeuroShell 2 (1996) reduces the learning rate as training progresses, causing progressively smaller weight changes, with improved network stability.

The network adjusts the weights for the neurons in proximity to the winning neuron. Initially the neighbourhood boundaries are fairly large (perhaps close to the number of categories) with increased learning the boundaries decrease, until during the last training events the neighbourhood is zero, resulting in changes to the winning neuron's weights only. When the learning rate is very small, and the clusters are well defined the clusters suffer only minor alterations.

4.1.3 The Kohonen self organising feature map

The Kohonen (1987) self-organising map network is an output grouping technique, implying that two or more outputs are required. There are sets of vectors that feed each output.

The network requires guidance to set parameters such as learning rate, initial weights, neighbourhood size, and number of epochs. The success of the network in classifying data is dependent upon how well these parameters are set.

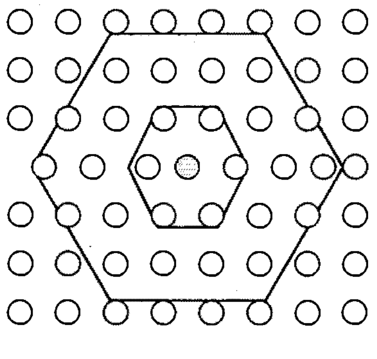

Hexagonal neighbourhoods.

Figure 4.6

The concept of neighbourhood size is illustrated Figure 4.6. Kohonen (1987) suggested that rectangular or hexagonal neighbourhoods were an aid to efficient network implementation.

In commercially available programs the neighbourhood size begins with a relatively high number, such as 90 percent of the number of neurons in the input layer. The learning rate should begin with a relatively high number such as 0.5. The number of epochs is dependent on the size of the problem to be solved.

A typical clustering problem might be the cost grouping for a parcel delivery service. The inputs could include trade volume, geographical mean distances, transport medium, cyclic trends, seasonality, weight, shape and content robustness, outputs may be cost classified into local, national, and international, each for low, medium, or high tariff.

Wasserman (1989) suggests the following algorithm for self-organizing map training:

Normalize all input vectors to unit length.

Apply an input vector **x**,

Calculate the distance D_j (in n dimensional space) between **x** and weight vectors $\mathbf{w}_j$ of each neuron. In Euclidean space this is calculated as:

$D_j = \sqrt{[\Sigma(x_i - w_{ij})^2]}$, where x_i is the ith component of input vector **x**

w_{ij} is the weight from input i to neuron j.

The neuron that has the output vector closest to **x** is declared the winner. This weight vector $\mathbf{w}_c$ becomes the centre of a group of weight vectors that lie within a distance D from $\mathbf{w}_c$.

Train this neighbourhood group of weight vectors with formula

$$\mathbf{w}_j (t + 1) = \mathbf{w}_j (t) + \alpha[\mathbf{x} - \mathbf{w}_j (t)],$$

Repeat the steps 1 to 4 through the cycle of input vectors.

As training progresses gradually reduce the values of D and α. Kohonen recommends that the number of training cycles should be at least 500 times the number of output neurons.

4.1.4 Learning vector quantization (LVQ)

Kohonen, Barna and Chrisley (1987) suggested a nearest neighbour classifier in which there are a fixed number of categories located in state space. The introduction of a new pattern in training, results in it's classification, which if correct, results in the weights of the nearest old pattern being adjusted to move closer to the new. In the event of a wrong classification the old pattern recedes.

Many problems work with noisy data and the distribution of possible examples may not be clearly bounded to a particular subspace. I e. the submitted pattern could belong to class A or B or.... or Z. Analytical approaches use probabilities to determine the most likely grouping. Like statistical modelling, vector quantization follows the probability density function of input patterns. In general, a coarser quantization is obtained in those areas where inputs are sparse.

Learning vector quantization is an application of competitive learning in which the input space is divided into disjoint subspaces in order that each input vector may be represented by its subspace label. I.e. for input vector $\mathbf{A}_1$ presented to the network with n_k the winning neuron the subspace label is $\mathbf{A}_1$. The output neuron fed from the subspace containing the winning neuron produces the output 1. It may be useful to have a programme that gives the output neuron values on a continuous scale, so that runners up can be examined.

These Kohonen networks are unsupervised. they can classify a set of training patterns into a specified number of categories without being shown in advance how to categorise. The network does this by clustering patterns. It clusters them by their proximity in N dimensional space where N is the number of inputs.

The user tells the network the maximum number of categories and it usually clusters the data into that number of categories. However, occasionally the network may not be able to separate the patterns into that many distinct categories.

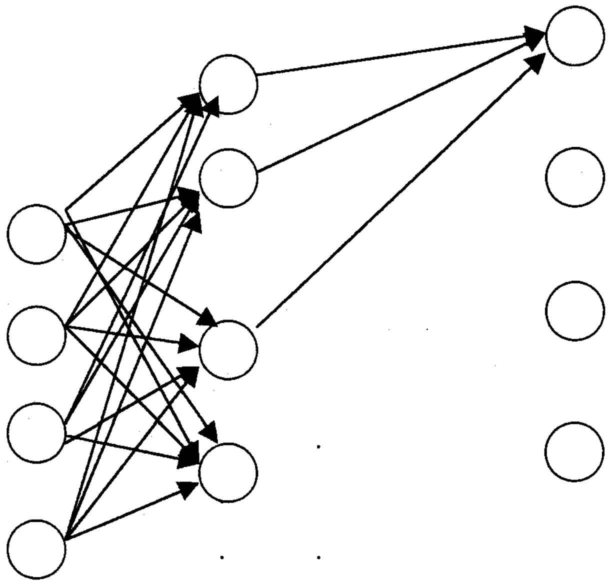
A practical example, which may be evaluated by this technique, is in classification of service providers for the internet industry as high, medium, or low quality. The inputs upon which classification could be based are:
access, personal contact, facilities offered, user protection, cost of service, supplied software, technical support times, service expansion and capability.

4.1.5 LVQ structure

The essentials of the LVQ structure are:

a) Full connection between inputs and the hidden layer.
b) Partial connection between the hidden and output layers each weighting being 1.
c) Each output is linked to a different cluster of hidden neurons.

A reference vector is formed by the weights of the input to hidden neuron connections, these being modified during the training of the network.



An LVQ structure.

Figure 4.7

The hidden (Kohonen neurons) and the output neurons have binary outputs. When an input pattern is supplied the Kohonen neuron with a reference vector closest to the input pattern is the winner with activation 1, all other Kohonen neurons have activation 0.

Pham, and Liu, (1995) offers the following simple LVQ training algorithm for output neuron determination

1) Initialise the weights of each reference vector.
2) Give an input training vector to the network.
3) Determine the Euclidean distance between the input training vector and the reference vector.
4) Update the weights of the reference vector having the least Euclidean value.
5) Recommence the cycle for a new input pattern.
6) Stop when all patterns have been cycled.

4.1.6 Cerebellar model articulation control (CMAC)

This is a class of neural networks that has been designed from physiological studies to simulate the brain as a controller. The cerebellum is the part of the brain primarily concerned with control of the motor functions, it is in some respects analogous to the perceptron.

An important idea put forward by Albus (1975) is that it may be possible to duplicate the functional properties of the brain without modelling the neuronal substrate.

The networks act as command sequence memories. They store and replay command sequences. Robotic movements are smoothed by combining multiple command sequences with different weights. The networks have been used for control of the motor function of robotic arms.

The network has been analysed as a supervised feed forward network with fuzzy memory, in which a series of sensory inputs, vectors $\mathbf{x}_i$, are mapped through a matrix of weights $\mathbf{W}$, to an output.

Albus (1984), in discussing the CMAC, noted that the manipulator control problem could be stated in simple terms, whilst its solution was complicated.

He points out that the simple action of picking up a glass involves the conscious actions of measuring the position of the hand relative to the glass and determining the vector direction to move the hand to the glass. At the sub conscious level we have movements of the shoulder joint, elbow joint and the muscular forces.

When translated to a manipulator problem, computations are required of individual joint rates, based on the trigonometric relationships between the structural members and the manipulator.

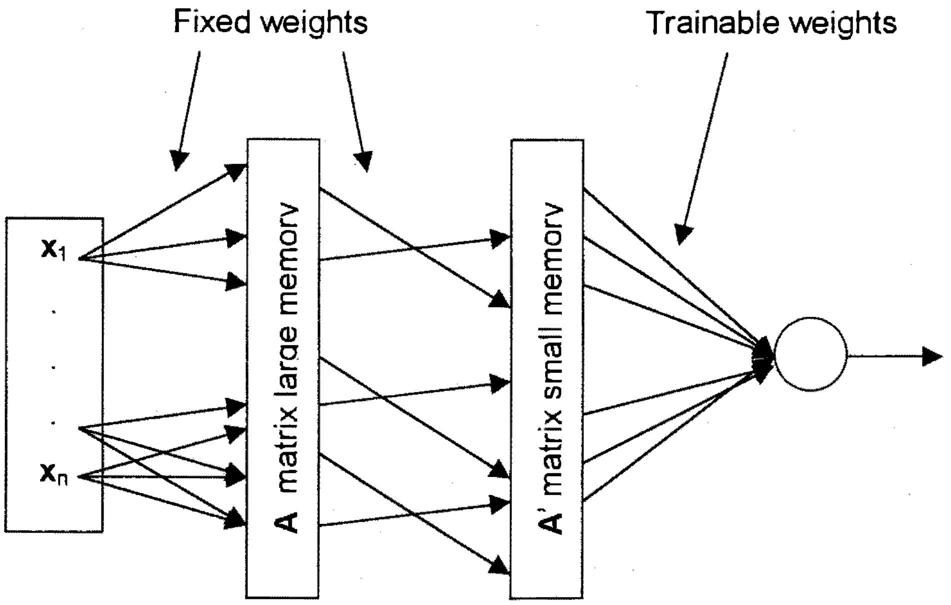
Albus (1979) explains, the simple mathematical models to cope with these states may be :

x = **J(θ)**θ where θ is a vector of individual joint velocities and **x** is a vector of end point velocities in a Cartesian co-ordinate system, **J(θ)** is the Jacobian matrix. For inversion of **J**, a given end point rate **x** may be used to determine joint velocities θ.

$$\theta = J^{-1}(\theta)x\,.$$

This simple model does not account for the complexity raised by gravitational forces, inertial loading and other aspects of a real life situation.

A CMAC module has $\mathbf{x}_i$ input vectors which are mapped to C locations in the large memory matrix **A** as shown in Figure 4.9. Pham and Lui (1995), state that the number of memory locations is large in comparison to the requirements of a typical control problem. This leads to the random mapping from the large memory to a smaller memory **A**'. There are still C locations in **A**' which are summed to produce the CMAC output.



A CMAC module.

Figure 4.8

Pham and Lui (1995) give some characteristics of the CMAC module:

1) They have input generalization in that similar inputs produce similar outputs.

2) A large CMAC can be trained and used in practical time.

3) The CMAC has a unique minimum due to the training rule used.

4) A wide variety of functions can be learned.

CMAC learning.

1) Assume f is a function to be learned and $u = f(.)$ is the required output.

2) Select the region where u is to be stored and compute the current value of $\underline{u'} = f(\mathbf{x}_i)$.

3) If $|u' - \underline{u}| < e$ (acceptable error), no action.

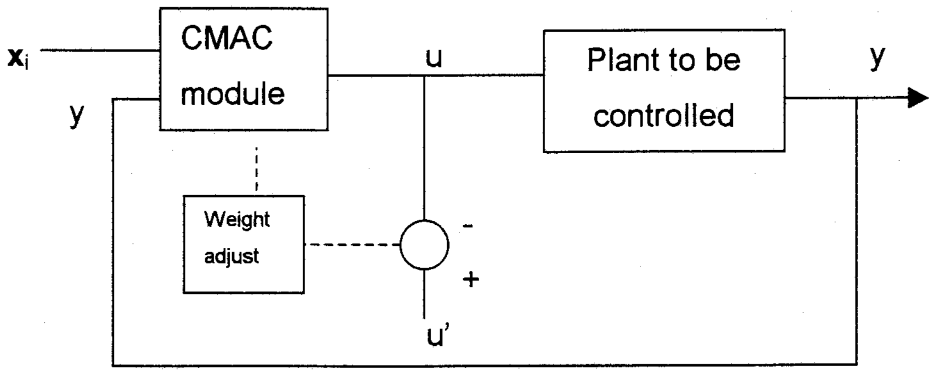
4) If $|u' - \underline{u}| > e$, then add to each weight that contributed to u the quantity $\alpha(u' - \underline{u})/|\mathbf{A}^*|$,

   where $|\mathbf{A}^*|$ = the number of weights from $\mathbf{A}'$ that have contributed to u' and α is the learning rate.

5) Repeat the above procedure until all inputs are processed.

Manipulator control is expected to be smooth and continuous so several similar responses are required for particular input space regions, widely separated regions are expected to be independent.

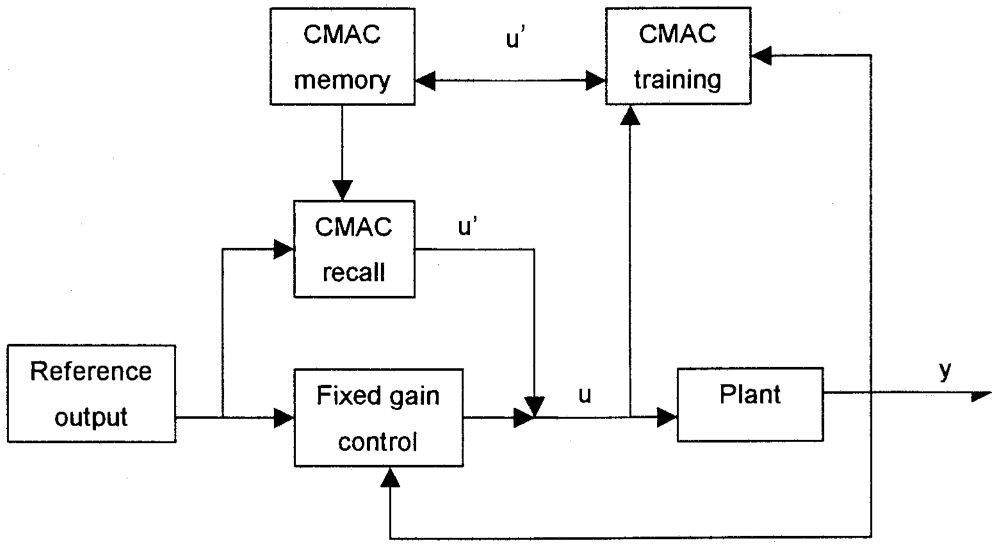
The perceptron ability to dichotomise over large distances and generalize close regions gives it potential in this type of control.



Direct CMAC based control

Figure 4.9

A more advanced control system due to Miller, Henes, Glanz, and Kraft, (1990) is illustrated in Figure 4.12.



Advanced CMAC control system
Figure 4.10

The reference output has the desired response for each control cycle which is fed to the fixed gain control and the CMAC recall. The CMAC recall sends a supplementary signal to that given by the fixed gain control, which is a conventional error feedback controller. At the completion of each control cycle a training procedure is used. The plant output from a previous control cycle is an input to the CMAC module. The difference between u and u' is used to compute the error. After training is complete the CMAC module will be the controller.

Examples

Q1 Give a possible instance for each of the following that may be cases for neural network applications:

Classification

Association

Optimization

Would any of the cases given be better suited to solution by an alternative approach?

A1 Classification:

This is placing a given input, or input set into one of several classes. One example would be estimating value of housing. Classification factors could include location, property size, features and security. An expert system might prove to be a better alternative, since judgement could be assisted by market histories.

Association:

The simplest view of association models relate a given pattern or part pattern to stored patterns (memory). The common networks for these tasks are Hopfield, Boltzmann and BSB. The number of neurons decide the storage capacity.

Anderson (1996) gives an example of the association of antibiotics and diseases. The neural network is a BSB (Brain State in Box) type and the network is taught to associate a particular antibiotic with a given input vector that is symptomatic of a disease. A suggestion is made that a subdivision of disease characteristics could lead to a more meaningful antibiotic formulation. In the direct associative state it would seem that a computer programme could be formulated to perform the association.

Optimization:

The classic traveling salesman problem is an example of optimization. Where a few nodes are involved the computations can be performed by a computer.

With increasing nodes to be visited the neural network may offer a solution that is acceptable although not necessarily optimal.

Q2 Suggest a clustering problem for a Kohonen self organizing map, indicate the potential inputs that may affect the clusters.

A2 To determine leisure activities for the population of a large city we may select the following output categories. Television viewing, computers, outdoor activities, cultural activities, social gatherings and cinema.

The inputs on which clustering would be made could include income, educational level, social status, state of health, environment, safety, available time and age.

Q3 Input weight vectors of a Kohonen network's hidden neurons are:

$\mathbf{w}_1^T = (0.451, 0.364, 0.726)$,

$\mathbf{w}_2^T = (0.380, 0.763, 0.525)$,

$\mathbf{w}_3^T = (0.852, 0.354, 0.427)$.

If the input vector $\mathbf{x} = \begin{bmatrix} 0.310 \\ 0.275 \\ 0.650 \end{bmatrix}$

a) determine the winning neuron,

b) update the winner's weight vector using $\Delta\mathbf{w}_{new} = \eta(\mathbf{x} - \mathbf{w}_{old})$ (assume a learning rate $\eta = 0.25$ and neighbourhood (**0**).

References

[1] Albus J.S. A new approach to manipulator control: cerebellar model articulation control (CMAC) *Trans. ASME, Journal of Dynamics systems, measurement and control,* 97, pp 220-227, (1975).

[2] Kohonen, T. *Self organization and associative memory* 2nd edition Berlin: Springer Verlag (1987).

[3] Kohonen, T. Self organized formation of topologically correct feature maps, *Biological Cybernetics, 43,* pp 59-69, (1982).

[4] Kohonen, T. Barna and Chrisley, (1987).

[5] Hagan M T, Demuth H B, & Beale M. *Neural network design,* PWS publishing, Boston Ma.17-6, (1996).

[6] LiMin, Fu. *Neural networks in computer intelligence,* McGraw Hill, pp. 27, 32, 50, (1994).

[6] Miller III, W. T., Henes, R.P. Glanz, F. H. and Kraft III, L.G. Real time dynamic control of an industrial manipulator using a neural network based learning controller. *IEEE Trans. On Robotics and Automation* 6(1) pp 1-9, (1990).

[7] Pham, D, T and Liu, X, Neural Networks for Identification Prediction and Control, Springer. pp 67-79 (1995).

Chapter 5.

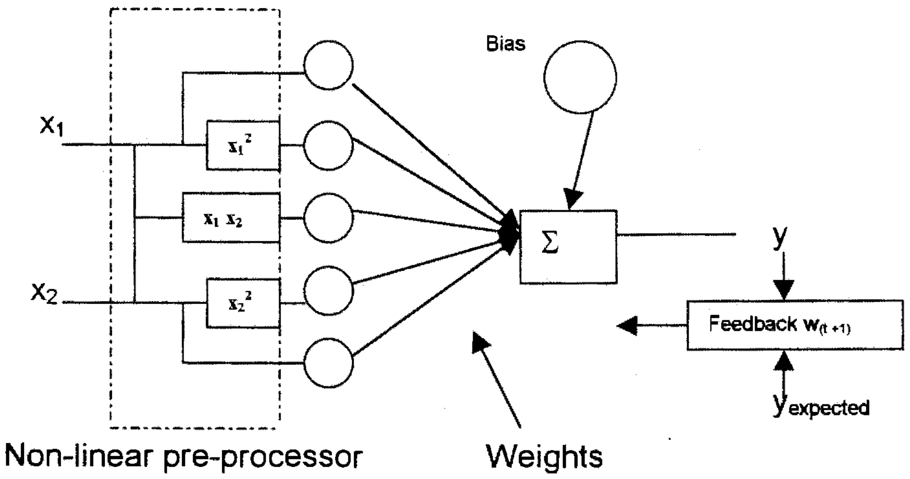
GROUP METHOD DATA HANDLING

Summary

This chapter describes a neural network type that may be considered to offer an alternative to multiple regression methods in certain cases. A major benefit of this network may be the ability to detect the less important factors in the generated polynomial.

5.1 Group Method Data Handling (GMDH)

Group Method of Data Handling (GMDH) networks are sometimes referred to as polynomial nets. They are classified as a feedforward network although they are not like regular feedforward networks. The invention of Ivakhnenko (1971), the network uses adaline neurons with non-linear pre-processors, that are called N-adaline neurons.



An N-Adaline neuron

Figure 5.1

Each N-adaline neuron in a typical GMDH network usually has two inputs x_1 and x_2 from which it produces an output y. The output of each neuron is a quadratic combination of the two inputs, together with the connection weights.

$$y = \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{q}^T\mathbf{x} + c,$$

Where $\mathbf{x}^T = (x_1 + x_2)$, $\mathbf{Q} = \begin{bmatrix} w_2 & \frac{1}{2}w_4 \\ \frac{1}{2}w_4 & w_5 \end{bmatrix}$, $\mathbf{q} = \begin{bmatrix} w_1 \\ w_3 \end{bmatrix}$ and $c = w_0$,

The number of neurons used as inputs is dependent on the number of external inputs available. Each pair of external inputs, requires one neuron.

GMDH networks have structures that grows in training. A network tries to build a function (called a polynomial model) that would behave in such a way that the predicted value of the output would be as close as possible to the actual value of output. The network may be used for modelling or prediction.

Training proceeds with presenting an input pattern as an input vector. The weights of each neuron are adjusted according to a suitable learning algorithm, such as the Widrow-Hoff learning rule.

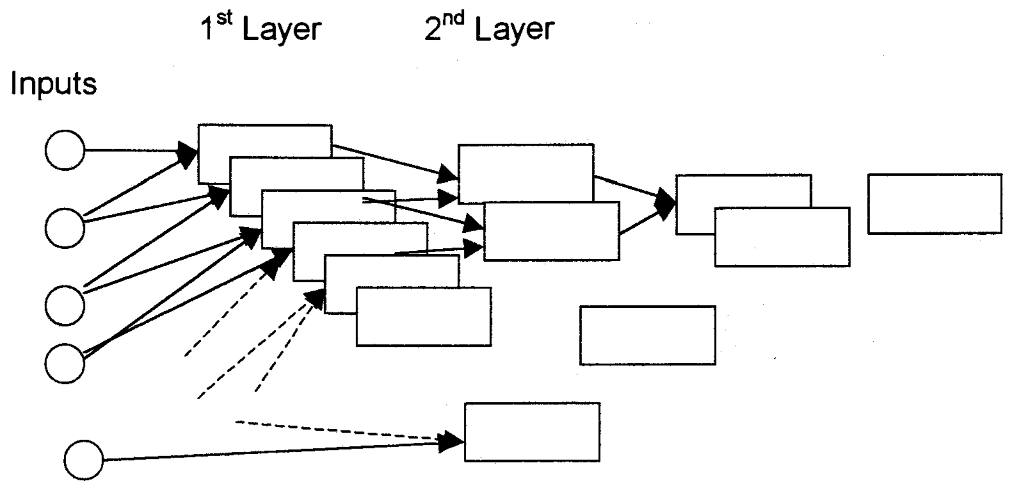
Pham and Liu quote the Widrow-Hoff modified training rule, as modified by Widrow and Lehr (1990):

$$\mathbf{w}_{(t+1)} = \mathbf{w}_{(t)} + \alpha\mathbf{x}_t/|\mathbf{x}_t|^2(y_{\text{expected}} - \mathbf{w}_t^T\ \mathbf{x}_t).$$

The application of this rule causes **w** to be modified to reduce the difference between the actual and desired outputs.

Training a GMDH network consists of configuring the network starting with the input vectors and adjusting the weights of each neuron. During training new layers are formed (Figure 5.2). With each new layer a number of new neurons are formed from preceding neurons that perform well.

Neurons in the preceding layer that have not perform well are discarded. The new layer formation continues until the accuracy of the mapping achieved with the network is optimal. I.e. further layers reduce accuracy.



A network of N-adalines for a required polynomial output.

Figure 5.2

Every neuron in the first batch is targeted to a desired output and is expected to achieve it. When the batch mean square errors (BMSE), summed over the desired outputs of the training set of a neuron, reaches a minimum its weight modification process is stopped. From the now stopped layer, the selection data set chooses trained neurons for the next stage.

The neurons with squared error below a certain threshold are the post selection choice for inputs to the next layer, the smallest BMSE is used as the criterion for stopping the whole network. A new layer is created if the BMSE of a layer is less than that achieved in a previous layer. When there is no reduction in BMSE, or if a single neuron remains, training is stopped. Training a new layer does not affect the results of previously trained layers.

When a layer shows an increase in BMSE the previous layer is the output layer and its output is that of the best neuron. The final structure is trimmed of all neurons not connected, directly or indirectly to the final neuron.

The output of a GMDH network having m layers can be expressed as a 2^m degree polynomial.

5.1.1 A GMDH training algorithm

Training is summarized by Pham and Lui (1995) as follows:

1) Pre-process the data. This is a data normalizing procedure, for inputs x and outputs y, use the following equations:

$$x_{norm} = \frac{x_i - \mu_x}{\sigma_x} \qquad y_{norm} = \frac{y_i - \mu_y}{\sigma_y}$$

Where x_i and y_i are the ith input and output experimental data pair, μ_x and σ_x, μ_y and σ_y; are the means and standard deviations respectively of the x_i's and y_i's.

2) Decide the external inputs to the network. For a modelling application, use m past inputs x(k - 1), x(k - 2), ... x(k - m), and n past outputs y(k - 1), y(k - 2), ... y(k - n). In prediction applications only n past outputs are used. Duffy and Franklin (1975) suggested that determination of m and n may be obtained by calculating correlation coefficients for the input output data.

3) Separate the experimental data into a training data set and a selection data set.

4) Create a set of N-Adalines based on the number of inputs, each pair of inputs produces an N-Adaline.

5) Initialise the weights of the neurons (N-Adalines) to zero.

6) Use the training data set to train all neurons in the created layer using the following procedure. At time k apply x(k-1), x(k-2), ... x(k-m), and y(k - 1), y(k - 2), ..., y(k - n) for a modelling task.

Apply y(k - 1), y(k - 2), ..., y(k - n) for a prediction task. Take y(k) as the desired output of all the neurons. Calculate the output errors of the neurons and modify their weights once. One epoch is complete when the whole training set has been presented to the network.

The squared error of each neuron is summed over the batch to obtain the BMSE if the result is smaller than that for the previous batch subject the batch to further training else stop. Current layer training stops when training for all elements in that layer stops.

7) Input the selection data set to the network. Obtain the BMSE for the layer just trained and the ratio of each MSE to the smallest MSE. From these MSEs assign a threshold, retain those processing elements whose ratios are below the threshold as post selection elements for the next layer

8) If the BMSE in the layer selected is larger than that of the previous layer or if the current layer has a single unit stop training. If training stops due to an increasing BMSE, or because a single element remains, use the preceding layer as the output layer and trim the network. Otherwise use post selection neurons to create a new layer and return to step 5.

9) Test the network with the evaluation data set, as only the training set is used in the determination of network weights generalization may be tested by using the combined training and evaluation sets.

A common approach to solving such models is founded in regression analysis, with the primary step being decision on the type of polynomial that regression should find.

Input variables are selected with their covariants and trivariants as the terms of the polynomial:

$(x_1, x_2, x_3, \ldots, x_1^2, x_2^2, x_3^2, \ldots, x_1^3, x_2^3, x_3^3, \ldots, \ldots, x_1^n, x_2^n, x_3^n \ldots)$.

The next step is to construct a linear combination of all of the polynomial terms with variable coefficients. The algorithm determines values of these coefficients by minimizing the squared sum (over all samples) of differences between sample outputs and model predictions.

Required complexity is a major aspect of choosing the set of polynomial terms. If we include in the model, polynomials of one variable, what should be the largest degree of that polynomial? Should it be 3, or should the model evaluate terms such as x^5? GMDH has in practice worked better than regression to answer this question if exhaustive analysis is not used.

5.1.2 Selection Criterion

The relative quality of each prospective model must be evaluated using some numeric criterion. A simple criterion, with it's origins in linear regression analysis, is the sum of the squared differences between the network output and the model prediction divided by the sum of the squared network output. (Normalized Mean Squared Error called the Training Squared Error, TSE).

Training square error on practical data, with or without noise factors, decreases with the addition of extra terms to the model. If you use only TSE, which determines the quality of the model by evaluating the same information you have already used to build the model. The results is an "over complex" model with an inability to generalise effectively because it pays too much attention to noise in the training data. This is the equivalent to over training in other neural nets.

This problem is negated by, using data other than that which was used to build the evaluated model. We may compute the squared sum of differences between the known output and model prediction a test set of data. An alternative strategy to avoid over fitting is to introduce a penalty for model complexity, using the Predicted Squared Error (PSE) criterion introduced by Barron, (1996).

5.1.3 Multi-layer procedure for GMDH

Computation time is reduced by decreasing the number of polynomial terms (and the number of input variables), used to build the models to be evaluated. The direct procedure of model selection is changed to a multi-layer procedure.

(i) The first pair of input variables ,(x_1 , x_2), yield a simple set of polynomial terms, (1, x_1, x_2, x_1x_2), (1 will represent the constant term).

(ii) Review all possible models made from these terms, and choose one which is the best. (Each of the evaluated models is called a candidate for survival.)

(iii) Repeat steps (i) and (ii) for each variable pair, for n input variables, $n(n - 1)/2$ candidates for survival are generated, each with its own value of the evaluation criterion.

Compare the values from step (iii) choose several candidates for survival which give the best approximation of the output variable. A pre-defined number of the best candidates for survival are stored in the first layer of the net and are preserved for the next layer.

The layer of survivors is used for inputs to build the next layer in the network. The original network inputs in the first layer may also be chosen as inputs to this new layer. The next layer is built with polynomials of this broadened set of inputs. Note that since some inputs are already polynomials, the next layer may contain very complex polynomials. The layer building GMDH procedure continues as long as the evaluation criteria continues to diminish.

5.2 Water level prediction using a GMDH network

This example is taken from an analysis Pham and Liu (1995) of data collected from correlation studies by Weisberg (1985) and Gentry and Lopez-Parodi (1980) between deforestation and Amazon flooding.

Year	High level metres	Low level metres
1962	25.82	18.24
1963	25.35	16.50
1964	24.29	20.26
1965	24.05	20.97
1966	24.89	19.43
1967	25.35	19.31
1968	25.23	20.85
1969	25.06	19.54
1970	27.13	20.49
1971	27.36	21.91
1972	26.65	22.51
1973	27.13	18.81
1974	27.49	19.42
1975	27.08	19.10
1976	27.51	18.80
1977	27.54	18.80
1978	26.21	17.57

Amazon flooding data.

Table 5.1

For the simulation of high water prediction, using the GMDH network, four input units were used. The first eleven data being used for training and the full data set for evaluation.

For the simulation of low water prediction, using the GMDH network, four input units were used. The first fourteen data being used for training and the full data set for evaluation.

A comparative evaluation was made using a multi-layer perceptron network, with four input units, eleven data for training and full data set evaluation at the high level.

A low level evaluation made using a multi-layer perceptron network, with four input units, used fifteen data for training and full data set evaluation.

Pham and Liu (1995), provide a group method data handling programme from which the following graphical data may be obtained. The programme is written in the 'Microsoft Quick C (version 1.0)' language. The authors do not provide the GMDH equation.

Group method data handling and multi-layer perceptron network outcomes for the data favour GMDH. Results are estimated from the graphic plots produced by Pham, D.T. and Liu, X. (1995) and copied in Microsoft Excel (Office 2000).

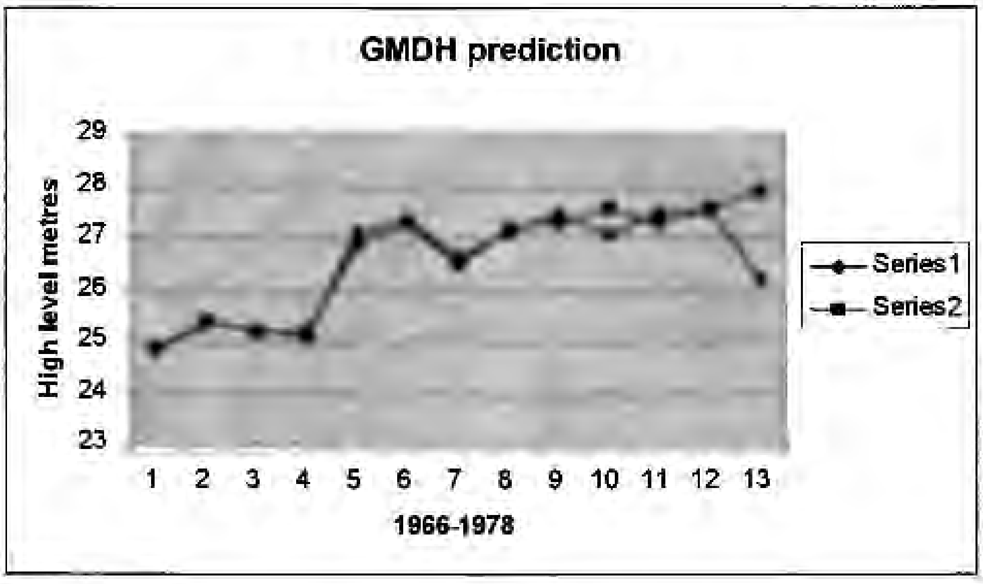


Figure 5.3

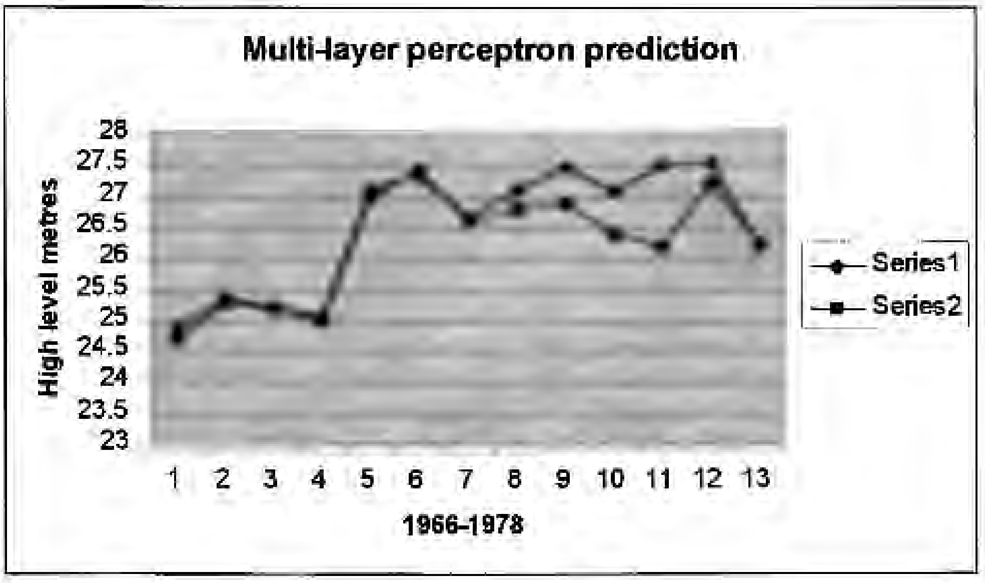


Figure 5.4

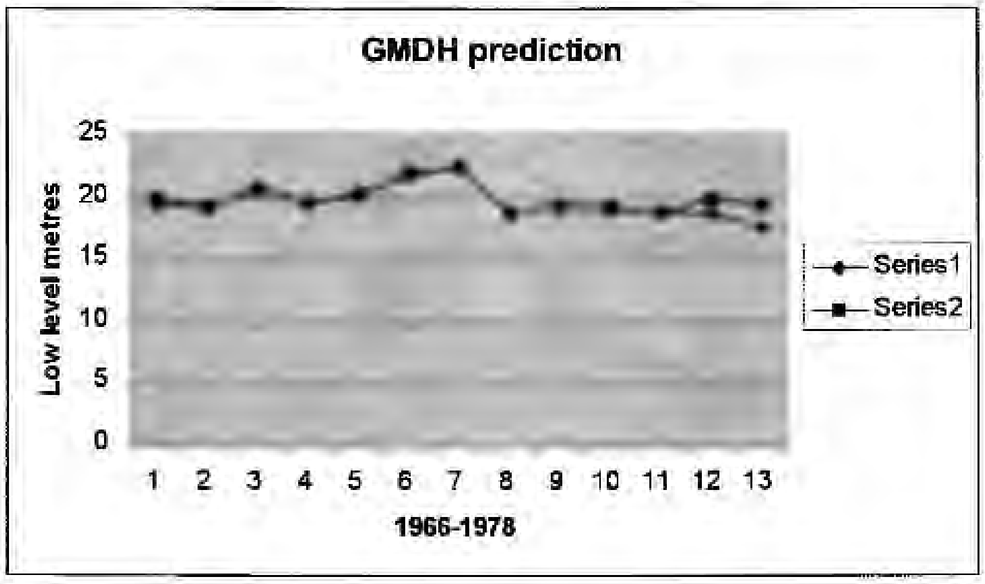


Figure 5.5

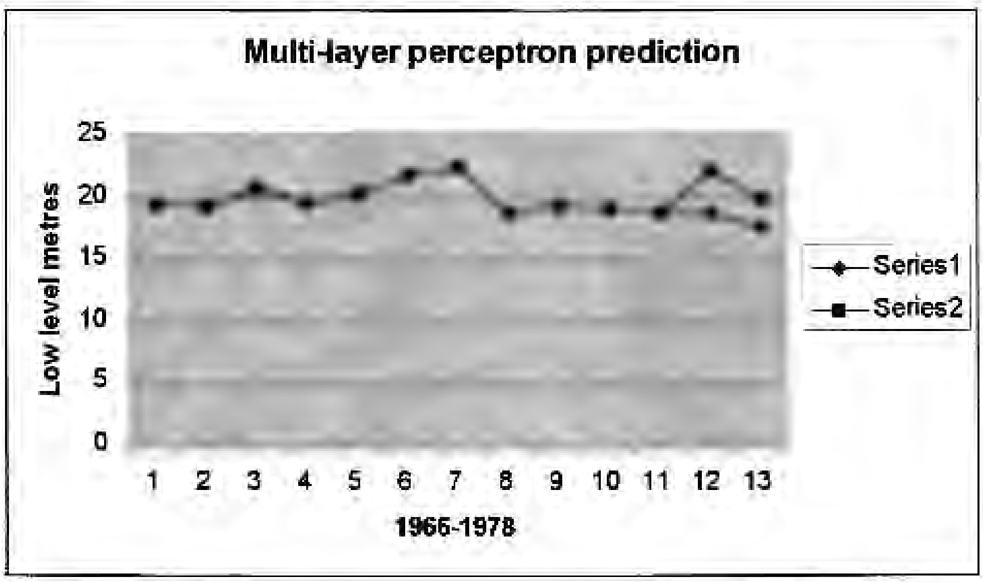


Figure 5.6

Examples

Q1 What type of neurons are used in the construction of GMDH networks and what learning rule is generally applied ?

A1 Use is made of adalines with non linear pre-processors. The Widrow-Hoff learning rule is used.

Practical examples for this chapter would require the use of a suitable programme for GMDH.
Available programmes include:
GMDH in Neuroshell 2 Ward Systems
Email: WardSystems@msn.com

A programme has been written in Microsoft Quick C (version 1.0) by DT Pham and X Lui, *Neural Networks for Identification, Prediction and Control,* Publisher, Springer, (1995).

Other programmes in the appendices include:
Multilayer perceptron for identification.
Modified Elman network for identification.

A series of programmes in ANSI C have been written by Karsten Kutza in June 1996. As a consequence of my computers being stolen I no longer have direct internet references. My download shows the following programmes.
Adaline, ART 1, BAM, Boltzman, BPN (backpropagation), CPN (counterpropagation), Hopfield and self organizing maps.

Adam Blum has published *Neural networks in C++*, Wiley and Sons inc. (1992). The programmes, written in Turbo C++, include:
A makefile programme for construction of the given networks.
Vector and matrix classes.
An abstract neural network base class.
Backpropagation. Counterpropagation. BAM. Hopfield.

References

[1] Duffy, J.J. and Franklin, M.A. A learning identification algorithm and its application to an environment system. *IEEE Trans. Systems , Man ,and Cybernetics,* 5(2), pp 226-240. (1975).

[2] Ivakhnenko, A.G. Polynomial theory of complex systems, *IEEE Trans. Systems, Man, and Cybernetics,* 12, pp 364-378 (1971).

[3] Pham, D.T. & Lui, X. Neural Networks for Identification Prediction and Control, Springer. pp 67-79 (1995).

[4] *Neuroshell 2* A proprietary Neural network programme by Ward systems (1996).

[5] Barron, A. R. *Neuroshell 2*, Ward systems(1996).

[6] Gentry, A.H. and Lopez-Parodi, J. Deforestation and increased flooding of the upper Amazon, *Science* 210(19), 1354-1356, (1980).

[7] Weisberg, S. *Applied Linear Regression.* Wiley, New York, pp 31-32 (1985).

Chapter 6

RECURRENT NETWORK DESCRIPTIONS

Summary

Recurrent neural networks have feedforward and feedback connections so that signals may be propagated in opposite directions. Each pattern passes through one or more neurons, two or more times, before an output response is generated in this network.

The networks explained are feedforward backpropagation, Hopfield, Boltzmann, Bi-directional Associated Memories, Elman and Counterpropagation.

6.1 Feedforward backpropagation

The network's name is based on its method of handling errors. As we have already observed, the perceptron network can train the output units to classify input patterns, subject to the classes being linearly separable.

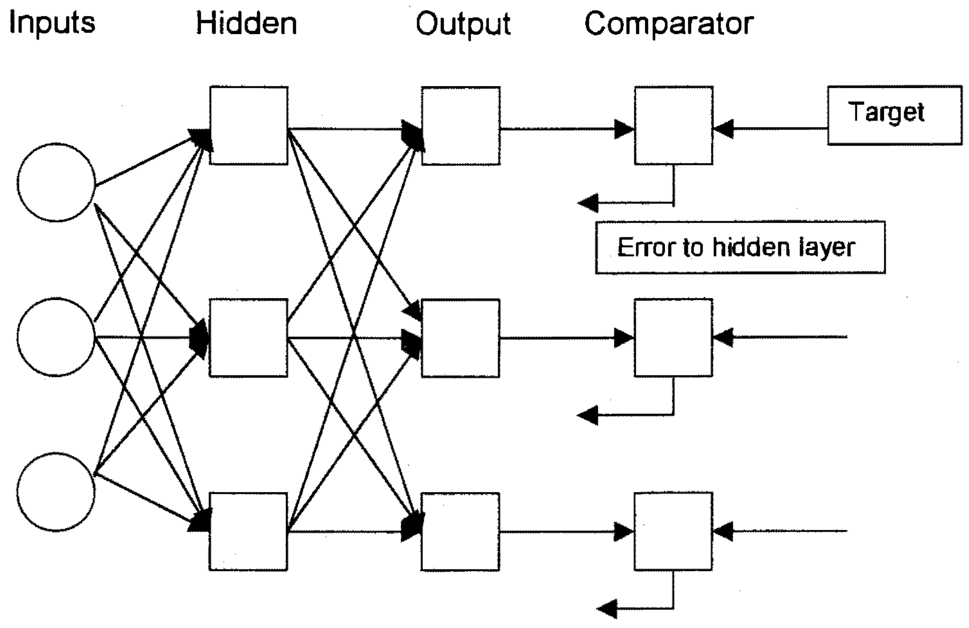
When the classes are not linearly separable the solution may be found through a multi-layer network. Minsky and Papert (1948) suggested that output errors would require adjustment to neurons or adjustments of inter-connections. Back-propagation offered a solution by apportioning the error to all neurons and connections. Apportioned error is by propagating output error backward through the connections to each preceding layer and to the input vector.

A typical back-propagation network has an input vector, one or more hidden layer(s) and an output layer. Commercially available networks normally have three or less hidden layers. Each layer being completely connected to the preceding layer. Whilst the network is being trained the level of error is used to update the connection weights.

Neurons in the hidden layer "fire" or produce outputs that are based upon the sum of weighted values passed to them. The hidden layer passes values to the output layer in the same fashion, and the output layer produces the desired results (predictions or classifications).

A vector of inputs **x** is applied to the hidden layer neurons, each of these is multiplied by a connection weight, and the products are summed. This summation of products $\mathbf{w}^T \mathbf{x}$ must be calculated for each neuron in the network.

After $\mathbf{w}^T \mathbf{x}$ is calculated, an activation function f is applied to modify it, thereby producing the signal $f(\mathbf{w}^T \mathbf{x} + b)$. The activation function usually used for backpropagation is sigmoid, because it has a simple derivative, used in implementing the backpropagation algorithm.



A backpropagation network

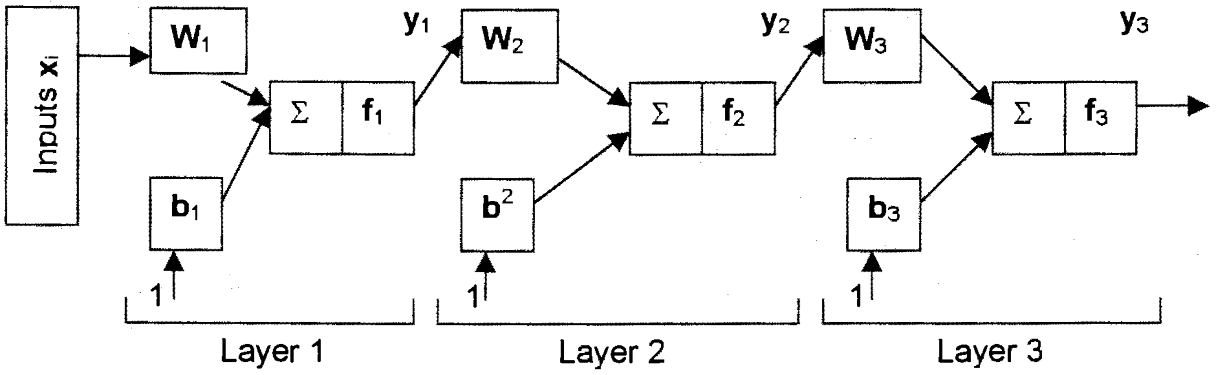
Figure 6.1

Multi-layer networks have greater representational power than single-layer networks only if a non-linearity is introduced. The sigmoid (squashing) function produces the required non-linearity. Other functions maybe used to satisfy the backpropagation algorithm requirement that the function be everywhere differentiable. Backpropagation is applicable to networks with any number of layers.

6.1.1 Backpropagation in multi-layer perceptron training

Single layer networks have been trained by use of the delta rule, Rosenblatt (1975), or the LMS rule Widrow and Hoff (1976). They are limited in their application to linear separable problems.

The backpropagation algorithm, which is a gradient descent algorithm, is often used for multi-layer perceptron training. This algorithm was proposed in a thesis by Werbos (1974). The algorithm was re-discovered independently by Parker (1985), LeCun (1985) and Rumelhart, Hinton and Williams (1986).



$y_1 = f_1\,(W_1 x_1 + b_1)$, $y_2 = f_2\,(W_2\, y_1 + b_2\,)$, $y_3 = f_3\,(W_3\, y_2 + b_3\,)$,

$y_3 = f_3\,(W_3\, f_2(W_2\, f_1(W_1 x_1 + b_1) + b_2\,) + b_3\,)$.

A block diagram of a three layer network

Figure 6.2

6.1.2 Backpropagation training

The backpropagation algorithm for multi-layer networks is a generalization of the least mean squares (LMS) algorithm. Training assumes that each input vector $\mathbf{x}_i$ is paired with a target vector $\mathbf{t}_i$ representing the desired output. ($\mathbf{x}_i$, $\mathbf{t}_i$) are called a training vector pair. Usually, a network is trained over a number of training pairs.

The training procedure is as follows:

Procedure

1 Initialise all weights to small random numbers.

2 Select the next training pair from the training set; apply the input vector to the network.

3 Calculate the output of the network.

4 Calculate the error between the network output and the target output.

5 Adjust the weights of the network to minimise the error.

Repeat steps 2 to 5 until the error is sufficiently low.

6.1.3 Exclusive OR implementation in backpropagation

This problem was the reason that many researchers abandoned neural network studies in the 1960's. Some twenty years later the problem had been solved by several methods including the multi-layer feedforward backpropagation network.(Li Min Fu, 1994)

The following table indicates for binary inputs a and b the expected output S.

a	B	S
0	0	0
1	0	1
0	1	1
1	1	0

An exclusive OR table.

Table 6.1

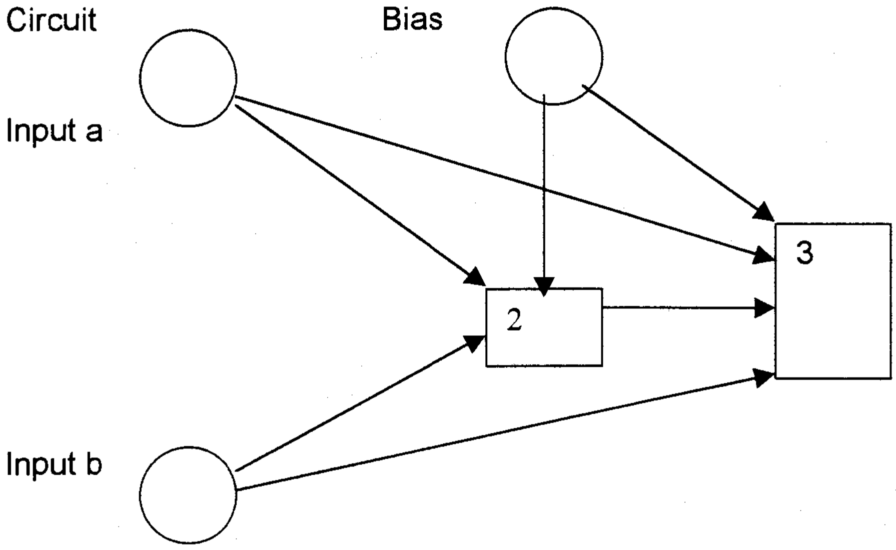


Figure 6.3

Processing

Randomise initial weights to small values

$w_{a3} = 0.02$, $w_{b3} = 0.03$, $w_{23} = -0.02$, $w_{a2} = 0.01$, $w_{b2} = 0.02$,

$w_{bias3} = -0.01$, $w_{bias\,2} = -0.01$.

For input pattern $\mathbf{x}^T = (1, 1)$, required output (0) and a sigmoid activation function, hidden unit 2 is H_2 and output is O_3 then:

$H_2 = 1/(1+e^{-[1(0.01)+1(0.02)-1(0.01)]}) = 0.505$

$O_3 = 1/(1+e^{-[(0.505)(0.02+1(0.02)+1(0.03)-1(0.01)]}) = 0.508$

Assume a learning rate $\eta = 0.03$

$\delta_1 = 0.508(1-0.508)(0-0.508) = -1.27$

$\Delta w_{13} = 0.3(-0.127)1 = -0.038$

$\delta_1 = 0.505(1-0.505)(-0.127)(-0.02) = 0.0006$

$\Delta w_{23} = 0.3(0.0006)1 = -0.0002$

After several similar iterations the weights become

$w_{a\,3} = 4.98$ $w_{b\,3} = 4.98$ $w_{2\,3} = -11.30$ $w_{a\,2} = 5.62$ $w_{b\,2} = 5.62$

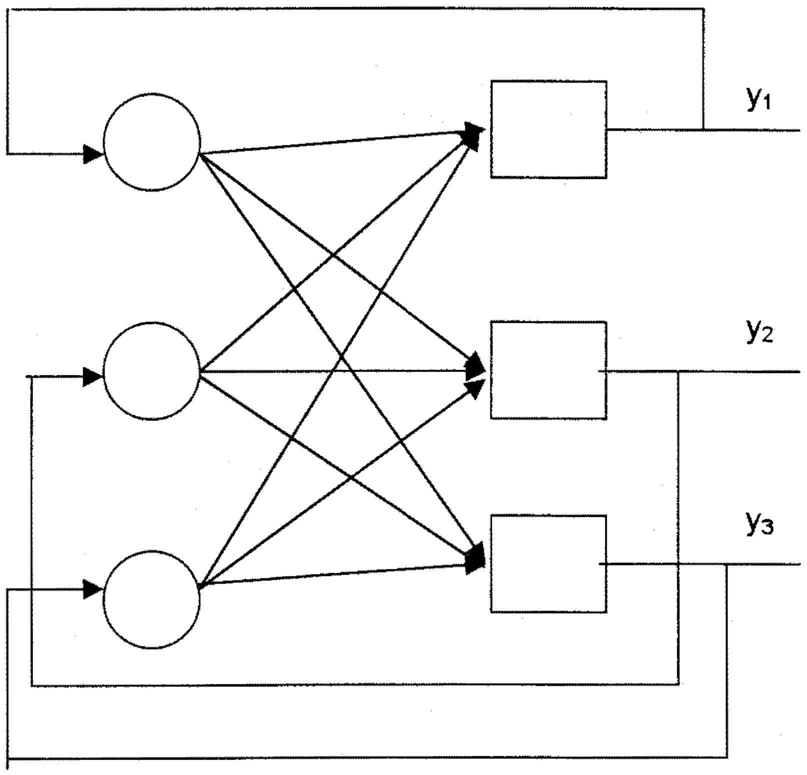
$w_{bias3} = -2.16$ $w_{bias\,2} = -8.83$

This yields a mean square error of less than 0.01

(Li Min Fu, 84, 1994)

6.1.4 Single layer recurrent network.

The following diagram shows a typical single layer recurrent network having three output neurons that feedback to the inputs.



Single layer recurrent network

Figure 6.4

The recurrent networks studied are the Hopfield network, Boltzmann, bi-directional associative memories (BAM), Elman and counter-propagation networks.

6.2 The Hopfield Network

Neural networks are complex and often contain non-linear components. Their behaviour is difficult to analyse. Hopfield (1982), applied some ideas to them from an important and developing area of mathematics called non-linear dynamical system theory.

Hopfield was able to show that sometimes a non-linear neural network evolving in time could be analysed qualitatively. As the neural network evolves, it minimizes a particular function related to what would be called energy in a physical system.

LiMin Fu (1994) observes that this network may be used for auto-associative or optimisation tasks. It is a neural network model, which exhibits the characteristic of associative memory in that, like human memory, a partial recollection produces a larger related memory. I.e. A few notes may trigger memory of a musical score.

The system may have n elements with connections between any element pair. The system energy E is the sum, over the n elements, of the energies of each unit pair i and j and the strength of their connection. It may be of value to visualise a mechanical, electrical, or chemical simulation of this concept.

Conceptually the Hopfield network considers that within a single network, binary valued neurons can store multiple stable states. A network is constructed of binary valued neurons connected to each other but not self connected. All connection weights are symmetric ($w_{ij} = w_{ji}$) and the network may have a set of stable states in which each binary neuron has a value –1 (0) or 1. For a given input pattern the network converges to the stable state closest to the pattern.

6.2.1 A Hopfield auto-associative algorithm.

LiMin Fu provides an auto-associative algorithm for m stored patterns in a Hopfield network:

$$w_{ji} = \sum_{p=1}^{m} P_{j,p} P_{i,p}, \quad i \neq j \text{ else } 0$$

w_{ji} is the connection weight from neuron i to neuron j and $P_{i,p}$ is the ith component in pattern vector **p**.

Activation calculation

1) At time t = 0 the activation level of neuron j is $O_j(0) = P_j$ the jth component of the input pattern.

2) At time t > 0, activation level of neuron j is $O_j(t+1) = f(\Sigma w_{ji} O_j(t))$ for a hard limiting function or smooth function.

$$f(\Sigma w_{ji} O_j(t)) = f(a) = \begin{cases} 1 & \text{for} \quad a > 0 \\ -1 & \text{for} \quad a < 0 \\ \text{unchanged} & a = 0 \end{cases}$$

Step 2 is repeated until neurons remain unchanged this is then the best match for the unknown input.

Note: In a Hopfield network it is desirable that a fixed point satisfies:
$\mathbf{x} = f(\mathbf{xw})$

Example

LiMin Fu shows how a weight matrix may be constructed for an associative Hopfield network. $\mathbf{x}_i$ is an n dimensional bipolar vector to be stored in the network and I_n is an n*n identity matrix.

If $\mathbf{x}_1^T = (1, -1, -1)$, $\mathbf{x}_2^T = (-1, 1, -1)$, $\mathbf{x}_3^T = (-1, -1, 1)$, Use $\mathbf{W} = \Sigma\, \mathbf{x}_i^T \mathbf{x}_i - I_n$,

$\mathbf{W} = (\mathbf{x}_1^T \mathbf{x}_1 - I_3) + (\mathbf{x}_2^T \mathbf{x}_2 - I_3) + (\mathbf{x}_3^T \mathbf{x}_3 - I_3)$,

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & -1 \\ -1 & 0 & -1 \\ -1 & -1 & 0 \end{bmatrix}$$

Using $\mathbf{x}_1$ as the probe vector: (The probe vector is one we use to perform association).

$f(\mathbf{x}_1\ \mathbf{W}) = f(2,0,0) = (1,-1,-1)$ which is the $\mathbf{x}_1$ vector, in this case a single iteration only was required. The capacity of the net dictates if all vectors may be correctly retrieved.

A Hopfield net limitation is that the minimum achieved is not necessarily a global minimum. A further limitation being that if too many patterns are stored, the network randomises and can no longer serve as a memory.

6.2.2 Hopfield network capacity

The Hopfield net can be activated either synchronously or asynchronously. In the synchronous mode, the weights are updated simultaneously. In asynchronous operation, the network updates only one neuron at a time. The neuron to be updated is selected from n neurons with a probability of 1/n of selection.

Defining memory capacity of the Hopfield associative network. If a binary n-dimensional vector **x** is a memory, then for neuron i = 1, ..., n:

$$x_i = f_h \left(\sum_{j=1}^{n} w_{ij} x_j\right) \text{ where } f_h \text{ is the hard-limiting function.}$$

Similar states map to a common memory which leads to it being called an attractor. If m, n dimensional, memory vectors $\mathbf{x}_1$ are to be stored, each neuron being 1 or -1. The outer product construction method is used to arrive at the appropriate weight matrix **W**, the construction method is used to ensure convergence to a stable state.

The following example was formulated by McEliece (1987)

Three memories of five dimensions are to be stored, their values being:
$\mathbf{x}_1 = (1, 1, 1, 1, 1)$,
$\mathbf{x}_2 = (1, -1, -1, 1, -1)$,
$\mathbf{x}_3 = (-1, 1, -1, -1, -1)$.
Asynchronous operation is assumed.

The weight matrix by the outer products method as given by LiMin Fu is:

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & 1 & 3 & 1 \\ -1 & 0 & 1 & -1 & 1 \\ -1 & 1 & 0 & 1 & 3 \\ 3 & -1 & 1 & 0 & 1 \\ 1 & 1 & 3 & 1 & 0 \end{bmatrix}$$

If the probe vector is $\mathbf{x} = (1, -1, -1, 1, 1)$, which has a Hamming distance of 1 from $\mathbf{x}_2$, product $\mathbf{xW}$ is calculated, hard-limited to (1, -1) to then yield $\mathbf{x}'$: (The Hamming distance calculates the number of bits that differ in two vectors).

$\mathbf{x}' =. (l, -1, 1, 1, -1)$.

For asynchronous operation, there is no set order of neuron updating. For this example, let the third element be chosen.

The new vector will be: $\mathbf{x}' = (l, -1, 1, 1, 1)$.

The steps are repeated until one of the three memories (fundamental memories) is reached. Eventually the second neuron of **x** will be updated to yield **x**" = (1, 1, 1, 1, 1) which equals $\mathbf{x}_1$.

Note, the system does not converge to $\mathbf{x}_2$ which is closest to the initial test vector.

This example reveals one of the problems with the Hopfield net. The probe may not settle onto one of the memories. If it is a memory, then it is not necessarily the closest memory.

6.3 Hebb's learning

The foundation for many of the current training algorithms arose from the work of Hebb (1949). Prior to Hebb's work, it was generally recognised that learning in a biological system involved some physical change to the neurons, but no clear idea had been formulated to explain how this could take place. Hebb's learning proves useful in pattern recognition.

Hebb's postulate:

"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."

In neural network terms if neuron A is repeatedly activated by neuron B then A will become more sensitive to stimuli from B, resulting in a more efficient synaptic connection from B to A. The theory, sometimes known as correlation learning, involves only local interactions between neurons with no global teacher; hence, the training is unsupervised.

Anderson, (1995), notes that the work by Hebb did not include a mathematical analysis, but the clarity of the idea received wide acceptance.

This idea is expressed by LiMin Fu in the equation that follows:

$w_{ji}(t + 1) = w_{ji}(t) + N_j N_i$

where $w_{ij}(t)$ = the synaptic strength from neuron i to neuron j: N_i = the excitation level of the source neuron N_j = the excitation level of the destination neuron

Hebb's concept offered an intuitively sound answer to the question of how learning; could take place without a teacher. In this system, learning involves only two neurons and the related synapse; a global feedback system is not required for the neural patterns to develop.

Hebb's learning has resulted in many successes, although it has also revealed its limitations; some patterns are not able to be learned by this method. Many learning methods have evolved from this foundation work.

6.3.1 Hebb's Learning Algorithm

Hebb's learning rule has been used to identify minima to the location of stored patterns. A network using the sigmoidal activation function with Hebbian learning is said to employ signal Hebbian learning. In this case, the Hebbian equations are modified to the form that follows:

$O_i = l(l + e^{-N}) = f(N_j)$
$w_{ji}(t + 1) = w_{ji}(t) + O_j O_i$

where $w_{ji}(t)$ = the synaptic strength from neuron i to neuron j : O_i = the output level of the source neuron = $f(N_i)$ and O_j = the output level of the destination neuron = $f(N_j)$

6.4 Boltzmann Machine.

Constant energy minimization causes the Hopfield network to settle into local minima, the Boltzmann machine is similar in function and operation to the Hopfield network. Its advantage is that by the use of a technique known as simulated annealing it searches the pattern layer's state space for a global minimum.
In a similar structure to the Hopfield network, the Boltzmann machine has an associated state space energy dependent on the connection weights in the pattern layer. Through the submission of a set of training patterns the learning process minimizes this state space energy. As a consequence of the learning process the Boltzmann machine will progress to connection weights that approach a global minimum.

Note: Annealing is a metallurgical process in which a metal is gradually cooled, successive lower thermal energy states leading to a minimum state being achieved.

The simulated annealing schedule is an added to the learning process of the network. In the metallurgical annealing process the high initial temperatures are decreased over time. The high initial weights in the Boltzmann network act as noise factors in the processing elements. As the weights reduce, the network settles to a minimum.

When the machine learns at high weights it behaves as a random movement model, whilst at low weights it behaves as a deterministic model in moving to the global minimum.

The initial random movement in annealed learning may mean that a processing element assumes a state that has increased rather than decreased the total system energy. This simulation of the metallurgical process is helpful in escaping local minima and moving toward a global minimum.

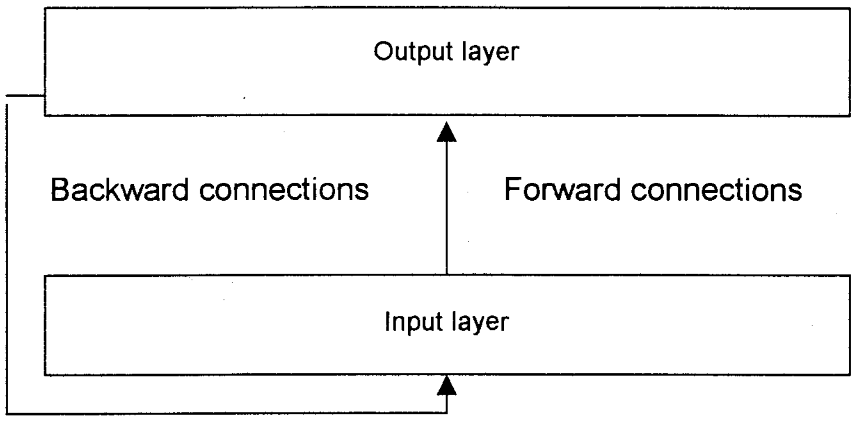
When learning is completed for a set of submitted patterns, presentation to the network of a part of a pattern related to the set results in the net completing the missing portion.

There is a limitation on the number of classifications, for the Hopfield and Boltzmann networks being less than fifteen percent of the total processing elements in the input layer. (Based on downloaded data from internet).

6.5 Bi-directional associative memories BAM

These networks have a two layer recurrent architecture, the forward and backward information flow between the layers searches for stimulus and response associations.

Given a partial memory, with or without noise caused by input corruption, auto-association permits retrieval, the BAM network extends this capability to relate an input vector to another vector thus permitting a generalization of outputs for similar inputs.



Bi-directional associated memory.

Figure 6.5

6.5.1 BAM algorithm

Weight assignments

If the network stores m pairs of patterns, forward weights are given by: $w_{ji} = \Sigma P_{i,p}\, Q_{j,q}$ where w_{ji} is the connection weight from i to j, $P_{i,p}$ is the ith component in vector pattern p, $Q_{j,q}$ is the jth component in vector pattern q, patterns p and q form a vector pair.

Activation calculation

Step 1 Initialize the input units at time zero

$O_i\,(0) = P_{i,}$

Step 2 At time = t (t>0)

$O_j(t+1) = f_h\,(\Sigma w_{ji}\, O_i\,(t))$

Where $O_j(t)$ is the activation level of unit j at time t and f_h is a hard limiting function, or a sigmoid function.

$$f_h\,(a) = \left[\begin{array}{ll} 1 & \text{for } a > 1 \\ -1\ (0) & \text{for } a < 0 \\ O_j(t) & \text{for } a = 0 \text{ (no state change)} \end{array}\right.$$

Step 2 is repeated until equilibrium is achieved. LiMin Fu (1994)

Example

This example is taken from LiMin Fu (1995).

Use a bi-directional associative memory to store the following three pairs of vectors:

Original vector	Associated vector
A$_1$ = (1, -1, -1)	**B**$_1$ = (1, -1)
A$_2$ = (-1, 1, -1)	**B**$_2$ = (-1, 1)
A$_3$ = (-1, -1, 1)	**B**$_3$ = (1, -1)

A vector storage table.

Table 6.2

The outer products are used to construct the weight matrix:

$$\mathbf{W} = \sum_{i=1}^{3} \mathbf{A}_i^T \mathbf{B}_i$$

$$\mathbf{W} = \begin{bmatrix} 1 & -1 \\ -3 & 3 \\ 1 & -1 \end{bmatrix}$$

If $\mathbf{A}_1$ is used as the probe vector, and **f** is a bi-polar activation function.

$\mathbf{f}(\mathbf{A}_1\mathbf{W}) = \mathbf{f}(3, -3) = (1, -1) = \mathbf{B}_1$

If $\mathbf{B}_1$ is used as the probe vector:

$\mathbf{f}(\mathbf{B}_1\mathbf{W}^T) = \mathbf{f}(2, -6, 2) = (1, -1, 1)$ □ $\mathbf{A}_1$, more iterations may be required to retrieve correctly. A complication in this example is that $\mathbf{B}_1 = \mathbf{B}_3$.

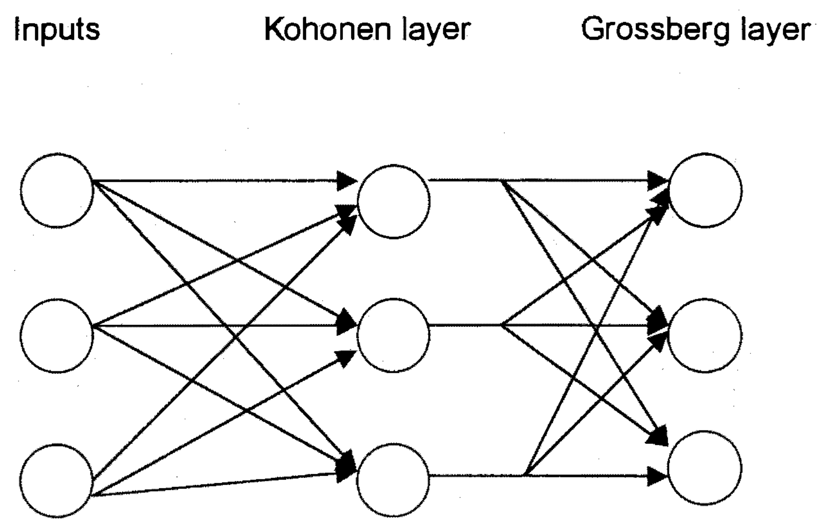
6.6 Counter-propagation Network.

Combining an unsupervised Kohonen layer with an output layer that was trained developed the counter-propagation network. This network is able to process complex classification problems, whilst minimizing the number of neurons and training time. Hecht-Nielsen, (1987)

The counter-propagation network is similar to the learning vector quantization network, the Kohonen layer behaves as a comparator, finding the closest fit to an input stimulus and outputting its equivalent mapping.

The first counter-propagation network used bi-directional mapping between the input and output layers. The data to the input layer generated an output classification pattern; the output layer would then receive an additional input vector, from which it would feed an output classification to the network's input layer. The flow and counterflow of data gave the network its name.

Current development uses a uni-flow variant of this formal representation of counter-propagation. There is only one feedforward path from input vector to output layer. The network is shown in Figure 6.5, it is a two layer uni-directional counter-propagation type.



. A counter-propagation Network

Figure 6.6

It is usual practice to normalize inputs, for every combination of input values, vectors are unit length. The network inputs are then fed to a self-organizing Kohonen layer, this in turn feeds a Grossberg outstar layer. This layer uses the delta Rule to modify its incoming connection weights.

The input layer size depends upon the number of parameters that define the problem. With too few neurons the network may not generalize sufficiently, too many may cause a lengthy processing time.

When input data has not been normalized a layer may be introduced between the between the inputs and the Kohonen layer to perform this task. This layer requires one neuron for each input, plus one more for a balancing element.

This pre-processing layer modifies the input sets so that they combine to the same total, permitting the Kohonen layer to find the correct class for the problem.

Normalization is necessary with large input vectors to prevent bias of the Kohonen neurons such that weaker value input sets cannot be properly classified. The competitive nature of the Kohonen layer results in the larger value input vectors being dominant over the smaller vectors.

Counterpropagation uses a standard Kohonen paradigm, which self-organizes the input sets into classification zones. It follows the classical Kohonen learning law, (described in section 4.2). This layer acts as a nearest neighbour classifier. The neurons in the competitive layer autonomously adjust their connection weights to divide up the input vector space in approximate correspondence to the frequency with which the inputs occur.

There need to be at least as many neurons in the Kohonen layer as output classes. The Kohonen layer usually has many more neurons than classes simply because additional neurons provide a finer resolution between similar objects.

The output layer for counter-propagation is basically made up of neurons which learn to produce an output when a particular input is applied. Since the Kohonen layer includes competition, only a single output is produced for a given input vector. This layer provides a way of decoding that input to a meaningful output class. It uses the delta Rule to back-propagate the error between the desired output class and the actual output generated with the training set. The errors only adjust the connection weights coming into the output layer, the Kohonen layer is not affected.

Only one output from the competitive Kohonen layer is active at a time and all other neurons are zero. The only weights adjusted for the output neurons are the ones connected to the winning neuron in the competitive layer. In this way the output layer learns to reproduce a certain pattern for each active neuron in the competitive layer. If several competitive neurons belong to the same class, that output neuron will evolve weights in response to those competitive neurons and zero for all others.

There is a problem that could arise with this architecture. The competitive Kohonen layer learns without any supervision. It does not know what class it is responding to. This means that it is possible for a neuron in the Kohonen layer to learn to take responsibility for two or more training inputs that belong to different classes. When this happens, the output of the network will be ambiguous for any inputs that activate this neuron. To alleviate this problem, the neurons in the Kohonen layer could be pre-conditioned to learn only about a particular class.

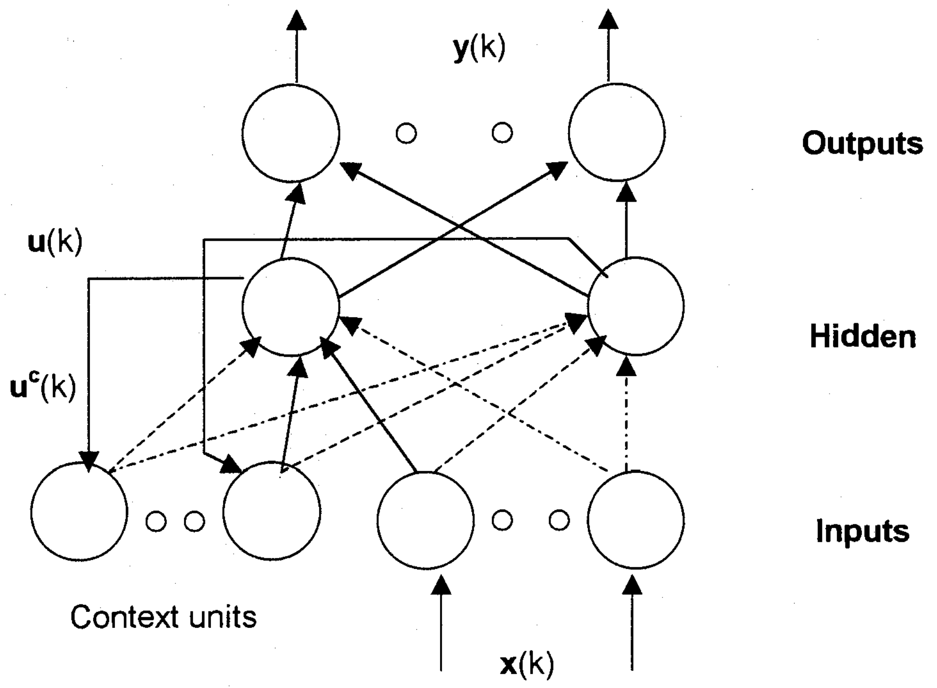
6.7 Elman Network

Pham and Lui (1995) describe this simple recurrent network that may be trained by the backpropagation algorithm. Inputs are fed to the hidden layer, which may have linear or non-linear activation functions. Feedback occurs from the hidden layer to context units that then join the input feeds.

Context units memorize some past states of the hidden units so the network outputs are a function of the current inputs and the previous states, this is referred to as having dynamic memory capability. The context units may be viewed as time delays. Outputs are linear from the summed signals to the output neurons.

Feed forward connection weights may be modified but the recurrent connections are fixed. The network was originally designed for speech processing applications.

The Elman network is shown in Figure 6.7



An Elman recurrent network.

Figure 6.7

6.7.1 Elman network analysis

Pham and Liu, (1995) offer the following analysis of the network. Vector and matrix notation have been changed to be in accord with previous work.

Set the input vector as $\mathbf{x}(k-1)$ and output vector $\mathbf{y}(k)$ with hidden units activation as $\mathbf{u}(k)$ with context unit outputs as $\mathbf{u}^c(k)$. From Figure 6.6 the following equations apply:

$\mathbf{u}(k) = \mathbf{f}(\mathbf{W}_2\, \mathbf{u}^c(k), \mathbf{W}_1\, \mathbf{x}(k-1))$,

$\mathbf{u}^c(k) = \mathbf{u}(k-1)$,

$\mathbf{y}(k) = \mathbf{W}_3\, \mathbf{u}(k)$, Where $\mathbf{W_1}$ is the input to hidden layer matrix of weights. $\mathbf{W_2}$ is the context layer to hidden layer matrix of weights. $\mathbf{W_3}$ is the hidden layer to output matrix of weights. **f** is a non linear activation function.

If the hidden units have linear activation functions, with zero biases for hidden and output functions, the above equations become:

$\mathbf{u}(k) = \mathbf{W_2}\,\mathbf{u}^c(k) + \mathbf{W_1}\,\mathbf{x}(k-1)$

$\mathbf{u}^c(k) = \mathbf{u}(k-1)$

$\mathbf{y}(k) = \mathbf{W}_{\mathbf{yx}}\,\mathbf{x}(k)$

The **x**(k) and **y**(k) equations, for the linear activation functions describe standard state space equations of dynamic systems. The order of the model is dependent on the number of states, which is the number of hidden units.

6.8 Probabilistic Neural Network.

The probabilistic neural network architecture was presented in two papers by Specht (1990). This network provides a general solution to pattern classification problems by following an approach, developed in statistics, called Bayesian classifiers.

Bayes theory, takes into account the relative likelihood of events and uses a priori information to improve prediction. The network paradigm also uses Parzen (1962) estimators, which were developed to construct the probability density functions required by Bayes theory.

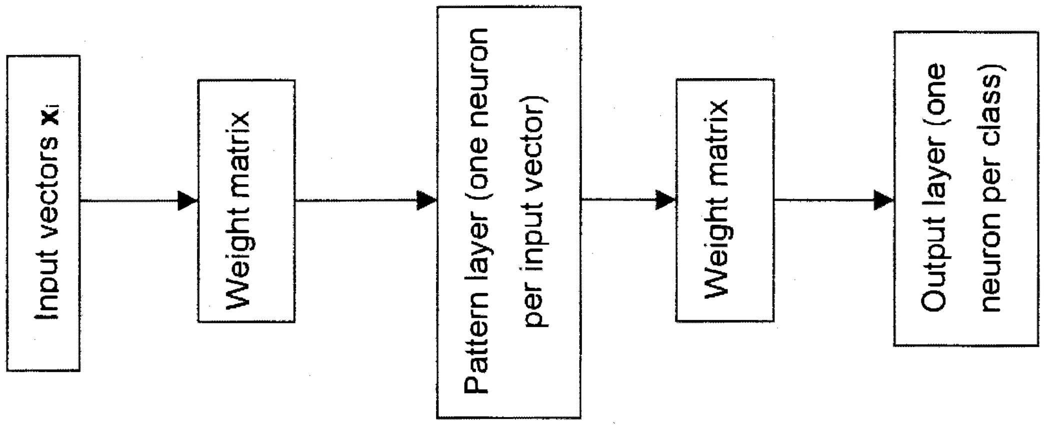
Hagan, Demuth and Beale, (1996) describe it as a parallel implementation of a standard Bayesian classifier. They then describe it as a three layer network that can perform pattern classification. (In their work the inputs are a layer). The statement that follows is that the network is not trained, "training vectors are the weight vectors in the first layer". The observation that the probabilistic neural network does not require training is given as its major advantage.

The disadvantage quoted is that the weight matrix can be very large if there are many vectors in the training set. An offered solution is a clustering operation to reduce the matrix size.

LiMin Fu, (1994), clarifies the method. The probabilistic neural network encodes each training pattern as the input weight vector. The net input to unit i being given by:

$z_i = \mathbf{x}\mathbf{W}_i$, where **x** and **W** are normalized to unit length.

An example of a probabilistic neural network is shown in Figure 6.8. This network has two layers. The network contains an input vector which has as many neurons as there are separable parameters needed to describe the objects to be classified. It has a pattern layer, which organizes the training set such that each input vector is represented by an individual neuron. Finally, the network contains an output layer, called the summation layer, which has as many neurons as there are classes to be recognized. Each neuron in this layer combines via neurons within the pattern layer which relate to the same class and prepares that category for output. As with the counter-propagation network, the input vector must be normalized to provided proper object separation in the pattern layer.


Input vectors $\mathbf{x}_i$
Weight matrix
Pattern layer (one neuron per input vector)
Weight matrix
Output layer (one neuron per class)

The probabilistic neural network.

Figure 6.8

LiMin Fu, (1994), summarizes:

Basic structures

Input vectors, a layer of pattern units and a layer of output units.

One to one correspondence between pattern units and training examples, pattern units = training examples.

Each output unit corresponds to a class (category).

A pattern unit connects to an output unit if and only if the corresponding example is labelled the corresponding class.

Weight initialisation

The input weight vector of pattern unit i is initialized to the ith pattern vector, which is normalized to unit length.

The input weight vector of every output unit is initialized to a vector of 1s.

Calculation of activation

The activation of input units are determined by the pattern presented to the network. The input vector is normalized to unit length.

The activation of O_j of pattern unit j is given by:

$$O_j = \exp[(\sum_i w_{ji} x_i - 1)/\sigma^2$$

Where w_{ji} is the connection weight from input unit i to pattern unit j and x_i is the activation of input unit i.

The activation of O_j of output unit j is given by:

$$O_j = 1/m\ P(\omega_j)\sum_i w_{ji} O_i$$

Where w_{ji} is the connection weight from pattern unit i to output unit j, m is the number of training examples labelled class ω_j and $P(\omega_j)$ is the prior probability of ω_j.

The decision of the network is the class corresponding to the network with maximum activation.

Network learning

The weights are non adjustable.

When a new pattern arrives, a pattern unit is added and the associated weights are initialized as previously stated. Nothing is said about the addition of outputs for further classes.

Examples

Q1 This example conveys the concept of associative memory retrieval.

Use the weight matrix (**W**) construction for an associative Hopfield network.

Inputs are bi-polar, (-1, 1)

determine **W** if $\mathbf{x}_1^T = (-1, -1, -1, -1, 1)$

$\mathbf{x}_2^T = (-1, -1, -1, 1, -1)$

$\mathbf{x}_3^T = (-1, -1, 1, -1, -1)$

$\mathbf{x}_4^T = (1, -1, -1, -1, -1)$

$\mathbf{x}_5^T = (1, -1, -1, -1, -1)$

Use $\mathbf{x}_3$ as the probe vector, is it retrieved on the first iteration?

A1 Using the outer products to construct the weight matrix:

$\mathbf{W} = \Sigma(\mathbf{x}_i^T\mathbf{x}_i - \mathbf{I}_n)$ where $\mathbf{x}_i$ is the given n-dimensional bipolar vector (1, -1).

$$\mathbf{W} = (\mathbf{x}_1^T\mathbf{x}_1 - \mathbf{I}_5) + (\mathbf{x}_2^T\mathbf{x}_2 - \mathbf{I}_5) + (\mathbf{x}_3^T\mathbf{x}_3 - \mathbf{I}_5) + (\mathbf{x}_4^T\mathbf{x}_4 - \mathbf{I}_5) + (\mathbf{x}_5^T\mathbf{x}_5 - \mathbf{I}_5)$$

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & -1 & -1 & -1 \\ -1 & 0 & -1 & -1 & -1 \\ -1 & -1 & 0 & -1 & -1 \\ -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & -1 & 0 \end{bmatrix}$$

Using $f(\mathbf{x}_3\mathbf{W}) = f(2, 2, 4, 2, 2) = (1, 1, 1, 1, 1)$ it is not retrieved.

Q3 Bi-directional associated memories may be used in pattern relationships.

If a BAM network is used to store the tabled vector pairs, construct the weight matrix and determine if:

Does $\mathbf{A}_1$ used as the probe vector retrieve $\mathbf{B}_1$ correctly on the first iteration?

Does $\mathbf{B}_1$ used as the probe vector retrieve $\mathbf{A}_1$correctly on the first iteration?

Original vector	Associate vector
$\mathbf{A}_1 = (-1, -1, 1, -1)$	$\mathbf{B}_1 = (-1, 1, -1)$
$\mathbf{A}_2 = (-1, -1, -1, 1)$	$\mathbf{B}_2 = (-1, -1, 1)$
$\mathbf{A}_3 = (-1, 1, -1, -1)$	$\mathbf{B}_3 = (1, -1, -1)$
$\mathbf{A}_4 = (1, -1, -1, -1)$	$\mathbf{B}_4 = (-1, -1, 1)$

A3 By outer products $\mathbf{W} = \begin{bmatrix} 4 & 0 & -2 \\ 0 & 0 & 2 \\ 0 & 0 & 2 \\ 0 & 4 & 2 \end{bmatrix}$

$f(\mathbf{A}_1\mathbf{W}) = f(-4, -4, 0) = (-1, -1, -1)$ which is not correct.

$f(\mathbf{B}_1\mathbf{W}^T) = f(-2, -2, -2, 2) = (-1, -1, -1, 1)$ which is not correct. Note, on this iteration $\mathbf{B}_1$ has retrieved $\mathbf{A}_2$.

Q3 Kohonen self-organizing maps cluster results into a specified number of classes, this example illustrates the clustering concept. This is based on an unworked and unexplained example from LiMin Fu 1994.

Gluck and Corter, (1985), devised the following formula for comparison of clusterings with different numbers of categories.

$$\frac{\sum_{k=1}^{K} P(C_k)\sum_{I}\sum_{j} P(A_i = V_{ij} \mid C_k)^2 - \sum_{I}\sum_{j} P(A_i = V_{ij})^2}{K}$$

Where K is the number of clusters, V_{ij} is the jth value for attribute A_i and C_k is the kth cluster.

Clustering A	Clustering B	Weight	Shape	Colour
1	1	light	sphere	red
1	3	heavy	sphere	red
1	3	heavy	sphere	green
1	1	light	sphere	green
2	2	light	sphere	blue
2	2	heavy	cube	blue
2	2	heavy	cube	red
2	2	heavy	sphere	blue
2	2	light	cube	red

Compare the clustering schemes and decide which is preferred.

A2 For clustering A

$$\frac{\sum_{k=1}^{2} P(C_k)\sum_{i}\sum_{j} P(A_i = V_{ij} \mid C_k)^2 - \sum_{i}\sum_{j} P(A_i = V_{ij})^2}{K}$$

Table of probabilities.

Cluster	$P(C_k)$	$P(A_i = V_{ij}$□$C_k)$						$P(A_i = V_{ij})$					
		Weight		Shape		Colour		Weight		Shape		Colour	
C_1	4/9	L	1/2	S	1	R	1/2	L	4/9	S	2/3	R	4/9
		H	1/2	C	0	G	1/2						
C_2	5/9	L	2/5	S	2/5	R	2/5	H	5/9			G	2/9
		H	3/5	C	3/5	B	3/5			C	1/3	B	1/3

Substituting in the equation:

$$\frac{(4/9)\times[4\times(1/2)^2 + 1] + (5/9)\times 3[(2/5)^2 + (3/5)^2] - [2\times(4/9)^2 + (2/3)^2 + (5/9)^2 + (2/9)^2 + 2\times(1/3)^2]}{2}$$

= 0.1679 (rounded)

For clustering B

Table of probabilities.

Cluster	P(C_k)	P(A_i = V_{ij}□C_k)						P(A_i = V_{ij})					
		Weight		Shape		Colour		Weight		Shape		Colour	
C_1	2/9	L	1	S	1	R	1/2	L	4/9	S	2/3	R	4/9
		H	0	C	0	G	1/2						
C_2	5/9	L	2/5	S	2/5	R	2/5						
		H	3/5	C	3/5	B	3/5	H	5/9			G	2/9
										C	1/3	B	1/3
C_3	2/9	L	0	S	1	R	1/2						
		H	1	C	0	G	1/2						

Substituting in the equation:

$$\frac{2x(2/9)x[1+1+2x(1/2)^2]+(5/9)x3[(2/5)^2+(3/5)^2] - [2x(4/9)^2 + (2/3)^2 + (5/9)^2 + (2/9)^2 + 2x(1/3)^2]}{2}$$

= 0.2790 (rounded).

Clustering B offers the higher probability.

References

[1] Elman, J. L. Finding structure in time, *Cognitive science,* 14, 179-211, (1990).

[2] Hagan, M.T., Demuth, H.B. and Beale, M.,*Neural network design,* PWS publishing, Boston, MA. 19-6, (1996).

[3] Hebb, Donald, *Organisation of behaviour.* New York John Wiley & Sons, (1949).

[4] Haines, K. and Hecht-Nielsen, R. A BAM with increased information storage capacity. In *Proceedings of the IEEE International conference on Neural Networks,* (San Diego), pp.1-181-190, (1988).

[5] Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Science,* 79, pp. 2554-2558, (1982).

[6] LeCun, Y. Une procedure d'apprentissage pour reseau a seuil assymetrique, *Cognitiva,* vol 85 pp. 599-604 (1985)

[7] LiMin Fu *Neural networks in computer intelligence*, McGraw Hill, pp 151-154, 166, (1995).

[8] McEliece, R. J., Posner, E. C.,Rodemich, E. R. and Venkatesh, S. S. The capacity of Hopfield associative memory, *IEEE Transactions on Information Theory,* 33(4), pp. 461-482, (1987).

[9] Minsky, M. and Papert, S. *Perceptrons,* Cambridge Ma. MIT press, (1969).

[10] Parzen, E. On estimation of a probability density function and mode, *Ann . Math. Stat.,* 33, pp. 1065-1076, (1962).

[11] Pham, D.T. & Lui, X. Neural Networks for Identification Prediction and Control, Springer. pp 67-79 (1995).

[12] Rumelhart, D. Hinton, G. E. and Williams, R. J. Learning representations by back-propagating errors, *Nature,* vol. 323 pp 533-536, (1986)

[13] Specht, D.F. Probabalistic neural networks and the polynomial adaline as complementary techniques for classification. *IEEE Transactions on Neural Networks,* 1(1), pp. 111-121, (1990).

[14] Werbos, P.J. Beyond regression: New tools for Prediction and Analysis in the Behavioural Sciences. Ph. D. thesis, Harvard University.

Chapter 7.

PRACTICAL INPUT METHODS

Summary

When neural networks are moved from the realms of computer models to practical applications the inputs are obtained from sensors, for which effective function requires suitable signal strength and response speed. This brief chapter suggests some mechanisms; some practical examples of sensor use have been downloaded from the Internet.

7.1 Data representation

Biologically equivalent inputs are engineered translations of one or more of the five sensory functions associated with humans. The neural network inputs can extend the sensitivity beyond the range associated with humans.

The sensory signal may be discrete. The two state signal is described in a neural network in binary or bipolar terms. Practical circuits may achieve either discrete state in response to signal strength many ways.

Continuous input data can be voltages, currents, lengths, masses, times, chemical, optical, or thermal measures. They may be standardized, by some transformation to the range 0 to 1.

Sensor output signals are usually transformed to an electrical form for use in the network. Several electrical devices are included in a text by Chickoki (1994) that are applied in signal processing.

The following table gives some examples of sensors and the methods of signal transformation from personal experience.

Source	Signal transformation
Mechanical, linear or angular movements.	Stepping motors, solenoid displacement, pressure transducers.
Mechanical, fluidics.	Coanda effect devices, turbulence amplifiers.
Thermal, radiation.	Optical pyrometer colour matching, nested thermopile.
Thermal, convection and conduction.	Thermocouples, bi-metal strip movement, fluid movement.
Optical.	Light intensity, colour gradation.
Electrical.	Schmitt trigger, voltage and current transformers and impedance based devices.
Time.	Clock pulse, capacitor charge and discharge, divide and count circuits.

Sensors and signal sources.

Table 7.1

7.2 Scent

The equivalent of the olfactory organ is the neural nose, in which "sniffing" draws an air sample and determines the amount of a particular chemical present in the sample. This technique may have practical application in the detection of drugs, freshness of foods and air condition. Police dogs are thought to track certain criminals by their body odours and the neural nose was being reviewed for this purpose as reported in Chemistry in Britain July 1995.

Limitations of the system will be, the levels at which sensory detection may be made, the response time for detection and the time taken to produce a meaningful corrective action.

If for example an volatile petroleum derivative is present in the extraction line of a fuels plant it may prove a fire hazard at a certain level. The level at which detection must be made of the substance, is relative to the hazard level, the surrounding gases and it's current rate of increase.

Recently artificial noses have been developed that measure the amplitude, intensity, or quality of smell. The potential uses of these "electronic smell fingerprints", which can run continuously, range from food quality control to drug detection.

True simulation of the human nose is an impossible task, nature's version has some 10 million olfactory receptors of 30 different types, which work in parallel to absorb a range of odour molecules. When an odour molecule reaches receptors that match its particular pattern, the receptor via the nervous system sends a signal to the appropriate brain region. It is thought that the number and location of the engaged receptors is analysed, producing an odour image, which is compared to images within the brain's memory.

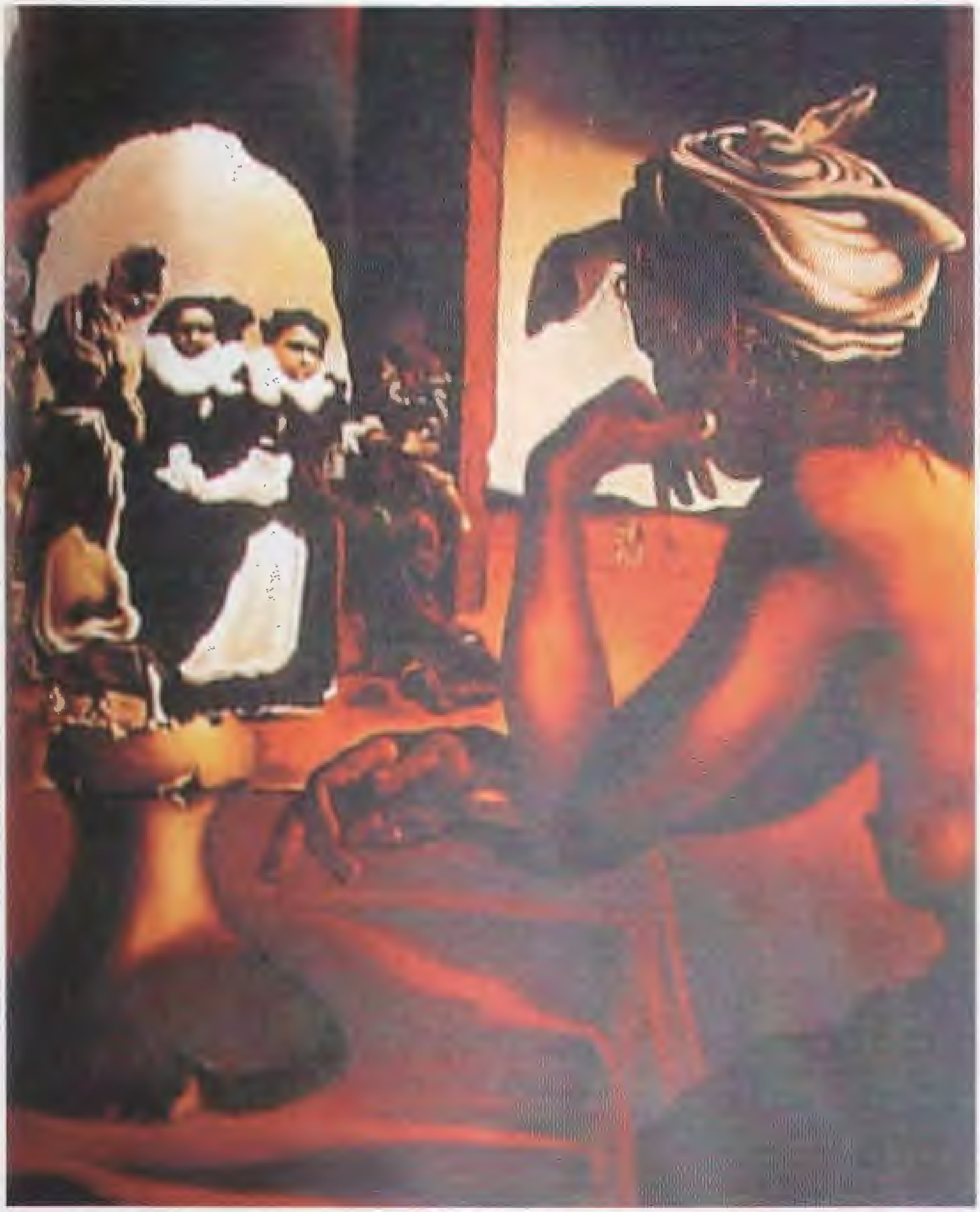
As early as 1990 a prototype olfactroscopy instrument, was produced which contained sensors based on the conducting polymers of pyrrole and aniline, which are sensitive to the characteristics of volatile chemicals producing the aroma. The sensors absorb then quickly desorb the volatiles at the polymer surface, causing temporary changes in their electrical resistance from which a digital output may be displayed which is specific to a particular odour.

7.3 Sight

The human eye is limited in it's range of visual detection to a portion of the spectrum of accessible wavelengths. Neural networks in brain simulation should offer the ability to detect beyond the range of normal human vision.

Limitations may be found when recognition is obscured by optical distortions, or if fuzziness occurs due to white light interference.

A well known painting by Salvador Dali (Slave market with the disappearing bust of Voltaire 1940) an illusion was created in which the brain perceives the face of the philosopher Voltaire or three figures against a light background. Gilling and Brightwell (1982), observe that when a machine is built that simulates the brains ability to respond to these illusions we will understand how human beings see.



7.3.1 A potential visual application in flour milling.

In the flour milling industry judgements of mill roll adjustments have been experience based. Observation of the millers behaviour showed the adjustments were made after passing the cupped hand under the lower cutting roll. The miller now compressed the drawn sample between both hands. He turns the hands over, examines the now inverted sample and adjusts the gap between the rolls.

The pattern of behaviour was studied so that the a methodical approach could be substituted for the miller's experience. A watch glass moved at a constant speed and fixed distance under the roller was used to gather a similar sample. A second watch class was pressed into the first and the pair inverted. Examination showed the sample to change colour in the direction of watch glass movement. The miller explained that the proportion of bran darkens the flour output. His adjustments being made to obtain uniform colour.

A neural network could provide the signals that would allow automatic adjustment to maintain uniform mill output. A set of expected output patterns could be gathered from the rolls, using pre-set increments of roll adjustment. Optical scanning for colour density from process sampling would provide the input signals. A supervised learning network could be trained, so that when a pattern match was recognised the appropriate output signal would be given.

7.3.2 Internet visual sensor examples.

The following two examples of sensors that have potential for neural network inputs have been downloaded from the internet. The request to the search engine was, sensory detection, electro-mechanical. Cases are given without modification, although my comment is given on the potential neural network technique.

Case 1

Environmental as well as economic factors are pushing forward the development of sensors and technologies for precision farming practice. Selective application of herbicides requires information on the location of weeds in the field. This information can be supplied as maps created and maintained by Geographical Information Systems (GIS) or acquired by sensors during application.

In this work, a sensor for automatic detection of weeds in the field was developed and tested. Visual detection of weeds requires discrimination between soil and plants, as well as discrimination between crop and weeds. The developed sensor was based on a multi-spectral imaging system in the range of 500-1000nm. An electronically tunable filter (Acousto-Optic Tunable Filter - AOTF) was coupled with a high resolution CCD camera and a frame grabber.

The developed sensor was mounted on a mobile platform and a large database of images was constructed by acquiring images of cotton plants and weeds in their early stage of development. Images were acquired from 500 to 1000nm in 5nm increments. Images were then analyzed and characteristic features of cotton plants and weeds were calculated. Weeds detection was based on spectral reflectance properties of their leaves and on shape analysis. (Alchanatis, Hetzroni, and Edan).

Comment :

This may be a case for data clustering, using a Kohonen network. The sensors have created a large number of images. An aerial release of suitable herbicides could be triggered by the trained network as the terrain scanned is fitted to a particular cluster.

Case 2

Three dimensional information has a major role in the commercial sorting and grading of fresh produce. The reliable extraction of such information for machine vision applications is usually accomplished through stereo vision or structured light. Stereo methods require multiple cameras or computationally intensive algorithms and are thus not suitable for a commercial system. As a simple application of structured light, a low-cost red diode laser with a line generator was used to project a line on apples, and the images were acquired through an array CCD camera. The images were then thresholded to extract the projected curve. The curve was then smoothed using a Savitzky-Golay filter and 'breaking points' were identified as local maxima of curvature.

Results show that it is possible to reliably identify a stem or a calyx using robust and simple criteria. For apple bruises the situation is far more difficult and at the current stage a detection system has to take into account spectral information for acceptable reliability. (Lev, Gofer).

Comment:

The observation of the need for "multiple cameras or computationally intensive algorithms" may indicate the possible use of sample data in a neural network. A recurrent network using error correction training could provide real time decisions in production.

Other sight dependent sensors include elevated temperature evaluation (radiation) by optical pyrometer and interferometry in precise physical measurement.

7.4 Taste

The artificial simulation of taste is extremely complex, for a particular problem such as wine tasting the many factors may include, sweetness, bitterness, acidity and a range of contaminants associated with the process. The latter may be associated with the cultivar, characteristics of the growing area, fertilizer chemicals, local water supplies and maturation casks.

The expected output patterns could be provided from wines graded by human tasters, the submitted wines being classified into a particular pattern. The benefits could be the elimination of false grading due to human error caused by age deterioration of taste sensitivity and masking by the presence of external pollutants.

The sensors may be spectrographic or of a similar nature to those used in the neural nose.

7.5 Touch

Touch sensitivity may be related to frequency and amplitude, as in vibrating strings, surface roughness, or machine vibrations. Touch may also be used in thermal detection, current or voltage probes, or motion sensing .

A typical vibration input might be from the movement of a tracer probe as is the case with the 'Rank Taylor Hobson Talysurf' surface roughness measuring machine. Transducers amplify surface irregularities to produce traces that may be used to detect deviations from expected surface finishes.

Neural networks may find a use in the determination of excessive process, or machine vibration that could cause premature failures.

Thermal conduction may be detected by thermocouples, or thermometers. Thermocouples produce continuous electrical signals which may be amplified for neural network use. Thermometers are unlikely to provide usable sensory signals for neural networks.

Vibration patterns arise from a wide variety of causes ranging from earth tremors to machine chatter. The use in neural networks may involve patterns generated by changing amplitude and frequency.

7.6 Sound

The vibrations transmitted through a fluid medium that are received by the human ear produce sounds. These sounds may be used industrially to detect potential troubles in process plant or machinery.

The primary considerations are frequency, amplitude, cut off length and the waveform.

The following two examples were downloaded, without modification, from the interrnet.

An acoustic impulse method was used for the non-destructive firmness evaluation of pineapple fruit of a wide range of internal maturity. Batches of pineapple fruit (cv. Cayenne lisse), from Ivory Coast, were harvested at 140, 143, 145 and 148 days after the floral induction treatment. Mechanical properties of pineapple fruit were also measured by using a texturometer.

Deformation, tensile and puncture tests were carried out on each pineapple respectively on the whole fruit, a standard shaped specimen of skin and on the core part of three slices of each fruit. The chemical composition of the entire fruit pulp was then analyzed to determine refractometry index, total sugar content and titratable acidity. Pineapple fruit spectrum displayed three main resonant peaks for the spherical mode. The resonant frequency of greatest amplitude and the resulting elasticity coefficient from 420 to 320 Hz and from 2 to 1.09 MPa respectively as the physiological development of the fruit increased.

Pineapple stored at 100C for a week exhibited lower resonant frequency and elasticity coefficient than references of the same maturity. Elasticity coefficient was found to be highly correlated to mechanical firmness (r=0.80) measured by deformation on individual whole fruits. The relationship was improved (r=0.91) by using mean values for the different fruit batches. Elasticity had a significant correlation with titratable acidity (r=-0.69) and refractometry index (r=-0.58), but was very poorly correlated to puncture core firmness (r=0.24)

Elasticity coefficient, obtained from acoustic measurement, may be considered as a promising way for non-destructive evaluation of global pineapple firmness (Valente, M. Duprat, F. Grotte, M)

Defense Research Technologies, Inc. (DRT) and the United States Department of Agriculture, Agricultural Research Service, entered into a Cooperative Research and Development Agreement (CRADA) several years ago to exploit the ultra-high sensitivity and low noise of acousto-fluidic microphonic sensors in the detection and quantification of insect pests in grain and stored products.

ARS has developed a technique for quantifying the number of insects (including internal feeding larvae) in a sample of grain called ALFID (Acoustic Location Fingerprinting Insect Detector). Individual insect sounds (from chewing, moving, etc.) are cross-correlated to determine relative times-of-arrival to multiple sensors.

This data is arranged in vectors, each indicative of a sound source location (akin to triangulation) and are compared and grouped to identify the number of sound locations (i.e., insect locations) in the sample. The effectiveness of this method has been limited due to the inadequacy of conventional microphones and the high cost of laboratory-grade-sensors.

Initial experiments with acousto-fluidic amplifiers that raise the acoustic amplitudes to high enough levels where very low cost electret microphones can readily process the sounds, have been extremely successful. We have been able to demonstrate a signal-to-noise ratio improvement that has resulted in an insect detection probability improvement of almost a factor of three. In fact, in three separate experiments with sample sizes of 64, more than half of the acousto-fluidic detections could not be discerned conventionally.

The three-stage acousto-fluidic amplifier, powered by a small aquarium pump and with an acoustic gain of 56dB and a bandwidth of 1 -7kHz, driving a pair Tibbetts 251-01 electret microphones, is able to detect and process sound levels of less than 0dB SPL (referenced to 0.0002 microbar). We have been able to reliably detect a two-week old rice weevil larva inside a grain of wheat, in a 1kg sample of grain, at a distance of over 10cm. Production cost of such a sensor will be on the order of $100, including the electronics and pneumatic power supply.

In the next year we are planning to fully instrument an ALFID system to demonstrate reliable insect quantification and hope to improve the confidence levels to better than 95-percent. Since inadvertent infested grain shipments (shipments that were thought to be clean based on visual insect detection) account for multi-million dollar annual losses it is anticipated that the acousto-fluidic version of ALFID system will greatly improve grain export cost effectiveness.

With future funding of the development of acousto-fluidic devices for the grain and stored products industry, monitoring of insect infestations in silos and packaged goods will be possible.

We also hope to be able to advance the technology of infestation quantification by using large arrays of multiplexed sensors coupled with ARS developed insect population algorithms.(Drzewiecki, T. M. Shuman, D.)

7.7 Multi-sensing

There may be cases of pattern recognition in which more than one sense is used.

Many modern performers in the arts world rely on optical and sound stimuli to impact on their audience, it may be that a suitable neural network could classify patterns in terms of the perceptions of different population sectors.

The following example is downloaded without modification from the internet.

During ageing, fruit and vegetables undergo a process of softening and loss of weight. A sensory system to monitor the softening of fruit in sealed commercial controlled atmosphere chambers would be a useful aid to making decisions on due times for marketing.

In recent years, it has been found that this softening process can be measured very accurately through the vibration resonance frequency of the fruit. On the basis of this discovery, a commercial system has been developed to monitor the firmness of fruit samples in CA storage from a remote point.

Each single monitor contains a flexible piezoelectric film connected to a central computer and mounted on top of the individual sample fruits during the entire storage period. The firmness of the sample is calculated from the resonance frequency of vibrations excited by gentle taps induced on the cheek of the samples. Operation of the system and the accumulation of data are fully automated.

Over the 1996-97 season, trials were conducted on four kinds of fruit: Red and Golden Delicious apples, Spadona pears, Triumph persimmons and Hayward kiwifruit. These trials were conducted in six commercial CA chambers in Kiryat Shmona, northern Israel. The monitors for pears were located in the center of the chamber and by the window, with and without polyethylene covering. Pears in the center of the chamber maintained their firmness (26FI - 30FI) for six months, whereas covered pears located by the window lost some ten units of firmness during the same period and uncovered pears by the window lost 15 - 20 units. Golden Delicious are sensitive to loss of moisture, a factor which directly affects their quality.

The remote sensing trial showed that this fruit maintained its firmness far longer when humidity in the chamber is kept high by misters. In Red Delicious, the results showed that the fruit maintained its firmness over a period of eight months' CA storage (with a slight rise during the first month) at 25 - 35 FI. Persimmons softened during storage, and the sensing system gave an accurate indication of the optimal time to open the chamber to market the fruit. In the case of kiwifruit, further research is required, over a longer period of storage.

When pears, Golden Delicious apples and persimmons are put into long-term storage at an unsuitable state of firmness, they do not maintain their quality. The ripeness monitoring system shows the state of the fruit in "real time", so that the fruit can be removed from storage while its quality is still high enough for the market.(Levin, A. Sandler, N. Carmi, Y. Shmulevich, I N. Galili, N.)

For neural network advances the hardware must be capable of replicating the massive parallelism that exists in the biological neuron. Some research is devoted to the production of processors that are tailored to performing the tasks of individual artificial neurons. An alternative is to package several fast RISC processors to a board, with hardware accelerators, to increase the parallelism of neural networks.

Fan out, the ability to drive other neurons, is provided by other accelerator boards for subsequent processing.

DACS homepage reports that accelerator board products are being developed both independently and by the makers of neural network development systems.

Examples

Q1 Suggest some practical input devices from which bipolar inputs may be achieved?

A1 A wide variety of answers may be given, two examples from my own background in the mechanical field are:

Fluidics wall attachment devices in which the Coanda effect results in a signal from one of two exits in the manner of the electrical JK flip-flop.

Go/not go gauging in which the decision of high or low sized components are discreet results.

Electrical equivalents are available in filtered negative and positive voltages.

Movement detection often results in continuous measurement. Use of fixed limit stops can turn the motion boundaries into positive and negative signal generators. This may find use in chemical, electrical and mechanical fields.

References

[1] Alchanatis, V. Hetzroni, A. Edan, Y. *A MULTISPECTRAL IMAGING SENSOR FOR SITE SPECIFIC APPLICATION OF CHEMICALS*, Institute of Agriculture Engineering, A.R.O., Volcani Center, Bet Dagan, Israel. Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Israel (1999).

[2] Lev, Z. Gofer, Z. *IDENTIFICATION OF FRUIT FEATURES USING STRUCTURED LIGHT*, Fruitonics, Ltd., Israel, (1999)

[3] Valente, M. Duprat, F. Grotte, M. Lasaygues, Ph. *NON-DESTRUCTIVE EVALUATION OF FIRMNESS OF FRESH PINEAPPLE BY ACOUSTIC METHOD*, Departement fruitier, CIRAD-FLHOR, Montpellier, France Laboratoire de Methodes Physiques d'Etudes, INRA, Avignon, France Laboratoire de Mecanique et d'Acoustique, CNRS, Marseille, France, (1999).

[4] Drzewiecki, T.M. Shuman, D. *ACOUSTO-FLUIDIC DETECTION OF WEEVIL LARVAE IN WHEAT*, Defense Research Technologies, Inc., Rockville, MD, USA Center for Medical, Agricultural and Veterinary Entomology, Gainesville, FL, USA, (1999).

[5] Levin, A. Sandler, N. Carmi, Y. Shmulevich, I.. Galili, N. *AUTOMATIC REMOTE MONITORING OF RIPENING IN CA CHAMBERS*, Israel Fruit Growers' Assoc., Post-Harvest Research Laboratory, Kiryat Shmona, Israel. Eshet Eilon, Agro-Industrial Systems, Kibbutz Eilon, Western Galilee, Israel. Faculty of Agricultural Engineering, Technion-Israel Institute of Technology, Haifa, Israel, (1999).

[6] Gilling, D. & Brightwell, R. *The Human Brain,* Orbis Publishing, London, (1982).

[7] Cichocki, A. and Unbehauen, R. *Neural networks for Optimization and Signal processing,* Wiley, Stuttgart (1994)

Chapter 8.

RECENT DEVELOPMENTS

Summary

The following examples were gleaned from the many cases explained on the Internet. Students or designers seeking ideas for neural network applications may find the inspiration needed through the same source.

8.1 General

The renewed interest in neural networks since the 1980's has attracted the attention of scientists in many fields. An examination of the topic on the Internet can reveal many applications and should provide ideas for many practical applications. On a domestic level a microwave oven has been developed that has been trained to respond to certain food cooking patterns.

Three practical examples from different spheres are given to illustrate the versatility of this technology.

8.2 Image segmentation

Closely related to the European Community funded AIR-project "Objective Plant Quality Measurement by Digital Image Processing", basic research is carried out to develop smart segmentation procedures. After reporting successful image segmentation of plants, using supervised trained neural networks (1), now a new successful unsupervised image segmentation procedure can be mentioned.

Two approaches are followed during the development: procedures, focusing on the centres of the clusters to distinguish; procedures, focusing on the separation of clusters by discovering areas with low occupation in an high dimensional environment.

The first approach did not result in qualitative good segmentation, even with extreme high investments in computing power. One of the experienced disadvantages of this approach, comparable to region growing techniques, is that weak connections between different clusters easily can result in fusing of clusters. The second approach however has shown to be successful. The segmentation of colour images (RGB) distinguishing leaves, flowers, stamen, pot and background, is performed in a 18 dimensional environment, representing the R, G and B values of each image point together with additional information from the direct environment to achieve separability of the clusters to distinguish.

At this point in research the method is restricted to linear separability of the clusters. Considering a two dimensional environment, clusters can be divided by lines, in a three dimensional environment by planes and in n dimensional environment by n-1 dimensional structures. Starting with the complete data set the first neural network represents a seventeen dimensional structure to divide the data set in two subsets.

By unsupervised training this structure is placed in the eighteen dimensional space through areas with low image point density. Each distinguished subset again is divided by an additional neural network: recursive partitioning. This results in a tree structure with at each branch a neural network.

Partitioning stops as soon as in a branch the separability criterion can't be fulfilled. After the unsupervised training the neural system can be used for the segmentation of images. (J. Meuleman, J. & van Kaam, C).

8.3 Insect detection

In agriculture, the detection and identification of insect pests is often carried out manually using trapping methods. Recent advances in signal processing and low power electronics have introduced the possibility of automatically identifying species that produce sound either as a communication signal (e.g. mating) or as a by-product of movement (e.g. flying) or eating. There has been much research into the acoustic detection of insects in stored produce and successful results have been obtained for a number of species. However, actual identification of species has received very little attention in entomology despite recent successful work on bird and amphibian species identification.

This paper is based on work carried out at Hull on the development of an automated system for the recognition of Orthoptera species (grasshoppers and crickets) which makes use of advanced signal processing techniques coupled with an artificial intelligence analyser based on an expert system or an artificial neural network.

The primary signal processing technique is called time-encoded speech (TES) which is computationally much less complex than frequency domain methods such as FFT's and acts as a preprocessor for the artifiicial intelligence analyser. Over 99% detection accuracy has been obtained with 7 species of British cricket, bush cricket and grasshopper using a neural network. The system is based on a personal computer and a low cost sound card thus allowing portability.

The paper will describe the principles of operation of the system and show results to date on Orthoptera and other orders. It will also discuss the potential for insect pest species identification and wider application for other orders, phyla and acoustic sources such as cavitation. (Chesmore E.D.)

References

[1] J. Meuleman, J. & van Kaam, C. European Community funded AIR-project Objective Plant Quality Measurement by Digital Image Processing. (1999).

[2] Chesmore E.D. Development of an automated system for the recognition of Orthoptera species. Hull University, (1999).

Chapter 9.

FUTURE TECHNOLOGY

Summary

Future advances in neural network technology will depend on the advances in hardware, as well as the advances in understanding of the way in which the brain functions. Considerable funding is available to researchers in the western world, as evidenced by defence industry investment.

9.1. Dedicated Neural Processors.

Dedicated neural processors are processors with specific capabilities that enable their use in neural networks. Several of the large chip manufacturers have developed neural processors, some of which are processors created specifically for the development system vendors. Modern technology has enabled packaging of a number of simplistic neurons onto a single chip. Some integrated circuits incorporate proprietary concepts, such as creating specific types of fuzzy neurons. Technologies range through a broad spectrum-analog, digital, optical, some of which are linked to create hybrids. No single type appears to have emerged as a clear winner at this time.

9.2 Developmental approaches

The vendors within the industry predict that migration from tools to applications will continue. In particular, the trend is to move toward hybrid systems. These systems will encompass other types of processes, such as fuzzy logic, expert systems, and kinetic algorithms. Indeed, several manufactures are working on "fuzzy neurons."

The greatest interest is on merging fuzzy logic with neural networks. Fuzzy logic incorporates the inexactness of life into mathematics. In life most pieces of data do not exactly fit into certain categories. For instance, a person is not just short or tall. He can be kinda short, pretty tall, a little above average, or very tall. Fuzzy logic takes these real-world variations into account. In potential application of neural networks, in systems that solve real problems, this fuzziness is a large part of the problem. In automating a car, to stop is not to slam on the brakes, to speed up is not to "burn rubber." To help neural networks accommodate this fuzziness of life, some researchers are developing fuzzy neurons. These neurons do not simply give yes/no answers; they provide a more fuzzy answer.

Systems built with fuzzy neurons may be initialized to what an expert thinks are the rules and the weights for a given application. This merging of expert systems and fuzzy logic with neural networks utilizes the strength of all three disciplines to provide a better system than either can provide themselves.

Expert systems have the problem that most experts don't exactly understand all of the nuances of an application and, therefore, are unable to clearly state rules that define the entire problem to someone else. The neural network doesn't care that the rules are not exact, for neural networks can then learn, and then correct, the expert's rules. It can add nodes for concepts that the expert might not understand. It can tailor the fuzzy logic that defines states like tall, short, fast, or slow. It can tweak itself until it can meet the user-identified state of being a workable tool. In short, hybrid systems are the future.

9.3 Hierarchal network application

A mobile robot whose behaviour is controlled by a structured hierarchical neural network the robot has four wheels that permit free movement using a drive motor and a steering motor. (Nagata, S. & Sekiguchi, M. & Asakawu, K.)

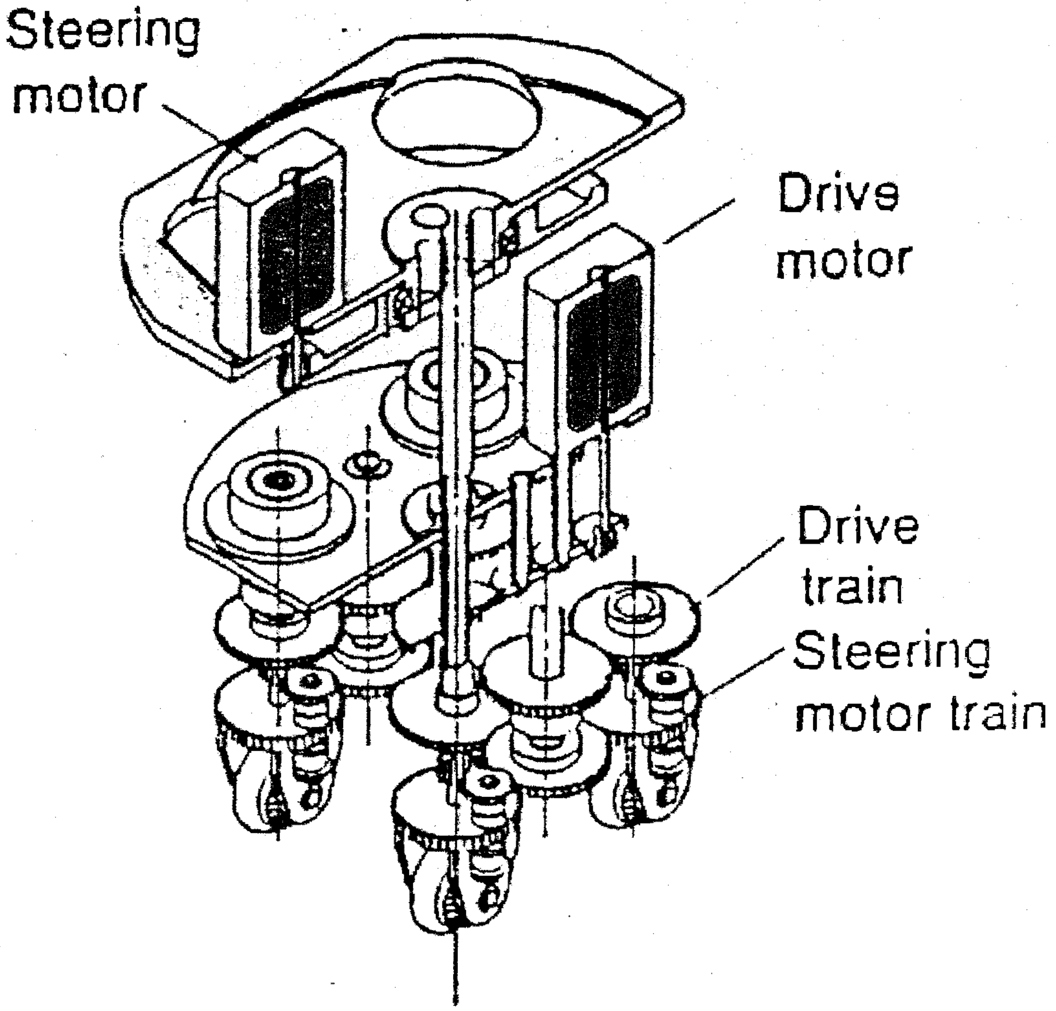
Twelve sensors monitor internal conditions and environmental changes that provide the network inputs, the outputs being used as motor control signals.

The network is divided into two sub networks connected to each other by short-term memory units that are used to process time dependent data.

Robot control involves complex motion control, with its associated sensor signal processing, which is not suited to the sequential processing of von Neumann type computers.
Mobile robot must be flexible in their learning and adaptation capabilities from their sensors, so that they may respond to environment changes in real time.

The conventional von Neumann would be programmed in advance so that environment changes detected by the sensors would be referred to the higher-level decision maker. By contrast, the neuro-computer would be flexible to unexpected situations by using learning and interpolation capabilities.

The robot was designed to move freely in confined spaces and sense environmental changes.



Locomotion mechanism.

Example

A typical question for this section might include an internet search or text search for topical data.

References

[1] Meuleman, J. & van Kaam, C. UNSUPERVISED IMAGE SEGMENTATION WITH NEURAL NETWORKS, Wageningen Agricultural University, Wageningen, The Netherlands, (1999).

[2] Chesmore, E.D. ACOUSTIC METHODS FOR THE AUTOMATED DETECTION AND IDENTIFICATION OF INSECTS. Environmental Electronics Research Group, School of Engineering and Computing, University of Hull, Hull, United Kingdom, (1999).

[3] DAC Home pages Internet

[4] A mobile robot controlled by a structured hierarchical neural network (Nagata, S. & Sekiguchi, M. & Asakawu, K. IEEE Control systems Magazine, pp 69-76, (April 1990).

Chapter 10.

NEURAL NETWORK TYPES

Summary

The following descriptions of some networks and related applications have been acquired from a wide variety of published texts.

ART 1

Adaptive resonance theory initial version for processing binary input patterns only. This accepts an input vector and classifies it, by similarity, into one of a number of stored patterns. Patterns that are distant from the stored patterns lead to the creation of new grandmother cells. Carpenter, and Grossberg, (1986).

ART 2

Adaptive resonance theory second version for processing real input patterns. It is a development that can classify binary and continuous outputs. Carpenter, and Grossberg, (1987).

ART 3

Adaptive resonance theory improvements on the second version offering greater stability for processing real input patterns. Carpenter, and Grossberg, (1990).

BAM

Bi-directional associative memories, the activation resonates between two sets of processing elements until a stable state is reached. Kosko, (1988).

Boltzmann machine

A distributed parallel processing algorithm, based on statistical mechanics, stable solutions being found from simulated annealing.(Ackley, Hinton, and Sejnowski, (1985).

BSB (Brain state in a box model)

A simple associative network coupled to non linear dynamics, it is unable to learn difficult discriminations. A group of neurons feeds back on itself. Anderson, Silverstein, Ritz, and Jones, (1977).

Counter-propagation network

The inputs feed a Kohonen layer to an outstar layer, when the network is connected it accepts inputs and generates outputs simultaneously, thus allowing counterflow through the system.

Feed-forward network

A network in which flow is unidirectional from inputs to outputs.

GMDH (Group method data handling)

A network that builds solution equations, (see Chapter 6)

GRNN (General regression neural network).

A three layer network having one hidden neuron for each training pattern, these networks are used to train quickly with sparse data sets. Useful for continuous function approximation, reported by Ward Systems to be superior to backpropagation networks for many types of problems, Specht, (1991).

Hopfield networks

Single layer attractor networks in a wide variety of designs. Useful for auto-association and optimization tasks, the concept is that a single network of interconnected binary valued neurons can store multiple stable states.

Kohonen self organising network

The network differentiates into multiple regions, each of which is responsive to a specific stimulus pattern.

PDM (polynomial discriminant method)

Based on the adaline this network uses a polynomial surface to categorise input patterns.

PNN (probabalistic neural network)

A neural network in which weights on the key layers are established according to probability based decision theory. A three layer network with as many hidden layer neurons as there are training patterns and one output neuron for each possible category. Specht, (1991).

References

[1] Carpenter, G. & Grossberg, S. Neural dynamics of category learning and recognition: Attention, memory consolidation and amnesia. In Brain Structure , Learning and Memory (AAAS Symposium series) 1986.

[2] Carpenter, G. & Grossberg, S. ART 2 Self organisation of stable category recognition codes for analog input patterns. (Applied optics 26 (23): 4919-30. 1987).

[3] Carpenter, G. & Grossberg, S. 1990. ART 3 Hierarchical search using chemical transmitters in self organizing pattern recognition architectures. Neural networks 3 129-152.

[4] Kosko, B. Bidirectional associative memories. IEEE Trans. Systems, Man, Cybernetics. SMC-L8 49-60 January/February 1988.

[5] Ackley, D.H. Hinton, G.E. & Sejnowski, T.J.1985 A learning algorithm for Boltzmann machines. Cognitive Science, 9, 147-169

[6] Anderson, J.A. Silverstein, J.W. Ritz, S.A. & Jones, R.S. 1977 Distinctive features, categorical perception and probability learning. Some applications of a neural model. Psychological Review 84, 413-451.

[7] Specht, D. & Shapiro, P. Generalization Accuracy of Probabalistic Neural Networks compared with Back-propagation Networks. Proceedings of the Joint Conference on Neural Networks July 8-12 1991, 1, 887-892

[8] Specht, D. A General Regression Neural Network. IEEE Transcript on Neural Networks, November 1991, 2, 6, 568-576.

Appendix A

Portfolio selection

An initial study intended to forecast a stock portfolio based on the Johannesburg Stock Exchange was well advanced, however the erratic behaviour of our exchange in this evolving democracy led to the project being abandoned.

The factors used and the techniques applied may however prove useful at some future date so they are given as an appendix to this thesis.

A.1 Spreadsheet construction

For ease of interpretation into the commercial statistical and neural network programs the Excel spreadsheet programme has been used

Portfolio selection has been limited to part of the industrial sector to make a feasible database for this work. Elements chosen include Electronics & electrical, Beverages hotels & leisure, Building & construction, Engineering, Chemicals, oils & plastics, Packaging & printing, Food, Furniture & Household, Media and Pharmaceuticals.

A.2 Input selection

The inputs have been selected, or derived, from those that are freely available in South Africa from the following publications

The ABSA Bank Investors' Guide which is published quarterly.

The JSE handbook, which is, published half yearly.

The financial mail reports.

To avoid negative values in the neural network models, an artificial base has been applied as the ratio of value to the lowest value by sector. In this manner an across sectors portfolio may be constructed. The following example used a GMDH network, with some selected inputs to produce an alternative to regression methods for evaluating shares.

The raw data pattern is indicated by the following table abstract, 113 patterns of the patterns were used as input patterns. Of the chosen patterns 91 were used for training and 22 for test.

JSE Code	Company	Sector	CA/CL	R on CE	R on Equity	PTP/T	Debt to Equity	NAV/M pr	Liquidity	P E ratio
ABI	Amalg beverage Ind	BH&L	1.6068	25	25	10.8	0.9	29.2	2.12	16.5
DSL	Distillers corporation	BH&L	2.9266	22.8	14	14	0	64.3	0.75	13.7
ITL	Interleisure	BH&L	0.6857	37.4	35.1	16.1	17.1	21.6	1.98	19.3
KER	Kersaf Investment	BH&L	0.9357	18.5	16.8	25.2	17.3	59.3	3.65	13.2
SAB	SA Breweries	BH&L	1.2809	23.7	33.4	9.2	53.5	17.5	8.65	20
SFW	Stellenbosch farmers	BH&L	2.7313	13.7	8.5	7.4	3.2	121	2.74	13.1
SIS	Sun International SA	BH&L	0.2734	15.6	18.1	18.5	21.2	55.3	15.44	7.6
SPU	Spur steak ranches	BH&L	1.4375	142.1	106	45.4	2.4	8.5	9.21	12.9
SRG	Servgro International	BH&L	1.2495	28	22.6	13	26.2	59	8.42	24.9
SUN	Suncrush	BH&L	0.9195	25.2	21.7	19.9	7.3	45	15.86	11.7
AAL	Alpha	B&C	1.4036	17.6	14.7	22.3	8.2	40.1	6.07	9.5

The following table shows the approach used to exponentially weight data.

Although this was written for several stock exchange categories only the food sector with a 4 year record is given

JSE Code	Company	Sector	CA/CL	CA/CL	CA/CL	CA/CL	wt a4	wt a3	wt a2	R on CE	R on CE	R on CE	R on CE
	Retro years						0.	0.666	0.833				
BRL	Brenner Mills	Food	2.258	2.641	3.946	4.843	1.1290	1.76074	3.2888	47.	37.	24.	2
CSF	C&G Smith Foods	Food	1.376	1.418	1.392	1.422	0.6883	0.94567	1.16007	17.	14.	15.	20.
CAS	Cadbury Schweppes	Food	1.0	1.114	1.316	1.386	0.52	0.7428	1.09716	20.	22.	23.	24.
COI	Choice H	Food	0.719	1.016	1.041	0.954	0.3597	0.67773	0.86816	-4.	7.	14.	2.
CKS	Crookes Brothers	Food	1.495	0.59	0.93	2.70	0.747	0.39533	0.78249	6.	5.	1	11.
DLF	Del Monte Royal	Food	1.205	1.201	1.195	1.048	0.602	0.80120	0.99616	10.	6.	6.	4.
FDC	Foodcorp	Food	1.192	1.388	1.402	1.741	0.5964	0.92567	1.16882	10.	1	16.	16.
HLH	Hunt Leuchars &Hepb	Food	1.341	1.264	1.079	1.090	0.6707	0.84327	0.8993	7.	6.	1.	0.
ICS	ICS Holdings	Food	1.088	1.128	1.05	1.068	0.544	0.75260	0.87749	8.	13.	18.	2
ILV	Illovo Sugar	Food	1.170	1.150	1.301	1.296	0.585	0.76693	1.08482	11.	6.	7.	14.
IRV	Irvin & Johnson	Food	2.300	2.556	2.566	2.29	1.150	1.70440	2.13865	1	11.	1	10.
LGB	Langeberg Holdings	Food	2.826	2.982	2.694	3.005	1.4134	1.98807	2.24574	23.	16.	18.	23.
MDM	Macadams Bakery H	Food	1.54	1.61	1.821	2.286	0.774	1.07533	1.51816		11.	13.	3
NFH	Namibian Fishing	Food	1.60	1.358	1.256	1.024	0.802	0.90593	1.04741	7.	7.	14.	4.
NMS	Namibian Sea Product	Food	0.901	1.132	1.129	1.004	0.4508	0.75473	0.94149	6.	17.	16.	1
NRK	Northem Bakeries	Food	1.768	1.1	0.917	1.162	0.8842	0.7466	0.7648	34.	1	-1.	8.
OCF	Oceana Fishing Group	Food	2.637	1.840	3.515	19.038	1.3189	1.22700	2.92990	37.	44.	53.	49.
PML	Premier Group	Food	1.227	1.172	1.034	1.000	0.613	0.78193	0.86199	18.	19.	18.	15.
RBW	Rainbow chickens	Food	0.599	0.6	0.755	1.405	0.2996	0.44000	0.62991	-6.	1.	2.	-14.
TIG	Tiger Oats	Food	1.498	1.544	1.418	1.545	0.7491	1.02953	1.18207	20.	17.	1	21.
TNT	Tongaat Hulett Group	Food	1.813	1.6	1.705	1.958	0.9069	1.12000	1.42149	7.	8.	13.	16.
WBH	W B Holdings	Food	1.482	0.166	1.274	1.007	0.7412	0.11120	1.06232	12.	-8.	13.	5.

A.3 Actual inputs

The selection of inputs that have a high probability of revealing the real state of the stock, in terms of its potential movement and estimated current value is critical to forecasting success. Too many inputs are better than too few as those shown to be of small impact may be eliminated during model assessment. The downside of a high input volume is the speed of processing that may be achieved.

The portfolio selection will be a function of knowledge of the stock market state. Freely available inputs are from the JSE.

Buy sell or hold decision for the acquired portfolio is based on weekly data from the "Financial mail", daily data is thought to be too noisy for practical use by the small investor.

Some specialist inputs, that may be discovered, include company development strategies, staff to staff turnover ratio, export business proportions, resource investment, loss analysis in labour and materials and world trends for that industry. These have not been included in this study, as they require personal research for the selected stock.

Portfolio selection inputs (These are for the past 4 years, weighted.)
Current asset ratio = current assets/current liabilities (Considered to be reasonable when the ratio >3/2) This reflects the ability to repay short term debt and may be more valuable if stock could be excluded from assets, as they are not easily realized.

Return on capital employed % is Pretax profit/Capital employed x 100. This is a direct measure of profitability.
Return on equity % is Profit after taxation/ Average ordinary shareholders interest x 100.

Cash flow per share % is Net cash generated/ Weighted number of ordinary shares issued x 100. This would reflect the desirability of the stock to investors, if achieved without undue risks to the enterprise.

Debt to equity % is Total interest bearing loans/ Total owners interest x 100. The indicator of how well debt is covered by equity, it may be an indicator of the ability to borrow for expansion, or new projects.

Effective tax rate % is Current taxation/ Net income before taxation x 100.

NAV/Market price % is Net asset value per share/ Market price per share x 100. Indicates whether the share is correctly priced.

Market capitalization (ordinary shares x current share price.)
Log (market capital at year-end) is Log (Number of shares issued x share price at the year-end). This is a size measure of a company for which logarithms have been taken, because of the large range of values.

Profit margin % is Pretax income/ Turnover x 100

Stock decision criteria inputs (4 week weighted decisions are used)
These are from the Financial Mail data. To simplify the inputs, in some cases, inputs are given weekly data to annual data ratios.
Week advance/(12 month high - low) x 100 (This simplifies the effect of change with respect to the year's change level, I.e. how important is the weekly change with respect to the year's change)
Market capital/Volume of shares traded (This factor stabilizes the effect of company size.
Dividend yield per share
Earnings per share 4 quarters /Earnings per share financial year (Recent share earnings decline or growth should be revealed in this measurement).

A.4 Importance of inputs

In the Neuroshell 2 programme the GMDH network may assist in studying the most important inputs. It performs a continuous evaluation of the inputs and excludes those it interprets as having little influence on the prediction of outputs.

An alternative screening procedure recommended by Ward Systems to use the smoothing effect of the GRNN and PNN neural networks. From the final best model the smoothing factors rank variable importance on the scale 0 to 3, where 0 is little or no importance and 3 is maximum importance.

Input screening techniques decrease in accuracy with increasing numbers of inputs. It is suggested that for large numbers of inputs they are screened in sets of 20 with the winners being retained for the final network. In the event that two or more non-important inputs, in different sets, may be combined to be important we may re-examine the initial discards.

Appendix B

MATHEMATICS FOR NEURAL NETWORKS

This appendix is in the nature of minimum requirements for understanding published text on neural networks. It gives some of the methods, abstracted from a large body of linear algebra knowledge, that may be needed for processing neural network data,.

B.1 Binary data representation

The representation of data in computation has two extreme possibilities, the grandmother cell concept in which a single neuron of a group represents the inputs and outputs of an element of a data set, or distributed representation in which the element is identified by a pattern of activated neurons. The pattern of grandmother cells representing any numbers 1 to 7 would be:

```
(1)   + - - - - - -
(2)   - + - - - - -
(3)   - - + - - - -
(4)   - - - + - - -
(5)   - - - - + - -
(6)   - - - - - + -
(7)   - - - - - - +
```

A pattern for distributed representation can be achieved using less neurons, e.g.:

1	+	-	-
2	-	+	-
3	+	+	-
4	-	-	+
5	+	-	+
6	-	+	+
7	+	+	+

In pattern recognition the grandmother cell approach gives conceptual clarity; when grandmother is recognized a single cell is activated. Distributed representation on the other hand may use some common activation for two different patterns. For numbers 5 and 7, neurons 1 and 3 would be activated. It is thought that biological functions lie between these extreme boundaries.

B.2 Vectors

Computations concerned with inputs to artificial neural networks involve ordered number sets called vectors. For a given number n of input elements we may use the designation a to describe a vector of elements.

$$\mathbf{a} = \begin{bmatrix} i_1 \\ i_2 \\ \cdot \\ \cdot \\ \cdot \\ i_n \end{bmatrix}$$

A vector is normally given in column form, transposition turns it into a row vector.

$$\mathbf{a} = \begin{bmatrix} 3 \\ 2 \\ -4 \\ -1 \\ 2 \end{bmatrix} \qquad \mathbf{a}^T = (3, 2, -4, -1, 2)$$

B.3 Neural network required vector operations in "R^n"

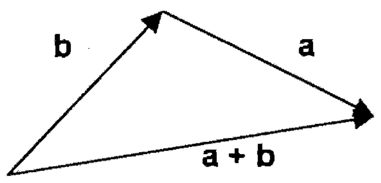
The basic operations are those of addition, subtraction, multiplication by a scalar and multiplication.

B.3.1 Addition and subtraction

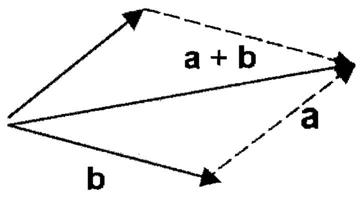
To add two vectors we add the corresponding terms

$$\begin{bmatrix} 3 \\ -4 \\ 2 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix} = \begin{bmatrix} 4 \\ -2 \\ 7 \end{bmatrix}$$

Drawing the 1st vector **b**, then starting the 2nd vector **a**, from the end of the 1st vector may illustrate the graphical addition of two vectors. The line joining the start of the 1st vector to the end of the 2nd vector is the sum vector **a** + **b**.



The same result may be achieved by the parallelogram method



Vector addition is commutative; it is independent of the order of addition.

a + b = b + a.

To subtract vector **b** from vector **a** we subtract the corresponding terms

$$\begin{bmatrix} 3 \\ -4 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix} = \begin{bmatrix} 2 \\ -6 \\ -3 \end{bmatrix}$$

To show graphically the subtraction of vectors, we add the negative of **b** to **a**.

These procedures are easily accomplished within a spreadsheet.

B.3.2 Scalar multiplication

For vectors **a** and **b** and scalars k, k':

$k(\mathbf{a} + \mathbf{b}) = k\mathbf{a} + k\mathbf{b}$.

$(k + k')\mathbf{a} = k\mathbf{a} + k'\mathbf{a}$.

$(k\,k')\mathbf{a} = k(k'\mathbf{a})$.

B.3.3 The inner product

For the inner product, or dot product, or scalar product of two vectors, they are aligned and their corresponding elements are multiplied together, then the resulting products are summed. At the completion of this operation we have a single number, which is a scalar.

Example

Given $\mathbf{a}^T = (5, 4, 1)$ and $\mathbf{b}=(3, 6, 2)$

then $\mathbf{ab} =(5)(3)+(4)(6)+(1)(2) = 15 + 24 + 2 = 41$.

The inner product operation applies only to vectors having the same number of elements. A practical application of the inner product is the primary stage of generic network computing in which we relate the input pattern with the connection strengths.

The scalar product of two vectors is the product of their lengths and the cosine of the angle between them.

$\mathbf{ab} = |\mathbf{a}|\,|\mathbf{b}|\cos\theta$ where = $|\mathbf{a}|$ is the vector length

A three-component vector represents a line in three-dimensional space, four or more component vectors are not visualized but may be thought of as line segments in hyperspace.

B.3.4 Length and distance in R^n

For vectors $\mathbf{a}^T = (a_1, a_2, \ldots, a_n)$ and $\mathbf{b}^T = (b_1, b_2, \ldots, b_n)$ the distance between points **a** and **b** is given by:

$\text{dist}(\mathbf{a},\mathbf{b}) = \sqrt{[(a_1-b_1)^2 + (a_2-b_2)^2 + \ldots + (a_n-b_n)^2]}$.

Vector length (norm) is defined as the non-negative root of **a.a**

$|\mathbf{a}| = \sqrt{\mathbf{a}.\mathbf{a}} = \sqrt{[a_1^2 + a_2^2 + \ldots + a_n^2]}$.

B.3.5 Normalizing

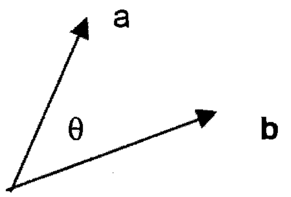
The dot product operation applies only to vectors having the same number of elements. A practical application of the inner product is the primary stage of generic network computing, in which we relate the input pattern with the connection strengths.

Geometric vector representations may help to visualise certain problems. A two element vector represents a point (a, b) on the x y plane, shown as a line joining the origin to this point.

In certain circumstances it may be convenient to make vectors a unit length by a process called normalization. Compute the length of a non-zero vector, then divide each component by that length.

B.3.6 Angle between vectors

Consider a plane on which are two different vectors **a** and **b** having the same origin, as shown.



The vector lengths of the sides of the triangle are **a**, **b** and **a** - **b**. From the Cauchy-Schwarz inequality for any vectors $\mathbf{a}, \mathbf{b} \in \mathbf{R}^n$

$\mathbf{ab} \leq |\mathbf{a}|\ |\mathbf{b}|$

The angle between vectors is $\theta = \cos^{-1} \frac{\mathbf{ab}}{|\mathbf{a}|\ |\mathbf{b}|}$.

B.3.7 Orthogonality

Vectors **a** and **b** are said to be orthogonal, (perpendicular) if their inner product is zero. I.e. if **ab** = 0.

B.3.8 The outer product

Of interest in neural networks is the outer product (cross product) of two vectors. Whereas the result of the inner product is a scalar the result of the outer product is a vector.

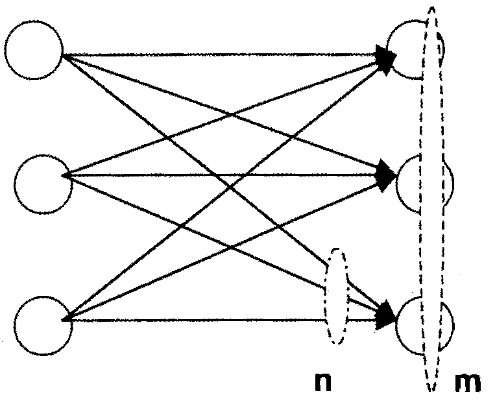
$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$\mathbf{a} \times \mathbf{b} = (a_2 b_3 - a_3 b_2)\mathbf{i} + (a_3 b_1 - a_1 b_3)\mathbf{j} + (a_1 b_2 - a_2 b_1)\mathbf{k}$

where **i**, **j**, and **k** are mutually perpendicular unit vectors.

B.4 Matrix operations for neural networks

A group of n weighted connections to a neuron, will be in vector form, for m neurons the result will be a matrix of connection weights. of dimension mn.



B.4.1 Addition and scalar multiplication of matrices

For two matrices **A** and **B** of the same dimensions add their corresponding elements.

Examples

$$\mathbf{A} = \begin{bmatrix} 4 & -3 \\ -5 & -1 \\ 1 & 0 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} -3 & 2 \\ 4 & -6 \\ 2 & 3 \end{bmatrix}$$

$$\mathbf{A} + \mathbf{B} = \begin{bmatrix} 1 & -1 \\ -1 & -7 \\ 3 & 3 \end{bmatrix}$$

For the above matrix **A** multiplication by a scalar k = 3 yields:

Example

$$3\mathbf{A} = \begin{bmatrix} 12 & -9 \\ -15 & -3 \\ 3 & 0 \end{bmatrix}$$

B.4.2 Multiplication of matrices

For matrices in which the number of columns in **A** is equal to the number of rows in **B**, i.e. **A** is an m x p matrix and **B** is a p x n matrix, then the product **AB** is an m x n matrix.

Product **AB** is not defined if **A** is an m x p matrix and **B** is a q x n matrix, where $p \neq q$.

$$\mathbf{AB} = \begin{bmatrix} 2 & -3 \\ 3 & 1 \\ -4 & 2 \end{bmatrix} \begin{bmatrix} 5 & -1 \\ -2 & 1 \end{bmatrix} = \begin{bmatrix} (2)(5) + (-3)(-2) & (2)(-1) + (-3)(1) \\ (3)(5) + (1)(-2) & (3)(-1) + (1)(1) \\ (-4)(5) + (2)(-2) & (-4)(-1) + (2)(1) \end{bmatrix}$$

$$\mathbf{AB} = \begin{bmatrix} 16 & -5 \\ 13 & -2 \\ -24 & 6 \end{bmatrix}$$

B.5 Eigenvalues and Eigenvectors

If we think of a vector as a geometric direction in n-dimensional space, multiplication of that vector by a matrix redirects the vector in another direction.

Some vectors have the special property that multiplication by a matrix results in the product vector pointing in the same direction as the original vector.

Since the new vector points in the same direction as the original it has a length that is a constant that can be a positive or negative multiple of the original length. The new vector lies somewhere along the line defined by x and the origin. The constants are called the eigenvalues or characteristic values, whilst the vectors are called the eigenvectors of the matrix.

This relationship is expressed formally as: $\mathbf{Ax} = \lambda\mathbf{x}$

Calculation is simplified by the use of eigenvectors, instead of performing the many operations required to multiply a matrix and a vector, we perform one multiplication, a vector times a constant. Eigenvectors sometimes may offer meaningful interpretations of the system the matrix describes.

An example with a simple geometric interpretation is the multiplication of a vector **f** with two components x and y by the matrix **A**

$$\mathbf{Af} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} y \\ x \end{bmatrix}$$

This matrix has interchanged the x and y co-ordinates this reflects the vector around the 45 degree line through the origin.

$$\mathbf{Ax}_a = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ x \end{bmatrix} = 1 . \begin{bmatrix} x \\ x \end{bmatrix} = \lambda \ \mathbf{x}_a \text{ with eigenvalue } 1$$

$$\mathbf{Ax}_b = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ -x \end{bmatrix} = \begin{bmatrix} -x \\ x \end{bmatrix} = -1 \begin{bmatrix} x \\ -x \end{bmatrix} = \lambda \mathbf{x}_2 \text{ with eigenvalue } -1$$

B.6 Lyapunov function

The stability of a network can be achieved if a function is found that decreases with each network change of state, until no further change occurs, at which state the function stops. The Lyapunov function behaves in this manner on some recurrent networks.

Lyapunov's stability theorem

If a positive definite function V ($\mathbf{a}$) can be found such that dV($\mathbf{a}$)/dt is negative semi definite, then the origin ($\mathbf{a} = \mathbf{0}$) is stable for the system: d($\mathbf{a}$)/dt = $\mathbf{g}$($\mathbf{a}$).

If a positive definite function V ($\mathbf{a}$) can be found such that dV($\mathbf{a}$)/dt is negative definite, then the origin ($\mathbf{a} = \mathbf{0}$) is asymptotically stable for the system.

Conceptually Lyapunov's method is based on the systems energy and the relationship of the stored energy to the system stability.

Hagan, Demuth, and Beale (1996) give the following excellent illustration of the Lyapunov stability theorem.

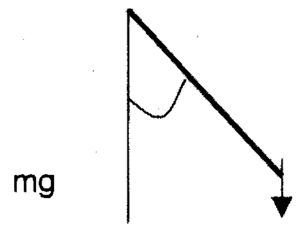


Figure 3.1

For a simple pendulum of mass m, length l, inclined at an angle θ to the vertical with damping coefficient c and gravitational force g. From Newton's second law.

$ml\, d^2\theta/dt^2 = -c d\theta/dt - mg\sin(\theta).$ (1)

$ml\, d^2\theta/dt^2 + c d\theta/dt + mg\sin(\theta) = 0$ the second term in the equation is the damping force, the third term is the gravitational force. (2)

Writing $a_1=\theta$ and $a_2=d\theta/dt=da_1/dt$ whence $d^2\theta/dt^2=da_2/dt$. (3)

$$\frac{da_2}{dt} = \frac{-g\sin(a_1)}{l} - \frac{c\, a_2}{ml}. \quad (4)$$

Now we want to investigate the stability of the origin (a = 0) for this pendulum system. (The origin corresponds to a pendulum angle of zero and a pendulum velocity of zero.) We first want to check that the origin is an equilibrium point. We do this by substituting a = 0 into the state equations.

$$\frac{da_1}{dt} = a_2 = 0. \quad (5)$$

$$\frac{da_2}{dt} = \frac{-g\sin(a_1)}{l} - \frac{c\, a_2}{ml}. \quad (6)$$

$$\frac{da_2}{dt} = \frac{-g\sin(0)}{l} - \frac{c(0)}{ml} = 0. \quad (7)$$

Since the derivatives are zero, the origin is an equilibrium point.

Using the total energy of the system as the Lyapunov function V including the kinetic and potential energies.

$$V(\mathbf{a}) = \tfrac{1}{2} ml^2 (a_2)^2 + mgl(1 - \cos(a_1)). \quad (8)$$

In order to test the stability of the system, we need to evaluate the derivative of V with respect to time.

$$\frac{dV(\mathbf{a})}{dt} = \left[\frac{\partial V}{\partial a_1}\right] \frac{da_1}{dt} + \left[\frac{\partial V}{\partial a_2}\right] \frac{da_2}{dt}. \quad (9)$$

The partial derivatives are found from the preceding V(**a**) equation.

$$\frac{dV(\mathbf{a})}{dt} = (mgl\ \sin(a_1))a_2 + (ml^2\ a_2)\ (-\frac{g}{l}\sin(a_1))(-\frac{c}{ml}\ a_2). \qquad (10)$$

After cancelling, $\frac{dV(\mathbf{a})}{dt} = -cl\ (a_2)^2 \leq 0.$ (11)

In order to prove that the origin ($\mathbf{a} = \mathbf{0}$) is asymptotically stable, we must show that this derivative is negative definite. The derivative is zero at the origin, but it also is zero for any value of a_1 , as long as $a_2 = 0$. Thus,
$\frac{dV(\mathbf{a})}{dt}$ is negative semidefinite, rather than negative definite.
From Lyapunov's theorem, then, we know that the origin is a stable point. However, we cannot say, from the theorem and this Lyapunov function, that the origin is asymptotically stable.

Common sense tells us that, due to friction, the pendulum will eventually settle in a vertical position, therefore the origin is asymptotically stable. The Lyapunov theorem, using our Lyapunov function, can only tell us that the origin is stable.

Hagan, Demuth, and Beale (1996) show that the origin is asymptotically stable, using LaSalle's Invariance Theorem.

The following is a numerical example for the pendulum, based on the work of Hagan, Demuth, and Beale. (1996)

Given g=9.8, m=1.5, l=2.5, c = 2.0.
The state equations for the pendulum are:
$\frac{da_1}{dt} = a_2$,

$\frac{da_2}{dt} = \frac{-9.8}{2.5}\sin(a_1) - \frac{2\ a_2}{(1.5)(2.5)} \quad = -3.92\sin(a_1) - 0.5333a_2$

Expressions for V and its derivative follow:

$V(\mathbf{a}) = (½)(1.5)(2.5^{2)}(a_2)^2 + (1.5)(9.8)(2.5)(1-\cos(a_1))$
$\quad = 4.685(a_2)^2 + 36.75(1-\cos(a_1))$

From (11) $\frac{dV}{dt} = -5\,(a_2)^2$

Note that $\frac{dV}{dt}$ is zero for any value of a_1 as long as $a_2 = 0$.

Examples

These are a few simple examples for revision purposes, more challenging examples are easily formulated from current UNISA curricula.

Q1 Add and subtract the following pairs of vectors:

$$\text{a)} \begin{bmatrix} 8 \\ 4 \\ -2 \end{bmatrix} \begin{bmatrix} -3 \\ 5 \\ 7 \end{bmatrix} \qquad \text{b)} \begin{bmatrix} -1 \\ 2 \\ -4 \end{bmatrix} \quad \begin{bmatrix} -6 \\ 8 \\ -1.5 \end{bmatrix}$$

A1 a) Addition $\begin{bmatrix} 5 \\ 9 \\ 5 \end{bmatrix}$ b) $\begin{bmatrix} -7 \\ 10 \\ -5.5 \end{bmatrix}$

a) Subtraction $\begin{bmatrix} 11 \\ -1 \\ -9 \end{bmatrix}$ b) $\begin{bmatrix} 5 \\ -6 \\ -2.5 \end{bmatrix}$

Q2 Given two vectors with common origin (0,0) and respective terminations (3.5, 4) and (4, 3) determine their lengths and the angle between them.

A2 Call the respective vectors **a** and **b**:

dist(**a**, **o**)= $\sqrt{[}(3.5 - 0)^2 + (4 - 0)^2 = 5.315$

dist(**b**, **o**)= $\sqrt{[}(4 - 0)^2 + (3 - 0)^2 = 5$

dist(**a**, **b**)= $\sqrt{[}(3.5-4)^2 + (4 - 3)^2 = 1.7697$ this value was determined so that the cosine rule could be used to find the angle between vectors.

$$\frac{5.315^2 + 5^2 - 1.7697^2}{2\text{x}5{,}315\text{x}5} = \text{Cos } \theta = 0.9429 \text{ therefore } \theta = 19.44^\circ$$

Q3 Which, if any, of the following vectors are orthogonal?

$$\mathbf{a} = \begin{bmatrix} 1 \\ -2 \\ 3 \\ -4 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 5 \\ -4 \\ 5 \\ 7 \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} 6 \\ 7 \\ 1 \\ -2 \end{bmatrix}$$

A3 **ab** = 0, **a** and **b** are orthogonal.

ac = 3, **a** and **c** are not orthogonal.

bc = -7, **b** and **c** are not orthogonal.

Q4 Given $\mathbf{a}^T = (4, 0, -3)$, $\mathbf{b}^T = (-8, 5, 0)$, $\mathbf{c}^T = (-4, 1, -2)$,

Compute i) 2**a** –3**b** ii) **a** + 3**b** – 2**c**

A4 i) $$\begin{bmatrix} 8 \\ 0 \\ -6 \end{bmatrix} - \begin{bmatrix} -24 \\ 15 \\ 0 \end{bmatrix} = \begin{bmatrix} 32 \\ -15 \\ -6 \end{bmatrix}$$

ii) $$\begin{bmatrix} 4 \\ 0 \\ -3 \end{bmatrix} + \begin{bmatrix} -24 \\ 15 \\ 0 \end{bmatrix} - \begin{bmatrix} -8 \\ 2 \\ -4 \end{bmatrix} = \begin{bmatrix} -28 \\ 13 \\ 1 \end{bmatrix}$$

Q7 For the following matrices:

$$\mathbf{A} = \begin{bmatrix} 1 & -2 & 0 \\ -3 & -1 & 4 \\ 2 & 1 & -1 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -1 & 2 & -3 \\ 1 & -3 & 1 \\ 2 & 1 & 7 \end{bmatrix}$$

a) Compute **A + B.** b) **A – B.**

c) **A x B.** d) Is **A** x **B** = **B** x **A**?

What is the outcome when **A** is multiplied by the identity matrix I_3.

A7 **A** + **B** = $\begin{bmatrix} 0 & 0 & -3 \\ -2 & -4 & 5 \\ 4 & 2 & 6 \end{bmatrix}$

A - **B** = $\begin{bmatrix} 2 & 4 & 3 \\ -4 & 2 & 3 \\ 0 & 0 & -8 \end{bmatrix}$

A x **B** = $\begin{bmatrix} -3 & 8 & -5 \\ 10 & 1 & 36 \\ -3 & 0 & -12 \end{bmatrix}$

B x **A** = $\begin{bmatrix} -13 & -3 & 11 \\ 12 & 2 & -13 \\ 13 & 2 & -3 \end{bmatrix}$

A x **B** is not equal to **B** x **A**

I x **A** = **A**

References

[1] Ayres, F. *Matrices* McGraw-Hill. (1962)

[2] Hagan, M. T. Demuth, H. B. & Beale, M. *Neural network design.* (1996)

[3] Lipschutz, S. *Linear Algebra* McGraw-Hill