Scientific contribution
No: INCOH/SAR-28/2007

STATE OF ART REPORT

ARTIFICIAL NEURAL NETWORK MODELLING

S. Mohan

Publication of
Indian National Committee on Hydrology (INCOH)
(IHP National Committee of India for UNESCO)
National Institute of Hydrology, Roorkee-247 667 INDIA

INDIAN NATIONAL COMMITTEE ON HYDROLOGY (INCOH)

(IHP National Committee of India for UNESCO)

Constituted by the Ministry of Water Resources in 1982

INCOH Secretariat

Member Secretary, Indian National Committee on Hydrology (INCOH)
National Institute of Hydrology, Roorkee – 247667, INDIA
Phone : +91 1332 277281
Fax : +91 1332 272123
E-mail : incoh@nih.ernet.in
Web : www.nih.ernet.in

Activities of INCOH

Sponsorship of Research/Applied Projects

INCOH provides sponsorship to research and development projects to fulfil its following objectives:

- To conduct studies and research using available technology.
- To develop new technology and methodologies for application to real life problems.
- To develop hydrological instruments indigenously and to evaluate new techniques.
- To develop new software for application in field hydrological problems.
- To give impetus to hydrological education and training.

About 60 research projects from different parts of the country have been sponsored uptil now. The funding for various projects has been provided under the following ten broad areas:

- Hydrology of surface water (rivers, lakes and reservoirs) including snow and glaciers.
- Remote sensing applications to hydrology and water resources.
- Hydrological information system and computers in hydrology.
- Hydrological instrumentation and telemetry.
- Environmental hydrology.
- Hydrometeorlogical aspects of water resources development and flood forecasting.
- Groundwater, springs, conjunctive use, drainage and water re-use.
- Drought, water conservation and evaporation control.
- Nuclear applications in hydrology.
- Education and training in hydrology.

Organisation of National Symposium on Hydrology

INCOH organises at least one national seminar/symposium every year to bring together hydrologists and water resources engineers from various parts of India. Since 1987, INCOH has been organising such National Seminars every year on specific focal themes.

Sponsorship of Seminar/Symposia/Conferences/Short-Term Courses

INCOH provides sponsorship to various agencies and organizations for organising national, regional and international events in hydrology.

Scientific contribution
No: INCOH/SAR-28/2007

STATE OF ART REPORT

ARTIFICIAL NEURAL NETWORK MODELLING

S. Mohan

Publication of
Indian National Committee on Hydrology (INCOH)
(IHP National Committee of India for UNESCO)
National Institute of Hydrology, Roorkee-247 667
INDIA

INDIAN NATIONAL COMMITTEE ON HYDROLOGY (INCOH)

(IHP National Committee of India for UNESCO)

Constituted by the Ministry of Water Resources in 1982

Chairman	Shri S K Das Chairman, Central Water Commission, New Delhi
Executive Member	Dr K D Sharma Director, National Institute of Hydrology, Roorkee
Member Secretary	Dr Ramakar Jha Scientist-E1, National Institute of Hydrology, Roorkee

SUB-COMMITTEES OF INCOH

STEERING COMMITTEE

Chairman	1. Member (D&R), Central Water Commission, New Delhi.
Members	2. Director, National Institute of Hydrology, Roorkee.
	3. Chairman, Central Ground Water Board, Faridabad.
	4. Chief Engineer (HSO), CWC, New Delhi.
	5. Director (R&D), Ministry of Water Resources, New Delhi.
Secretary	6. Dr Ramakar Jha, Sc. E1, NIH, Roorkee.

RESEARCH COMMITTEE (SURFACE WATER)

Chairman	1. Director, National Institute of Hydrology, Roorkee.
Members	2. Director, Centre for Water Resources Studies, National Institute of Technology, Patna.
	3. Chief Engineer (Dam Design), Irrigation Design Organisation, Roorkee.
	4. Director (Hydrology-DSR),CWC, New Delhi.
	5. Director, R&D Dte., MoWR, New Delhi.
	6. Chief Engineer (Surface Water), Irrigation Dept., Bangalore.
	7. Representative of DG, IMD, New Delhi.
	8. Director, GERI, Baroda, Gujarat.
Secretary	9. Dr S. K Jain, Sc. F, NIH, Roorkee.

RESEARCH COMMITTEE (GROUND WATER)

Chairman	1. Member (SML), Central Ground Water Board, Faridabad.
Members	2. Director (R&D), Ministry of Water Resources, New Delhi.
	3. Joint Director, CWPRS, Pune
	4. Prof. B.B.S. Singhal, Emeritus Professor, IIT Roorkee,
	5. Director, NGRI, New Delhi.
	6. Dr G.D. Gupta, Advisor, DST, New Delhi.
	7. Chief Engineer (Ground Water), Irrigation Department, Hyderabad.
	8. Dr Ramakar Jha, Sc. E1, NIH, Roorkee.
Secretary	9. Mr R.C. Jain, Sc. D, CGWB, New Delhi

PREAMBLE

The Indian National Committee on Hydrology (INCOH) is an apex body under the Ministry of Water Resources (MoWR), Government of India with the responsibility of co-ordinating activities concerning hydrology and water resources development in the country. The Committee has its members drawn from central and state government agencies as well as experts from academic and research organisations. INCOH provides technical support to MoWR in identifying the R&D schemes and studies for funding, publishing Hydrology Review Journal "Jal Vigyan Sameeksha" and supporting a number of activities including seminars, symposia, conferences, workshops, training courses, etc. The Committee is also participating in the activities of International Hydrological Programme (IHP) of UNESCO by organizing regional courses and workshops and conducting R&D works on themes of IHP. In pursuance of its objective updating the state-of-art in hydrology of the world in general and India in particular, INCOH encourages the experts to prepare these reports.

During the last decade, ANNs have emerged as a powerful tool for pattern recognition and modeling output from a system using the input data. Artificial Neural Networks are relatively crude electronic models based on the neural structure of the brain. The application of ANNs to water resources problems has become popular due to their immense power and potential in modeling non-linear systems. Among the many ANN structures that have been studied, the widely used structure in the area of hydrology is the multi-layer, feed-forward network. Neural networks learn, they are not programmed. Yet, even though they are not traditionally programmed, the designing of neural networks does require a skill. This skill involves the understanding of the network topologies, current hardware, current software tools, the application to be solved, and a strategy to acquire the necessary data to train the network. This skill further involves the selection of learning rules, transfer functions, summation functions, and how to connect the neurons within the network.

Realizing the importance of ANN modeling in water resources, INCOH invited Dr. S. Mohan, Professor, IIT Madras, Chennai for preparing a state-of-art report on the status of "ANN Modelling". It is hoped that this state-of-art report would serve as a useful reference material to practicing engineers, researchers, field engineers, planners, stakeholders and implementing agencies, who are involved in the estimation and optimal utilization of water resources in the country.

18/11/07

(K.D.Sharma)
Executive Member INCOH &
Director, National Institute of Hydrology, Roorkee

CONTENTS

ARTIFICIAL NEURAL NETWORKS

1.0 INTRODUCTION

Artificial Neural Networks are relatively crude electronic models based on the neural structure of the brain. The brain basically learns from experience. It is natural proof that some problems that are beyond the scope of current computers are indeed solvable by small energy efficient packages. This brain modeling also promises a less technical way to develop machine solutions. This new approach to computing also provides a more graceful degradation during system overload than its more traditional counterparts.

These biologically inspired methods of computing are thought to be the next major advancement in the computing industry. Even simple animal brains are capable of functions that are currently impossible for computers. Computers do rote things well, like keeping ledgers or performing complex math. But computers have trouble recognizing even simple patterns much less generalizing those patterns of the past into actions of the future.

Now, advances in biological research promise an initial understanding of the natural thinking mechanism. This research shows that brains store information as patterns. Some of these patterns are very complicated and allow us the ability to recognize individual faces from many different angles. This process of storing information as patterns, utilizing those patterns, and then solving problems encompasses a new field in computing. This field, as mentioned before, does not utilize traditional programming but involves the creation of massively parallel networks and the training of those networks to solve specific problems. This field also utilizes words very different from traditional computing, words like behave, react, self-organize, learn, generalize, and forget.

1.1 Analogy to the Brain

The exact workings of the human brain are still a mystery. Yet, some aspects of this amazing processor are known. In particular, the most basic element of the human brain is a specific type of cell which, unlike the rest of the body, doesn't appear to regenerate. Because this type of cell is the only part of the body that isn't slowly replaced, it is assumed that these cells are what provides us with our abilities to remember, think, and apply previous experiences to our every action. These cells, all 100 billion of them, are known as neurons. Each of these neurons can connect with up to 200,000 other neurons, although 1,000 to 10,000 are typical. The power of the human mind comes from the sheer numbers of these basic components and the multiple connections between them. It also comes from genetic programming and learning.

The individual neurons are complicated. They have a myriad of parts, sub-systems, and control mechanisms. They convey information via a host of electrochemical pathways. There are over one hundred different classes of neurons, depending on the classification method used. Together these neurons and their connections form a process which is not binary, not stable, and not synchronous. In short, it is nothing like the currently available electronic computers, or even artificial neural networks. These artificial neural networks try to replicate only the most basic elements of this complicated, versatile, and powerful organism.

2.0 HISTORY OF NEURAL NETWORKS

The history of the ANNs stems from the 1940s, the decade of the first electronic computer. However, the first significant step took place in 1957 when Rosenblatt introduced the first concrete neural model, the perceptron. Rosenblatt also took part in constructing the first successful neurocomputer, the Mark I Perceptron. After this initial impulse, the development of ANNs has proceeded as described in Figure 1.

In 1959, Bernard Widrow and Marcian Hoff of Stanford developed models they called ADALINE and MADALINE. These models were named for their use of Multiple ADAptive LINear Elements. MADALINE was the first neural network to be applied to a real world problem. It is an adaptive filter which eliminates echoes on phone lines. This neural network is still in commercial use.

Rosenblatt's original perceptron model contained only one layer. From this, a multi-layered model was derived in 1960. At first, the use of the multi-layer perceptron (MLP) was complicated by the lack of a suitable learning algorithm. In 1974, Werbos came to rescue by introducing a so-called backpropagation algorithm for the three-layered perceptron network. The application area of the MLP networks remained rather limited until the breakthrough in 1986 when a general backpropagation algorithm for a multi-layered perceptron was introduced by Rummelhart and Mclelland.

Hopfield brought out his idea of a neural network in 1982. Unlike the neurons in MLP, the Hopfield network consists of only one layer whose neurons are fully connected with each other. Since then, new versions of the Hopfield network have been developed. The Boltzmann machine has been influenced by both the Hopfield network and the MLP.

Adaptive Resonance Theory (ART) was first introduced by Carpenter and Grossberg in 1983. The development of ART has continued and resulted in the more advanced ART II and ART III network models.

Radial Basis Function (RBF) networks were first introduced by Broomhead & Lowe in 1988. Although the basic idea of RBF was developed 30 years ago under the name method of potential function, the work by Broomhead & Lowe opened a new frontier in the neural network community.

A totally unique kind of network model is the Self-Organizing Map (SOM) introduced by Kohonen in 1982. SOM is a certain kind of topological map which organizes itself based on the input patterns that it is trained with. The SOM originated from the LVQ (Learning Vector Quantization) network the underlying idea of which was also Kohonen's in 1972.

By 1985 the American Institute of Physics began what has become an annual meeting - Neural Networks for Computing. By 1987, the Institute of Electrical and Electronic Engineer's (IEEE) first International Conference on Neural Networks drew more than 1,800 attendees.

By 1989 at the Neural Networks for Defense meeting Bernard Widrow told his audience that they were engaged in World War IV, "World War III never happened," where the battlefields are world trade and manufacturing. The 1990 US Department of Defense Small Business Innovation Research Program named 16 topics which specifically

targeted neural networks with an additional 13 mentioning the possible use of neural networks.

Today, neural networks discussions are occurring everywhere. Their promise seems very bright as nature itself is the proof that this kind of thing works. Yet, its future, indeed the very key to the whole technology, lies in hardware development.

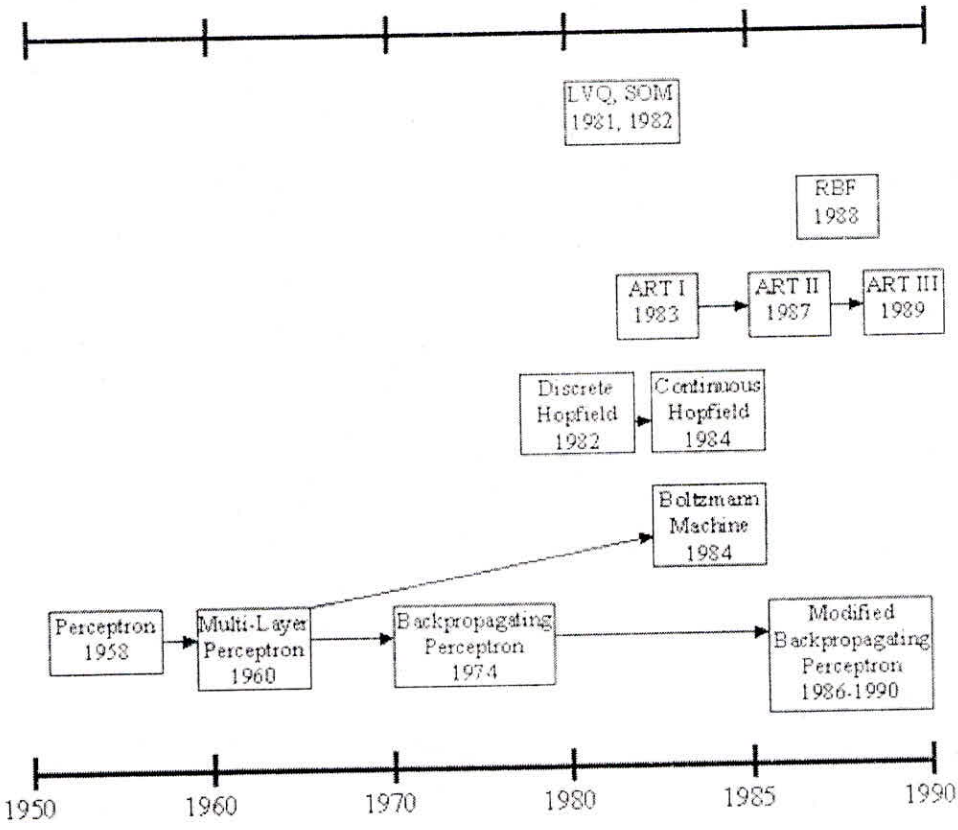


Figure 1: The evolution of the most popular artificial neural networks.

2.1 Structure of the ANN

The artificial neural networks can be classified according to the structure that they exhibit. Figure 2 represents four commonly used neural network structures.

Figure 2.a represents the structure of a multi-layered feedforward network. The neurons in this ANN model are grouped in layers which are connected to the direction of the passing signal (from left to right in this case). There are no lateral connections within each layer and also no feedbackward connections within the network. The best-known ANN of this type is the perceptron network.

Figure 2.b depicts a single-layered fully connected network model where each neuron is laterally connected to all neighbouring neurons in the layer. In this ANN model, all

neurons are both input and output neurons. The best-known ANN of this type is the Hopfield network.

Figure 2.c demonstrates the connections in a two-layered feedforward/feedbackward network. The layers in this ANN model are connected to both directions. As a pattern is presented to the network, it 'resonates' a certain number of times between the layers before a response is received from the output layer. The best-known ANN of this type is the Adaptive Resonance Theory (ART) network.

Figure 2.d illustrates the idea of a topologically organized feature map. In this model, each neuron in the network contains a so-called feature vector. As a pattern from the training data is given to the network, the neuron whose feature vector is closest to the input vector is activated. The activated neuron is called the best matching unit (BMU) and it is updated to reflect input vector causing the activation. In the process of updating the BMU, the neighbouring neurons are updated towards the input vector or away from it (according to the learning algorithm in use). The network type exhibiting this kind of behaviour is the Self-Organizing Map of Kohonen.

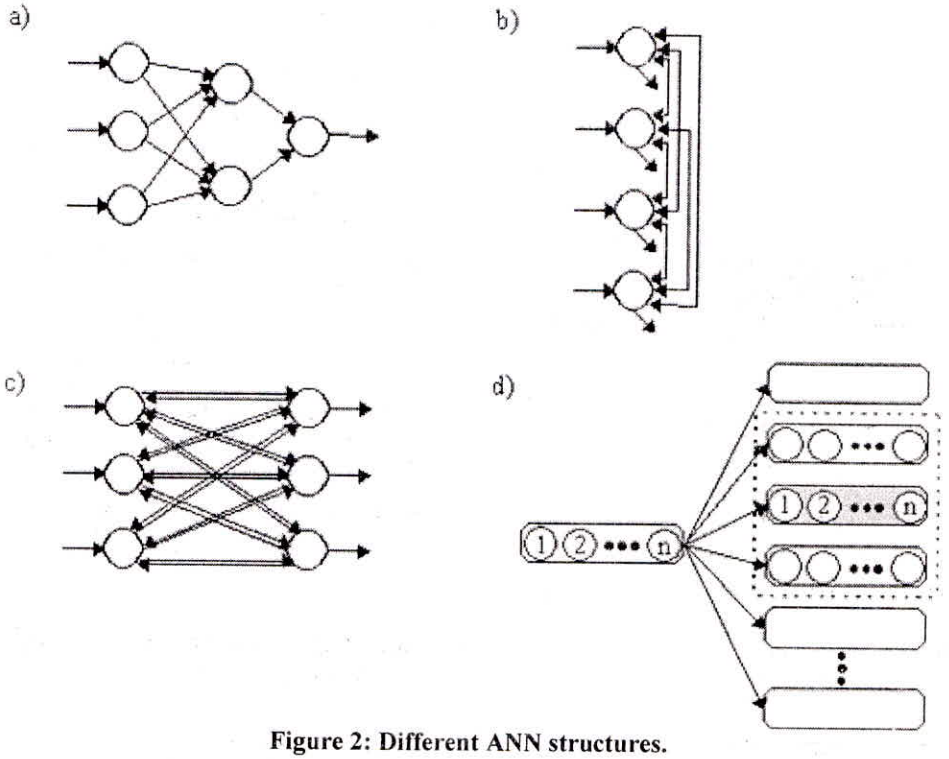


Figure 2: Different ANN structures.

a) Multi-layered feedforward network, b) Single-layered fully connected network,

c) Two-layered feedforward/feedbackward network d) Topographically organized vector map.

2.2 Artificial Neurons and How They Work

The fundamental processing element of a neural network is a neuron. This building block of human awareness encompasses a few general capabilities. Basically, a biological neuron receives inputs from other sources, combines them in some way, performs a generally nonlinear operation on the result, and then outputs the final result. Figure 3 shows the relationship of these four parts.

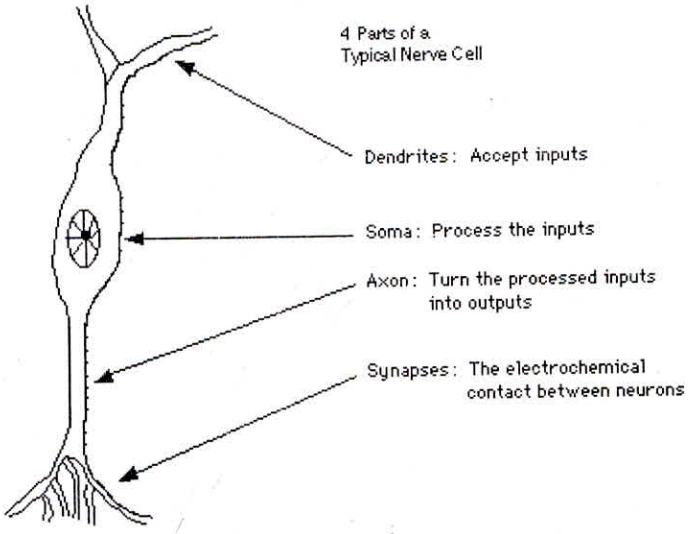


Figure 3: A Simple Neuron.

Within humans there are many variations on this basic type of neuron, further complicating man's attempts at electrically replicating the process of thinking. Yet, all natural neurons have the same four basic components. These components are known by their biological names - dendrites, soma, axon, and synapses. Dendrites are hair-like extensions of the soma which act like input channels. These input channels receive their input through the synapses of other neurons. The soma then processes these incoming signals over time. The soma then turns that processed value into an output which is sent out to other neurons through the axon and the synapses.

Recent experimental data has provided further evidence that biological neurons are structurally more complex than the simplistic explanation above. They are significantly more complex than the existing artificial neurons that are built into today's artificial neural networks. As biology provides a better understanding of neurons, and as technology advances, network designers can continue to improve their systems by building upon man's understanding of the biological brain.

But currently, the goal of artificial neural networks is not the grandiose recreation of the brain. On the contrary, neural network researchers are seeking an understanding of nature's capabilities for which people can engineer solutions to problems that have not been solved by traditional computing.

To do this, the basic unit of neural networks, the artificial neurons, simulate the four basic functions of natural neurons. Figure 4 shows a fundamental representation of an artificial neuron.

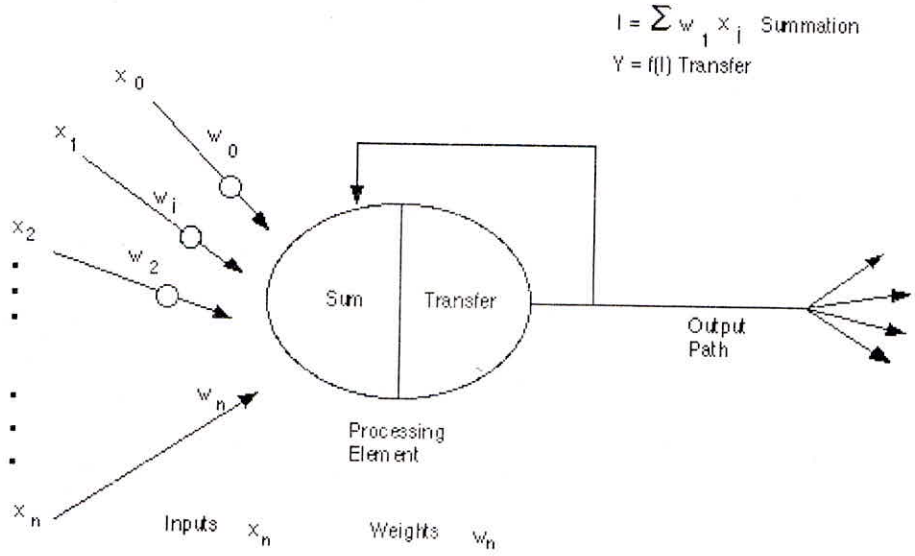


Figure 4: A Basic Artificial Neuron.

In Figure 4, various inputs to the network are represented by the mathematical symbol, x(n). Each of these inputs are multiplied by a connection weight. These weights are represented by w(n). In the simplest case, these products are simply summed, fed through a transfer function to generate a result, and then output. This process lends itself to physical implementation on a large scale in a small package. This electronic implementation is still possible with other network structures which utilize different summing functions as well as different transfer functions.

Some applications require "black and white," or binary, answers. These applications include the recognition of text, the identification of speech, and the image deciphering of scenes. These applications are required to turn real-world inputs into discrete values. These potential values are limited to some known set, like the ASCII characters or the most common 50,000 English words. Because of this limitation of output options, these applications don't always utilize networks composed of neurons that simply sum up, and thereby smooth, inputs. These networks may utilize the binary properties of ORing and ANDing of inputs. These functions, and many others, can be built into the summation and transfer functions of a network.

Other networks work on problems where the resolutions are not just one of several known values. These networks need to be capable of an infinite number of responses. Applications of this type include the "intelligence" behind robotic movements. This "intelligence" processes inputs and then creates outputs which actually cause some device to move. That movement can span an infinite number of very precise motions. These networks do indeed want to smooth their inputs which, due to limitations of sensors,

comes in non-continuous bursts, say thirty times a second. To do that, they might accept these inputs, sum that data, and then produce an output by, for example, applying a hyperbolic tangent as a transfer function. In this manner, output values from the network are continuous and satisfy more real world interfaces.

Other applications might simply sum and compare to a threshold, thereby producing one of two possible outputs, a zero or a one. Other functions scale the outputs to match the application, such as the values minus one and one. Some functions even integrate the input data over time, creating time-dependent networks.

2.3 Electronic Implementation of Artificial Neurons

In currently available software packages these artificial neurons are called "processing elements" and have many more capabilities than the simple artificial neuron described above. Those capabilities will be discussed later in this report. Figure 2.2.3 is a more detailed schematic of this still simplistic artificial neuron.

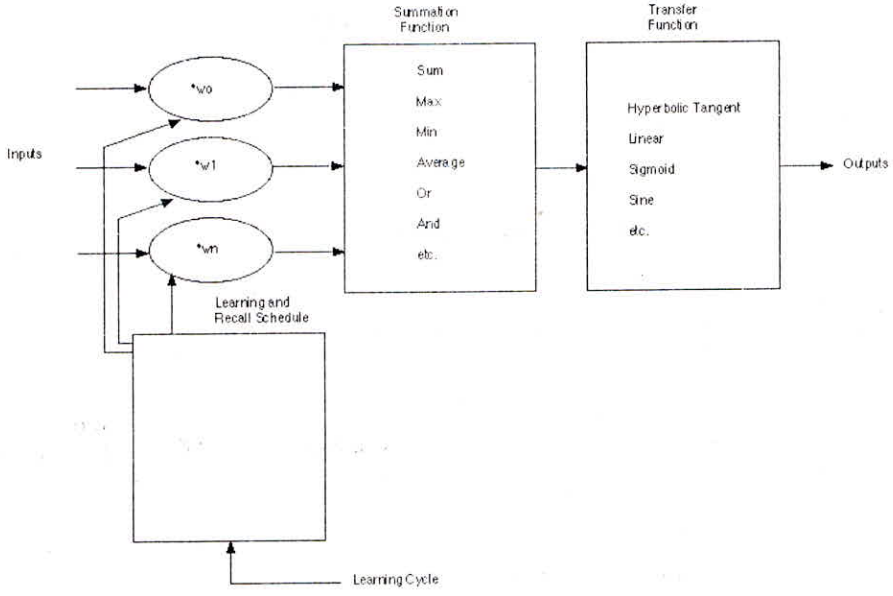


Figure 5: A Model of a "Processing Element"

In Figure 5, inputs enter into the processing element from the upper left. The first step is for each of these inputs to be multiplied by their respective weighting factor (w(n)). Then these modified inputs are fed into the summing function, which usually just sums these products. Yet, many different types of operations can be selected. These operations could produce a number of different values which are then propagated forward; values such as the average, the largest, the smallest, the ORed values, the ANDed values, etc. Furthermore, most commercial development products allow software engineers to create their own summing functions via routines coded in a higher level language (C is commonly supported). Sometimes the summing function is further complicated by the addition of an activation function which enables the summing function to operate in a time sensitive way.

Either way, the output of the summing function is then sent into a transfer function. This function then turns this number into a real output via some algorithm. It is this algorithm that takes the input and turns it into a zero or a one, a minus one or a one, or some other number. The transfer functions that are commonly supported are sigmoid, sine, hyperbolic tangent, etc. This transfer function also can scale the output or control its value via thresholds. The result of the transfer function is usually the direct output of the processing element. An example of how a transfer function works is shown in Figure 6.

This sigmoid transfer function takes the value from the summation function, called sum in the Figure 2.2.4, and turns it into a value between zero and one.

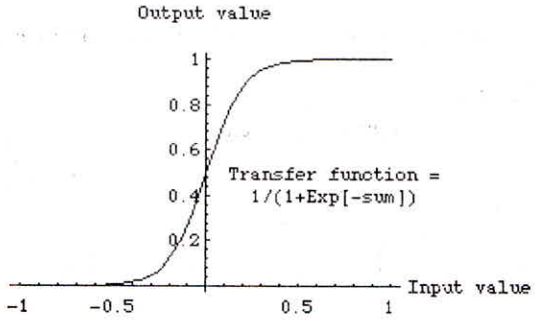


Figure 6: Sigmoid Transfer Function.

Finally, the processing element is ready to output the result of its transfer function. This output is then input into other processing elements, or to an outside connection, as dictated by the structure of the network.

All artificial neural networks are constructed from this basic building block - the processing element or the artificial neuron. It is variety and the fundamental differences in these building blocks which partially cause the implementing of neural networks to be an "art."

2.4 Artificial Network Operations

The other part of the "art" of using neural networks revolve around the myriad of ways these individual neurons can be clustered together. This clustering occurs in the human mind in such a way that information can be processed in a dynamic, interactive, and self-organizing way. Biologically, neural networks are constructed in a three-dimensional world from microscopic components. These neurons seem capable of nearly unrestricted interconnections. That is not true of any proposed, or existing, man-made network. Integrated circuits, using current technology, are two-dimensional devices with a limited number of layers for interconnection. This physical reality restrains the types, and scope, of artificial neural networks that can be implemented in silicon.

Currently, neural networks are the simple clustering of the primitive artificial neurons. This clustering occurs by creating layers which are then connected to one another. How these layers connect is the other part of the "art" of engineering networks to resolve real world problems.

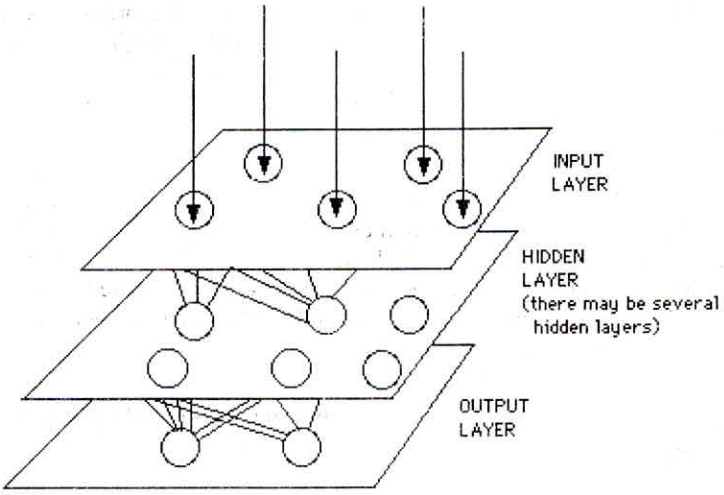


Figure 7: A Simple Neural Network Diagram

Basically, all artificial neural networks have a similar structure or topology as shown in Figure 7. In that structure some of the neurons interface to the real world to receive its inputs. Other neurons provide the real world with the network's outputs. This output might be the particular character that the network thinks that it has scanned or the particular image it thinks is being viewed. All the rest of the neurons are hidden from view.

But a neural network is more than a bunch of neurons. Some early researchers tried to simply connect neurons in a random manner, without much success. Now, it is known that even the brains of snails are structured devices. One of the easiest ways to design a structure is to create layers of elements. It is the grouping of these neurons into layers, the connections between these layers, and the summation and transfer functions that comprises a functioning neural network. The general terms used to describe these characteristics are common to all networks.

Although there are useful networks which contain only one layer, or even one element, most applications require networks that contain at least the three normal types of layers - input, hidden, and output. The layer of input neurons receives the data either from input files or directly from electronic sensors in real-time applications. The output layer sends information directly to the outside world, to a secondary computer process, or to other devices such as a mechanical control system. Between these two layers can be many hidden layers. These internal layers contain many of the neurons in various interconnected structures. The inputs and outputs of each of these hidden neurons simply go to other neurons.

In most networks each neuron in a hidden layer receives the signals from all of the neurons in a layer above it, typically an input layer. After a neuron performs its function it passes its output to all of the neurons in the layer below it, providing a feed forward path to the output.

These lines of communication from one neuron to another are important aspects of neural networks. They are the glue to the system. They are the connections which provide a variable strength to an input. There are two types of these connections. One causes the summing mechanism of the next neuron to add while the other causes it to subtract. In more human terms one excites while the other inhibits.

Some networks want a neuron to inhibit the other neurons in the same layer. This is called lateral inhibition. The most common use of this is in the output layer. For example in text recognition if the probability of a character being a "P" is .85 and the probability of the character being an "F" is .65, the network wants to choose the highest probability and inhibit all the others. It can do that with lateral inhibition. This concept is also called competition.

Another type of connection is feedback. This is where the output of one layer routes back to a previous layer. An example of this is shown in Figure 8.

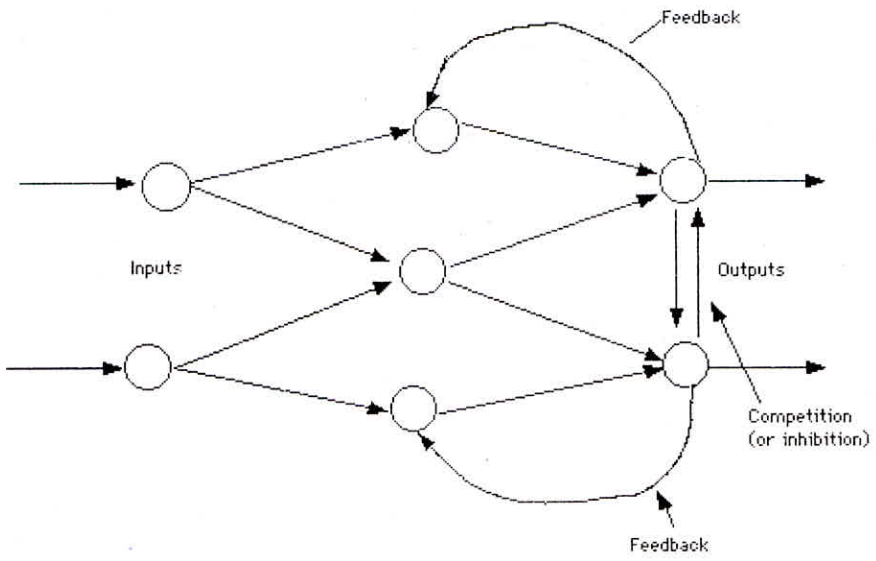


Figure 8: Simple Network with Feedback and Competition

The way that the neurons are connected to each other has a significant impacton the operation of the network. In the larger, more professional softwaredevelopment packages the user is allowed to add, delete, and control theseconnections at will. By "tweaking" parameters these connections can be made toeither excite or inhibit.

2.5 Training an Artificial Neural Network

Once a network has been structured for a particular application, that network is ready to be trained. To start this process the initial weights are chosen randomly. Then, the training, or learning, begins.

There are two approaches to training - supervised and unsupervised. Supervised training involves a mechanism of providing the network with the desired output either by

manually "grading" the network's performance or by providing the desired outputs with the inputs. Unsupervised training is where the network has to make sense of the inputs without outside help.

The vast bulk of networks utilize supervised training. Unsupervised training is used to perform some initial characterization on inputs. However, in the full blown sense of being truly self learning, it is still just a shining promise that is not fully understood, does not completely work, and thus is relegated to the lab.

2.5.1 Supervised Training

In supervised training, both the inputs and the outputs are provided. The network then processes the inputs and compares its resulting outputs against the desired outputs. Errors are then propagated back through the system, causing the system to adjust the weights which control the network. This process occurs over and over as the weights are continually tweaked. The set of data which enables the training is called the "training set." During the training of a network the same set of data is processed many times as the connection weights are ever refined.

The current commercial network development packages provide tools to monitor how well an artificial neural network is converging on the ability to predict the right answer. These tools allow the training process to go on for days, stopping only when the system reaches some statistically desired point, or accuracy. However, some networks never learn. This could be because the input data does not contain the specific information from which the desired output is derived. Networks also don't converge if there is not enough data to enable complete learning. Ideally, there should be enough data so that part of the data can be held back as a test. Many layered networks with multiple nodes are capable of memorizing data. To monitor the network to determine if the system is simply memorizing its data in some non-significant way, supervised training needs to hold back a set of data to be used to test the system after it has undergone its training. (Note: memorization is avoided by not having too many processing elements.)

If a network simply can't solve the problem, the designer then has to review the input and outputs, the number of layers, the number of elements per layer, the connections between the layers, the summation, transfer, and training functions, and even the initial weights themselves. Those changes required to create a successful network constitute a process wherein the "art" of neural networking occurs.

Another part of the designer's creativity governs the rules of training. There are many laws (algorithms) used to implement the adaptive feedback required to adjust the weights during training. The most common technique is backward-error propagation, more commonly known as back-propagation. These various learning techniques are explored in greater depth later in this report.

Yet, training is not just a technique. It involves a "feel," and conscious analysis, to insure that the network is not over-trained. Initially, an artificial neural network configures itself with the general statistical trends of the data. Later, it continues to "learn" about other aspects of the data which may be spurious from a general viewpoint.

When finally the system has been correctly trained, and no further learning is needed, the weights can, if desired, be "frozen." In some systems this finalized network is then turned

into hardware so that it can be fast. Other systems don't lock themselves in but continue to learn while in production use.

2.5.2 Unsupervised or Adaptive Training

The other type of training is called unsupervised training. In unsupervised training, the network is provided with inputs but not with desired outputs. The system itself must then decide what features it will use to group the input data. This is often referred to as self-organization or adaption.

At the present time, unsupervised learning is not well understood. This adaption to the environment is the promise which would enable science fiction types of robots to continually learn on their own as they encounter new situations and new environments. Life is filled with situations where exact training sets do not exist. Some of these situations involve military action where new combat techniques and new weapons might be encountered. Because of this unexpected aspect to life and the human desire to be prepared, there continues to be research into, and hope for, this field. Yet, at the present time, the vast bulk of neural network work is in systems with supervised learning. Supervised learning is achieving results.

One of the leading researchers into unsupervised learning is Tuevo Kohonen, an electrical engineer at the Helsinki University of Technology. He has developed a self-organizing network, sometimes called an auto-associator that learns without the benefit of knowing the right answer. It is an unusual looking network in that it contains one single layer with many connections. The weights for those connections have to be initialized and the inputs have to be normalized. The neurons are set up to compete in a winner-take-all fashion.

Kohonen continues his research into networks that are structured differently than standard, feedforward, back-propagation approaches. Kohonen's work deals with the grouping of neurons into fields. Neurons within a field are "topologically ordered." Topology is a branch of mathematics that studies how to map from one space to another without changing the geometric configuration. The three-dimensional groupings often found in mammalian brains are an example of topological ordering.

Kohonen has pointed out that the lack of topology in neural network models make today's neural networks just simple abstractions of the real neural networks within the brain. As this research continues, more powerful self learning networks may become possible. But currently, this field remains one that is still in the laboratory.

2.6 How Neural Networks Differ from Traditional Computing and Expert Systems

Neural networks offer a different way to analyze data, and to recognize patterns within that data, than traditional computing methods. However, they are not a solution for all computing problems. Traditional computing methods work well for problems that can be well characterized. Balancing checkbooks, keeping ledgers, and keeping tabs of inventory are well defined and do not require the special characteristics of neural networks. Table 1 identifies the basic differences between the two computing approaches.

Traditional computers are ideal for many applications. They can process data, track inventories, network results, and protect equipment. These applications do not need the special characteristics of neural networks.

Expert systems are an extension of traditional computing and are sometimes called the fifth generation of computing. (First generation computing used switches and wires. The second generation occurred because of the development of the transistor. The third generation involved solid-state technology, the use of integrated circuits, and higher level languages like COBOL, Fortran, and "C". End user tools, "code generators," are known as the fourth generation.) The fifth generation involves artificial intelligence.

Table 1 Comparison of Computing Approaches

CHARACTERISTICS	TRADITIONAL COMPUTING (including Expert Systems)	ARTIFICIAL NEURAL NETWORKS
Processing style Functions	Sequential Logically (left brained) via Rules Concepts Calculations	Parallel Gestault (right brained) via Images Pictures Controls
Learning Method Applications	by rules (didactically) accounting word processing, math inventory digital communications	by example (Socratically) Sensor processing speech recognition pattern recognition text recognition

Typically, an expert system consists of two parts, an inference engine and a knowledge base. The inference engine is generic. It handles the user interface, external files, program access, and scheduling. The knowledge base contains the information that is specific to a particular problem. This knowledge base allows an expert to define the rules which govern a process. This expert does not have to understand traditional programming. That person simply has to understand both what he wants a computer to do and how the mechanism of the expert system shell works. It is this shell, part of the inference engine that actually tells the computer how to implement the expert's desires. This implementation occurs by the expert system generating the computer's programming itself, it does that through "programming" of its own. This programming is needed to establish the rules for a particular application. This method of establishing rules is also complex and does require a detail oriented person.

Efforts to make expert systems general have run into a number of problems. As the complexity of the system increases, the system simply demands too much computing resources and becomes too slow. Expert systems have been found to be feasible only when narrowly confined.

Artificial neural networks offer a completely different approach to problem solving and they are sometimes called the sixth generation of computing. They try to provide a tool that both programs itself and learns on its own. Neural networks are structured to provide the capability to solve problems without the benefits of an expert and without the need of programming. They can seek patterns in data that no one knows are there. A comparison of artificial intelligence's expert systems and neural network is contained in Table 2.

Expert systems have enjoyed significant successes. However, artificial intelligence has encountered problems in areas such as vision, continuous speech recognition and synthesis, and machine learning. Artificial intelligence also is hostage to the speed of the processor that it runs on. Ultimately, it is restricted to the theoretical limit of a single processor. Artificial intelligence is also burdened by the fact that experts don't always speak in rules.

Yet, despite the advantages of neural networks over both expert systems and more traditional computing in these specific areas, neural nets are not complete solutions. They offer a capability that is not ironclad, such as a debugged accounting system. They learn, and as such, they do continue to make "mistakes." Furthermore, even when a network has been developed, there is no way to ensure that the network is the optimal network.

Table 2 Comparison of Expert Systems and Neural Networks

Characteristics	Von Neumann Architecture Used for Expert Systems	Artificial Neural Networks
Processors	VLSI (traditional processors)	Artificial Neural Networks; variety of technologies; hardware development is on going
Processing Approach	Separate	The same
Processing Approach	Processes problem rule at a one time; sequential	Multiple, simultaneously
Connections	Externally programmable	Dynamically self programming
Self learning	Only algorithmic parameters modified	Continuously adaptable
Fault tolerance	None without special processors	Significant in the very nature of the interconnected neurons
Neurobiology in design	None	Moderate
Programming	Through a rule based complicated	Self-programming; but network must be set up properly
Ability to be fast	Requires big processors	Requires multiple custom-built chips

Neural systems do exact their own demands. They do require their implementer to meet a number of conditions. These conditions include:

- a data set which includes the information which can characterize the problem.
- an adequately sized data set to both train and test the network.
- an understanding of the basic nature of the problem to be solved so that basic first-cut decision on creating the network can be made. These decisions include the activization and transfer functions, and the learning methods.
- an understanding of the development tools.

adequate processing power (some applications demand real-time processing that exceeds what is available in the standard, sequential processing hardware. The development of hardware is the key to the future of neural networks).

Once these conditions are met, neural networks offer the opportunity of solving problems in an arena where traditional processors lack both the processing power and a step-by-step methodology. A number of very complicated problems cannot be solved in the traditional computing environments. For example, speech is something that all people can easily parse and understand. A person can understand a southern drawl, a Bronx accent, and the slurred words of a baby. Without the massively paralleled processing power of a neural network, this process is virtually impossible for a computer. Image recognition is another task that a human can easily do but which stymies even the biggest of computers. A person can recognize a plane as it turns, flies overhead, and disappears into a dot. A traditional computer might try to compare the changing images to a number of very different stored patterns.

This new way of computing requires skills beyond traditional computing. It is a natural evolution. Initially, computing was only hardware and engineers made it work. Then, there were software specialists - programmers, systems engineers, data base specialists, and designers. Now, there are also neural architects. This new professional needs to be skilled different than its predecessors of the past. For instance, he will need to know statistics in order to choose and evaluate training and testing situations. This skill of making neural networks work is one that will stress the logical thinking of current software engineers.

In summary, neural networks offer a unique way to solve some problems while making their own demands. The biggest demand is that the process is not simply logic. It involves an empirical skill, an intuitive feel as to how a network might be created.

3.0 Detailed Description of Neural Network Components

Now that there is a general understanding of artificial neural networks, it is appropriate to explore them in greater detail. But before jumping into the various networks, a more complete understanding of the inner workings of an neural network is needed. As stated earlier, artificial neural networks are a large class of parallel processing architectures which are useful in specific types of complex problems. These architectures should not be confused with common parallel processing configurations which apply many sequential processing units to standard computing topologies. Instead, neural networks are radically different than conventional Von Neumann computers in that they crudely mimic the fundamental properties of man's brain.

As mentioned earlier, artificial neural networks are loosely based on biology. Current research into the brain's physiology has unlocked only a limited understanding of how neurons work or even what constitutes intelligence in general. Researchers are working in both the biological and engineering fields to further decipher the key mechanisms for how man learns and reacts to everyday experiences. Improved knowledge in neural processing helps create better, more succinct artificial networks. It also creates a cornucopia of new, and ever evolving, architectures. Kunihiko Fukushima, a senior research scientist in Japan, describes the give and take of building a neural network model; "We try to follow physiological evidence as faithfully as possible. For parts not yet clear, however, we construct a hypothesis and build a model that follows that

hypothesis. We then analyze or simulate the behavior of the model and compare it with that of the brain. If we find any discrepancy in the behavior between the model and the brain, we change the initial hypothesis and modify the model. We repeat this procedure until the model behaves in the same way as the brain." This common process has created thousands of network topologies.

3.1 Major Components of an Artificial Neuron

This section describes the seven major components which make up an artificial neuron. These components are valid whether the neuron is used for input, output, or is in one of the hidden layers.

Component 1

Weighting Factors: A neuron usually receives many simultaneous inputs. Each input has its own relative weight which gives the input the impact that it needs on the processing element's summation function. These weights perform the same type of function as do the the varying synaptic strengths of biological neurons. In both cases, some inputs are made more important than others so that they have a greater effect on the processing element as they combine to produce a neural response.

Weights are adaptive coefficients within the network that determine the intensity of the input signal as registered by the artificial neuron. They are a measure of an input's connection strength. These strengths can be modified in response to various training sets and according to a network's specific topology or through its learning rules.

Component 2

Summation Function: The first step in a processing element's operation is to compute the weighted sum of all of the inputs. Mathematically, the inputs and the corresponding weights are vectors which can be represented as (i1, i2 . . . i*n*) and (w1, w2 . . . w*n*). The total input signal is the dot, or inner, product of these two vectors. This simplistic summation function is found by multiplying each component of the i vector by the corresponding component of the w vector and then adding up all the products. Input1 = i1 * w1, input2 = i2 * w2, etc., are added as input1 + input2 + . . . + input *n*. The result is a single number, not a multi-element vector.

Geometrically, the inner product of two vectors can be considered a measure of their similarity. If the vectors point in the same direction, the inner product is maximum; if the vectors point in opposite direction (180 degrees out of phase), their inner product is minimum.

The summation function can be more complex than just the simple input and weight sum of products. The input and weighting coefficients can be combined in many different ways before passing on to the transfer function. In addition to a simple product summing, the summation function can select the minimum, maximum, majority, product, or several normalizing algorithms. The specific algorithm for combining neural inputs is determined by the chosen network architecture and paradigm.

Some summation functions have an additional process applied to the result before it is passed on to the transfer function. This process is sometimes called the activation function. The purpose of utilizing an activation function is to allow the summation output to vary with respect to time. Activation functions currently are pretty much confined to research. Most of the current network implementations use an "identity" activation function, which is equivalent to not having one. Additionally, such a function is likely to be a component of the network as a whole rather than of each individual processing element component.

Component 3

Transfer Function: The result of the summation function, almost always the weighted sum, is transformed to a working output through an algorithmic process known as the transfer function. In the transfer function the summation total can be compared with some threshold to determine the neural output. If the sum is greater than the threshold value, the processing element generates a signal. If the sum of the input and weight products is less than the threshold, no signal (or some inhibitory signal) is generated. Both types of response are significant.

The threshold, or transfer function, is generally non-linear. Linear (straight-line) functions are limited because the output is simply proportional to the input. Linear functions are not very useful. That was the problem in the earliest network models as noted in Minsky and Papert's book *Perceptrons*.

The transfer function could be something as simple as depending upon whether the result of the summation function is positive or negative. The network could output zero and one, one and minus one, or other numeric combinations. The transfer function would then be a "hard limiter" or step function. Figure 9 depicts some samples of transfer functions.

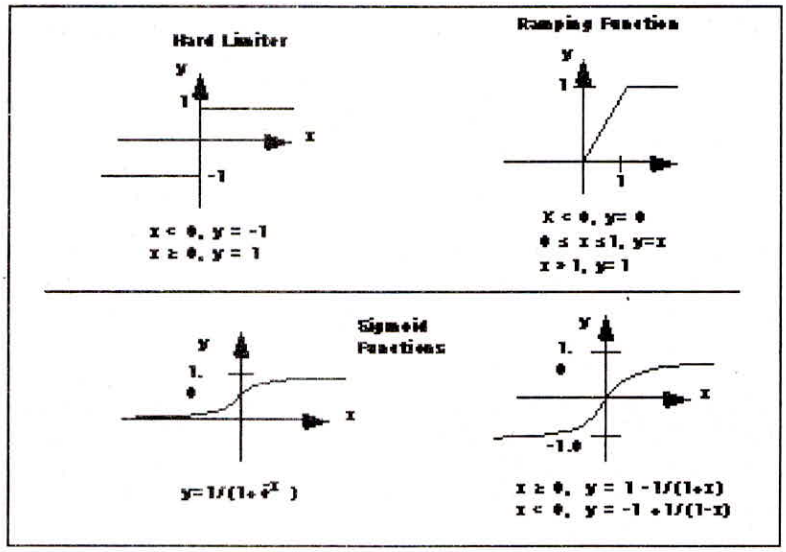


Figure 9: Sample Transfer Functions

Another type of transfer function, the threshold or ramping function, could mirror the input within a given range and still act as a hard limiter outside that range. It is a linear function that has been clipped to minimum and maximum values, making it non-linear. Yet another option would be a sigmoid or S-shaped curve. That curve approaches a minimum and maximum value at the asymptotes. It is common for this curve to be called a sigmoid when it ranges between 0 and 1, and a hyperbolic tangent when it ranges between -1 and 1. Mathematically, the exciting feature of these curves is that both the function and its derivatives are continuous. This option works fairly well and is often the transfer function of choice. Other transfer functions are dedicated to specific network architectures and will be discussed later.

Prior to applying the transfer function, uniformly distributed random noise may be added. The source and amount of this noise is determined by the learning mode of a given network paradigm. This noise is normally referred to as "temperature" of the artificial neurons. The name, temperature, is derived from the physical phenomenon that as people become too hot or cold their ability to think is affected. Electronically, this process is simulated by adding noise. Indeed, by adding different levels of noise to the summation result, more brain-like transfer functions are realized. To more closely mimic nature's characteristics, some experimenters are using a gaussian noise source. Gaussian noise is similar to uniformly distributed noise except that the distribution of random numbers within the temperature range is along a bell curve. The use of temperature is an ongoing research area and is not being applied to many engineering applications.

Another network topology which uses what it calls a temperature coefficient in a new feed-forward is back-propagation learning function. But this temperature coefficient is a global term which is applied to the gain of the transfer function. It should not be confused with the more common term, temperature, which is simple noise being added to individual neurons. In contrast, the global temperature coefficient allows the transfer function to have a learning variable much like the synaptic input weights. This concept is claimed to create a network which has a significantly faster (by several order of magnitudes) learning rate and provides more accurate results than other feedforward, back-propagation networks.

Component 4

Scaling and Limiting: After the processing element's transfer function, the result can pass through additional processes which scale and limit. This scaling simply multiplies a scale factor times the transfer value, and then adds an offset. Limiting is the mechanism which insures that the scaled result does not exceed an upper or lower bound. This limiting is in addition to the hard limits that the original transfer function may have performed.

This type of scaling and limiting is mainly used in topologies to test biological neuron models, such as James Anderson's brain-state-in-the-box.

Component 5

Output Function (Competition): Each processing element is allowed one output signal which it may output to hundreds of other neurons. This is just like the biological neuron, where there are many inputs and only one output action. Normally, the output is directly equivalent to the transfer function's result. Some network topologies, however, modify

the transfer result to incorporate competition among neighboring processing elements. Neurons are allowed to compete with each other, inhibiting processing elements unless they have great strength. Competition can occur at one or both of two levels. First, competition determines which artificial neuron will be active, or provides an output. Second, competitive inputs help determine which processing element will participate in the learning or adaptation process.

Component 6

Error Function and Back-Propagated Value: In most learning networks the difference between the current output and the desired output is calculated. This raw error is then transformed by the error function to match a particular network architecture. The most basic architectures use this error directly, but some square the error while retaining its sign, some cube the error, other paradigms modify the raw error to fit their specific purposes. The artificial neuron's error is then typically propagated into the learning function of another processing element. This error term is sometimes called the current error.

The current error is typically propagated backwards to a previous layer. Yet, this back-propagated value can be either the current error, the current error scaled in some manner (often by the derivative of the transfer function), or some other desired output depending on the network type. Normally, this back-propagated value, after being scaled by the learning function, is multiplied against each of the incoming connection weights to modify them before the next learning cycle.

Component 7

Learning Function: The purpose of the learning function is to modify the variable connection weights on the inputs of each processing element according to some neural based algorithm. This process of changing the weights of the input connections to achieve some desired result can also be called the adaption function, as well as the learning mode. There are two types of learning: supervised and unsupervised. Supervised learning requires a teacher. The teacher may be a training set of data or an observer who grades the performance of the network results. Either way, having a teacher is learning by reinforcement. When there is no external teacher, the system must organize itself by some internal criteria designed into the network. This is learning by doing.

3.2 Learning by Artificial Neural Network

3.2.1 Supervised Learning

The vast majority of artificial neural network solutions have been trained with supervision. In this mode, the actual output of a neural network is compared to the desired output. Weights, which are usually randomly set to begin with, are then adjusted by the network so that the next iteration, or cycle, will produce a closer match between the desired and the actual output. The learning method tries to minimize the current errors of all processing elements. This global error reduction is created over time by continuously modifying the input weights until acceptable network accuracy is reached.

With supervised learning, the artificial neural network must be trained before it becomes useful. Training consists of presenting input and output data to the network. This data is often referred to as the training set. That is, for each input set provided to the system, the corresponding desired output set is provided as well. In most applications, actual data must be used. This training phase can consume a lot of time. In prototype systems, with inadequate processing power, learning can take weeks. This training is considered complete when the neural network reaches a user defined performance level. This level signifies that the network has achieved the desired statistical accuracy as it produces the required outputs for a given sequence of inputs. When no further learning is necessary, the weights are typically frozen for the application. Some network types allow continual training, at a much slower rate, while in operation. This helps a network to adapt to gradually changing conditions.

Training sets need to be fairly large to contain all the needed information if the network is to learn the features and relationships that are important. Not only do the sets have to be large but the training sessions must include a wide variety of data. If the network is trained just one example at a time, all the weights set so meticulously for one fact could be drastically altered in learning the next fact. The previous facts could be forgotten in learning something new. As a result, the system has to learn everything together, finding the best weight settings for the total set of facts. For example, in teaching a system to recognize pixel patterns for the ten digits, if there were twenty examples of each digit, all the examples of the digit seven should not be presented at the same time.

How the input and output data is represented, or encoded, is a major component to successfully instructing a network. Artificial networks only deal with numeric input data. Therefore, the raw data must often be converted from the external environment. Additionally, it is usually necessary to scale the data, or normalize it to the network's paradigm. This pre-processing of real-world stimuli, be they cameras or sensors, into machine readable format is already common for standard computers. Many conditioning techniques which directly apply to artificial neural network implementations are readily available. It is then up to the network designer to find the best data format and matching network architecture for a given application.

After a supervised network performs well on the training data, then it is important to see what it can do with data it has not seen before. If a system does not give reasonable outputs for this test set, the training period is not over. Indeed, this testing is critical to insure that the network has not simply memorized a given set of data but has learned the general patterns involved within an application.

3.2.2 Unsupervised Learning

Unsupervised learning is the great promise of the future. It shouts that computers could someday learn on their own in a true robotic sense. Currently, this learning method is limited to networks known as self-organizing maps. These kinds of networks are not in widespread use. They are basically an academic novelty. Yet, they have shown they can provide a solution in a few instances, proving that their promise is not groundless. They have been proven to be more effective than many algorithmic techniques for numerical aerodynamic flow calculations. They are also being used in the lab where they are split into a front-end network that recognizes short, phoneme-like fragments of speech which are then passed on to a back-end network. The second artificial network recognizes these strings of fragments as words.

This promising field of unsupervised learning is sometimes called self-supervised learning. These networks use no external influences to adjust their weights. Instead, they internally monitor their performance. These networks look for regularities or trends in the input signals, and makes adaptations according to the function of the network. Even without being told whether it's right or wrong, the network still must have some information about how to organize itself. This information is built into the network topology and learning rules.

An unsupervised learning algorithm might emphasize cooperation among clusters of processing elements. In such a scheme, the clusters would work together. If some external input activated any node in the cluster, the cluster's activity as a whole could be increased. Likewise, if external input to nodes in the cluster was decreased, that could have an inhibitory effect on the entire cluster.

Competition between processing elements could also form a basis for learning. Training of competitive clusters could amplify the responses of specific groups to specific stimuli. As such, it would associate those groups with each other and with a specific appropriate response. Normally, when competition for learning is in effect, only the weights belonging to the winning processing element will be updated.

At the present state of the art, unsupervised learning is not well understood and is still the subject of research. This research is currently of interest to the government because military situations often do not have a data set available to train a network until a conflict arises.

3.2.3 Learning Rates

The rate at which ANNs learn depends upon several controllable factors. In selecting the approach there are many trade-offs to consider. Obviously, a slower rate means a lot more time is spent in accomplishing the off-line learning to produce an adequately trained system. With the faster learning rates, however, the network may not be able to make the fine discriminations possible with a system that learns more slowly. Researchers are working on producing the best of both worlds.

Generally, several factors besides time have to be considered when discussing the off-line training task, which is often described as "tiresome." Network complexity, size, paradigm selection, architecture, type of learning rule or rules employed, and desired accuracy must all be considered. These factors play a significant role in determining how long it will take to train a network. Changing any one of these factors may either extend the training time to an unreasonable length or even result in an unacceptable accuracy.

Most learning functions have some provision for a learning rate, or learning constant. Usually this term is positive and between zero and one. If the learning rate is greater than one, it is easy for the learning algorithm to overshoot in correcting the weights, and the network will oscillate. Small values of the learning rate will not correct the current error as quickly, but if small steps are taken in correcting errors, there is a good chance of arriving at the best minimum convergence.

3.2.4 Learning Laws

Many learning laws are in common use. Most of these laws are some sort of variation of the best known and oldest learning law, Hebb's Rule. Research into different learning functions continues as new ideas routinely show up in trade publications. Some researchers have the modeling of biological learning as their main objective. Others are experimenting with adaptations of their perceptions of how nature handles learning. Either way, man's understanding of how neural processing actually works is very limited. Learning is certainly more complex than the simplifications represented by the learning laws currently developed. A few of the major laws are presented as examples.

Hebb's Rule: The first, and undoubtedly the best known, learning rule was introduced by Donald Hebb. The description appeared in his book *The Organization of Behavior* in 1949. His basic rule is: If a neuron receives an input from another neuron, and if both are highly active (mathematically have the same sign), the weight between the neurons should be strengthened.

Hopfield Law: It is similar to Hebb's rule with the exception that it specifies the magnitude of the strengthening or weakening. It states, "if the desired output and the input are both active or both inactive, increment the connection weight by the learning rate, otherwise decrement the weight by the learning rate."

The Delta Rule: This rule is a further variation of Hebb's Rule. It is one of the most commonly used. This rule is based on the simple idea of continuously modifying the strengths of the input connections to reduce the difference (the delta) between the desired output value and the actual output of a processing element. This rule changes the synaptic weights in the way that minimizes the mean squared error of the network. This rule is also referred to as the Widrow-Hoff Learning Rule and the Least Mean Square (LMS) Learning Rule.

The way that the Delta Rule works is that the delta error in the output layer is transformed by the derivative of the transfer function and is then used in the previous neural layer to adjust input connection weights. In other words, this error is back-propagated into previous layers one layer at a time. The process of back-propagating the network errors continues until the first layer is reached. The network type called Feedforward, Back-propagation derives its name from this method of computing the error term.

When using the delta rule, it is important to ensure that the input data set is well randomized. Well ordered or structured presentation of the training set can lead to a network which can not converge to the desired accuracy. If that happens, then the network is incapable of learning the problem.

The Gradient Descent Rule: This rule is similar to the Delta Rule in that the derivative of the transfer function is still used to modify the delta error before it is applied to the connection weights. Here, however, an additional proportional constant tied to the learning rate is appended to the final modifying factor acting upon the weight. This rule is commonly used, even though it converges to a point of stability very slowly.

It has been shown that different learning rates for different layers of a network help the learning process converge faster. In these tests, the learning rates for those layers close to the output were set lower than those layers near the input. This is especially important for applications where the input data is not derived from a strong underlying model.

Kohonen's Learning Law: This procedure, developed by Teuvo Kohonen, was inspired by learning in biological systems. In this procedure, the processing elements compete for the opportunity to learn, or update their weights. The processing element with the largest output is declared the winner and has the capability of inhibiting its competitors as well as exciting its neighbors. Only the winner is permitted an output, and only the winner plus its neighbors are allowed to adjust their connection weights.

Further, the size of the neighborhood can vary during the training period. The usual paradigm is to start with a larger definition of the neighborhood, and narrow in as the training process proceeds. Because the winning element is defined as the one that has the closest match to the input pattern, Kohonen networks model the distribution of the inputs. This is good for statistical or topological modeling of the data and is sometimes referred to as self-organizing maps or self-organizing topologies.

4.0 Network Selection

Because all artificial neural networks are based on the concept of neurons, connections and transfer functions, there is a similarity between the different structures or architectures or neural networks. The majority of the variations stems from the various learning rules and how those rules modify a network's typical topology. The following sections outline some of the most common artificial neural networks. They are organized in very rough categories of application. these categories are not meant to be exclusive, they are merely meant to separate out some of the confusion over networks architectures and their best matches to specific applications. Basically, most applications of neural networks fall into the following five categories:

1. Prediction
2. Classification
3. Data association
4. Data conceptualization
5. Data filtering

Table 3 Network Selector Table

Network Type	Networks	Use for Network
Prediction	• Back-propagation • Delta Bar Delta • Extended Delta Bar Delta • Directed Random Search • Higher Order Neural Networks • Self-organizing map into Back-	Use input values to predict some output (e.g. pick the best stocks in the market, predict weather, identify people with cancer risks etc.)

	propagation	
Classification	• Learning Vector Quantization • Counter-propagation • Probabalistic Neural Networks	Use input values to determine the classification (e.g. is the input the letter A, is the blob of video data a plane and what kind of plane is it)
Data Association	• Hopfield • Boltzmann Machine • Hamming Network • Bidirectional associative Memory • Spation-temporal Pattern Recognition	Like Classification but it also recognizes data that contains errors (e.g. not only identify the characters that were scanned but identify when the scanner isn't working properly)
Data Conceptualization	• Adaptive Resonance Network • Self Organizing Map	Analyze the inputs so that grouping relationships can be inferred (e.g. extract from a database the names of those most likely to buy a particular product)
Data Filtering	• Recirculation	Smooth an input signal (e.g. take the noise out of a telephone signal)

Table 3 shows the differences between these network categories and shows which of the more common network topologies belong to which primary category. This chart is intended as a guide and is not meant to be all inclusive. While there are many other network derivations, this chart only includes the architectures explained within this section of this report. Some of these networks, which have been grouped by application, have been used to solve more than one type of problem. Feedforward back-propagation in particular has been used to solve almost all types of problems and indeed is the most popular for the first four categories. the next five subsections describe these five network types.

4.1 Networks for Prediction

The most common use for neural networks is to project what will most likely happen. There are many applications where prediction can help in setting priorities. For example, the emergency room at a hospital can be a hectic place. to know who needs the most time critical help can enable a more successful operation. Basically, all organizations must establish priorities which govern the allocation of their resources. This projection of the future is what drove the creation of networks of prediction.

4.1.1. Feedforward and Back-Propagation

The feedforward, back-propagation architecture was developed in the early 1970's by several independent sources (Werbor; Parker; Rumelhart, Hinton and Williams). This independent co-development was the result of a proliferation of articles and talks at various conferences which stimulated the entire industry. Currently, this synergistically developed back-propagation architecture is the most popular, effective, and easy to earn model for complex, multi-layered networks. This network is used more than all other combined. It is used in many different types of applications. This architecture has spawned a large class of network types with many different topologies and training methods. Its greatest strength is in non-linear solutions to ill-defined problems.

The typical back-propagation network has an input layer, an output layer, and at least one hidden layer. There is no theoretical limit on the number of hidden layers but typically there is just one or two. Some work has been done which indicates that a minimum of four layers (three hidden layers plus an output layer) are required to solve problems of any complexity. Each layer is fully connected to the succeeding layer, as shown in Figure 10. (Note: all of the drawings of networks in section 5 are from NeuralWare's NeuralWorks Professional II/Plus artificial neural network development tool.)

The in and out layers indicate the flow of information during recall. Recall is the process of putting input data into a trained network and receiving the answer. Back-propagation is not used during recall, but only when the network is learning a training set.

The number of layers and the number of processing element per layer are important decisions. These parameters to a feedforward, back-propagation topology are also the most ethereal. They are the $^{3}art^{2}$ of the network designer. There is no quantifiable, best answer to the layout of the network for any particular application. There are only general rules picked up over time and followed by most researchers and engineers applying this architecture of their problems.

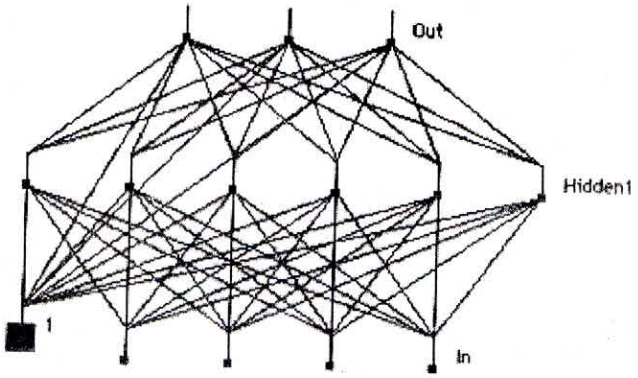


Figure 10: Feedforward Back-propagation Network

Rule One: As the complexity in the relationship between the input data and the desired output increases, then the number of the processing elements in the hidden layer should also increase.

Rule Two: If the process being modeled is separable into multiple stages, then additional hidden layer(s) may be required. If the process is not separable into stages, then additional layers may simply enable memorization and not a true general solution.

Rule Three: The amount of training data available sets an upper bound for the number of processing elements in the hidden layers. To calculate this upper bound, use the number of input output pair examples in the training set and divide that number by the total number of input and output processing elements in the network. Then divide that result again by a scaling factor between five and ten. Larger scaling factors are used for relatively noisy data. Extremely noisy data may require a factor of twenty or even fifty, while very clean input data with an exact relationship to the output might drop the factor to around two. It is important that the hidden layers have few processing elements. Too many artificial neurons and the training set will be memorized. If that happens then no generalization of the data trends will occur, making the network useless on new data sets.

Once the above rules have been used to create a network, the process of teaching begins. This teaching process for a feedforward network normally uses some variant of the Delta Rule, which starts with the calculated difference between the actual outputs and the desired outputs. Using this error, connection weights are increased in proportion to the error times a scaling factor for global accuracy. Doing this for an individual node means that the inputs, the output, and the desired output all have to be present at the same processing element. The complex part of this learning mechanism is for the system to determine which input contributed the most to an incorrect output and how does that element get changed to correct the error. An inactive node would not contribute to the error and would have no need to change its weights.

To solve this problem, training inputs are applied to the input layer of the network, and desired outputs are compared at the output layer. During the learning process, a forward sweep is made through the network, and the output of each element is computed layer by layer. The difference between the output of the final layer and the desired output is back-propagated to the previous layer(s), usually modified by the derivative of the transfer function, and the connection weights are normally adjusted using the Delta Rule. This process proceeds for the previous layer(s) until the input layer is reached.

There are many variations to the learning rules for back-propagation network. Different error functions, transfer functions, and even the modifying method of the derivative of the transfer function can be used. The concept of [3]momentum error[2] was introduced to allow for more prompt learning while minimizing unstable behavior. Here, the error function, or delta weight equation, is modified so that a portion of the previous delta weight is fed through to the current delta weight. This acts, in engineering terms, as a low-pass filter on the delta weight terms since general trends are reinforced whereas oscillatory behavior is canceled out. This allows a low, normally slower, learning coefficient to be used, but creates faster learning.

Another technique that has an effect on convergence speed is to only update the weights after many pairs of inputs and their desired outputs are presented to the network, rather than after every presentation. This is referred to as cumulative back-propagation because

the delta weights are not accumulated until the complete set of pairs is presented. The number of input-output pairs that are presented during the accumulation is referred to as an [3]epoch[2]. This epoch may correspond either to the complete set of training pairs or to a subset.

There are limitations to the feedforward, back-propagation architecture. Back-propagation requires lots of supervised training, with lots of input-output examples. Additionally, the internal mapping procedures are not well understood, and there is no guarantee that the system will converge to an acceptable solution. At times, the learning gets stuck in a local minima, limiting the best solution. This occurs when the network systems finds an error that is lower than the surrounding possibilities but does not finally get to the smallest possible error. Many learning applications add a term to the computations to bump or jog the weights past shallow barriers and find the actual minimum rather than a temporary error pocket.

Typical feedforward, back-propagation applications include speech synthesis from text, robot arms, evaluation of bank loans, image processing, knowledge representation, forecasting and prediction, and multi-target tracking. Each month more back-propagation solutions are announced in the trade journals.

4.1.2 Delta Bar Delta

The delta bar delta network utilizes the same architecture as a back-propagation network. The difference of delta bar delta lies in its unique algorithmic method of learning. Delta bar delta was developed by Robert Jacobs to improve the learning rate of standard feedforward, back-propagation networks.

As outlined above, the back-propagation procedure is based on a steepest descent approach which minimizes the network[1]s prediction error during the process where the connection weights to each artificial neuron are changed. The standard learning rates are applied on a layer by layer basis and the momentum term is usually assigned globally. Some back-propagation approaches allow the learning rates to gradually decrease as large quantities of training sets pass through the network. Although this method is successful in solving many applications, the convergence rate of the procedure is still too slow to be used on some practical problems.

The delta bar delta paradigm uses a learning method where each weight has its own self-adapting coefficient. It also does not use the momentum factor of the back-propagation architecture. The remaining operations of the network, such as feedforward recall, are identical to the normal back-propagation architecture. Delta bar delta is a [3]heuristic[2] approach to training artificial networks. What that means is that past error values can be used to infer future calculated error values. Knowing the probable errors enables the system to take intelligence steps in adjusting the weights. However, this process is complicated in that empirical evidence suggests that each weight may have quite different effects on the overall error. Jacobs then suggested the common sense notion that the back-propagation learning rules should account for these variations in the effect on the overall error. In other words, every connection weight of a network should have its own learning rate. The claim is that the step size appropriate for one connection weight may not be appropriate for all weights in that layer. Further, these learning rates should be allowed to vary over time. by assigning a learning rate to each connection and permitting

this learning rate to change continuously over time, more degrees of freedom are introduced to reduce the time to convergence.

Rules which directly apply to this algorithm are straight forward and easy to implement. Each connection weight has its own learning rate. These learning rates are varied based on the current error information found with standard back-propagation. When the connection weight changes, if the local error has the same sign for several consecutive time steps, the learning rate for that connection is linearly increased. Incrementing linearly prevents the learning rates from becoming too large too fast. When the local error changes signs frequently, the learning rate is decreased geometrically. Decrementing geometrically ensures that the connection learning rates are always positive. Further, they can be decreased more rapidly in regions where the change in error is large.

By permitting different learning rates for each connection weight in a network, a steepest descent search (in the direction of the negative gradient) is no longer being performed. Instead, the connection weights are updated on the basis of the partial derivatives of the error with respect to the weight itself. It is also based on an estimate of the [3]curvature of the error surface[2] in the vicinity of the current point weight value. Additionally, the weight changes satisfy the locality constraint, that is, they require information only from the processing elements to which they are connected.

4.1.3 Extended Delta Bar Delta

Ali Minai and Ron Williams developed the extended delta bar delta algorithm as a natural outgrowth from Jacob's work. Here, they enhance the delta bar delta by applying an exponential decay to the learning rate increase, add the momentum component back in, and put a cap on the learning rate and momentum coefficient. As discussed in the section on back-propagation, momentum is a factor used to smooth the learning rate. It is a term added to the standard weight change which is proportional to the previous weight change. In this way, good general trends are reinforced, and oscillations are dampened.

The learning rate and the momentum rate for each weight have separate constants controlling their increase and decrease. Once again, the sign of the current error is used to indicate whether an increase or decrease is appropriate. The adjustment for decrease is identical in form to that of Delta Bar Delta. However, the learning rate and momentum rate increases are modified to be exponentially decreasing functions of the magnitude of the weighted gradient components. Thus, greater increases will be applied in areas of small slope or curvature than in areas of high curvature. This is a partial solution to the jump problem of delta bar delta.

To take a step further to prevent wild jumps and oscillations in the weights, ceilings are placed on the individual connection learning rates and momentum rates. And finally, a memory with a recovery feature is built into the algorithm. When in use, after each epoch presentation of the training data, the accumulated error is evaluated. If the error is less than the previous minimum error, the weights are saved in memory as the current best. A tolerance parameter controls the recovery phase. Specifically, if the current error exceeds the minimum previous error, modified by the tolerance parameter, than all connection weight values revert stochastically to the stored best set of weights in memory. Furthermore, the learning and momentum rates are decreased to begin the recovery process.

4.1.4 Directed Random Search

The previous architectures were all based on learning rules, or paradigms, which are based on calculus. Those paradigms use a gradient descent technique to adjust each of the weights. The architecture of the directed random search, however, uses a standard feedforward recall structure which is not based on back-propagation. Instead, the directed random search adjusts the weights randomly. To provide some order to this process a direction component is added to the random step which insures that the weights tend toward a previously successful search direction. All processing elements are influenced individually.

This random search paradigm has several important features. Basically, it is fast and easy to use if the problem is well understood and relatively small. The reason that the problem has to be well understood is that the best results occur when the initial weights, the first guesses, are within close proximity to the best weights. It is fast because the algorithm cycles through its training much more quickly than calculus-bases techniques (i.e., the delta rule and its variations), since no error terms are computed for the intermediate processing elements. Only the output error is calculated. This learning rule is easy to use because there are only two key parameters associated with it. But the problem needs to result in a small network because if the number of connections becomes high, then the training process becomes long and cumbersome.

To facilitate keeping the weights within the compact region where the algorithm works best, an upper bound is required on the weight's magnitude. Yet, by setting the weight's bounds reasonably high, the network is still allowed to seek what is not exactly known - the true global optimum. The second key parameter to this learning rule involves the initial variance of the random distribution of the weights. In most of the commercial packages there is a vendor recommended number for this initial variance parameter. Yet, the setting of this number is not all that important as the self-adjusting feature of the directed random search has proven to be robust over a wide range of initial variances.

There are four key components to a random search network. They are the random step, the reversal step, a directed component, and a self-adjusting variance.

Random Step: A random value is added to each weight. Then, the entire training set is run through the network, producing a "prediction error." If this new total training set error is less than the previous best prediction error, the current weight values (which include the random step) becomes the new set of "best" weights. The current prediction error is then saved as the new, best prediction error.

Reversal Step: If the random step's results are worse than the previous best, then the same random value is subtracted from the original weight value. This produces a set of weights that is in the opposite direction to the previous random step. If the total "prediction error" is less than the previous best error, the current weight values of the reversal step are stored as the best weights. The current prediction error is also saved as the new, best prediction error. If both the forward and reverse steps fail, a completely new set of random values are added to the best weights and the process is then begun again.

Directed Component: To add in convergence, a set of directed components is created, based on the outcomes of the forward and reversal steps. These directed components

reflect the history of success or failure for the previous random steps. The directed components, which are initialized to zero, are added to the random components at each step in the procedure. Directed components provide a "common sense, let's go this way" element to the search. It has been found that the addition of these directed components provide a dramatic performance improvement to convergence.

Self-adjusting Variance: An initial variance parameter is specified to control the initial size (or length) of the random steps which are added to the weights. An adaptive mechanism changes the variance parameter based on the current relative success rate or failure rate. The learning rule assumes that the current size of the steps for the weights is in the right direction if it records several consecutive successes, and it then expands to try even larger steps. Conversely, if it detects several consecutive failures it contracts the variance to reduce the step size.

For small to moderately sized networks, a directed random search produces good solutions in a reasonable amount of time. The training is automatic, requiring little, if any, user interaction. The number of connection weights imposes a practical limit on the size of a problem that this learning algorithm can effectively solve. If a network has more than 200 connection weights, a directed random search can require a relatively long training time and still end up yielding an acceptable solution.

4.1.5 Higher-order Neural Network or Functional-link Network

Either name is given to neural networks which expand the standard feedforward, back-propagation architecture to include nodes at the input layer which provide the network with a more complete understanding of the input. Basically, the inputs are transformed in a well understood mathematical way so that the network does not have to learn some basic math functions. These functions do enhance the network's understanding of a given problem. These mathematical functions transform the inputs via higher-order functions such as squares, cubes, or sines. It is from the very name of these functions, higher-order or functionally linked mappings, that the two names for this same concept were derived.

This technique has been shown to dramatically improve the learning rates of some applications. An additional advantage to this extension of back-propagation is that these higher order functions can be applied to other derivations - delta bar delta, extended delta bar delta, or any other enhanced feedforward, back-propagation networks.

There are two basic ways of adding additional input nodes. First, the cross-products of the input terms can be added into the model. This is also called the output product or tensor model, where each component of the input pattern multiplies the entire input pattern vector. A reasonable way to do this is to add all interaction terms between input values. For example, for a back-propagation network with three inputs (A, B and C), the cross-products would include: AA, BB, CC, AB, AC, and BC. This example adds second-order terms to the input structure of the network. Third-order terms, such as ABC, could also be added.

The second method for adding additional input nodes is the functional expansion of the base inputs. Thus, a back-propagation model with A, B and C might be transformed into a higher-order neural network model with inputs: A, B, C, SIN(A), COS(B), LOG(C), MAX(A,B,C), etc. In this model, input variables are individually acted upon by appropriate functions. Many different functions can be used. The overall effect is to

provide the network with an enhanced representation of the input. It is even possible to combine the tensor and functional expansion models together.

No new information is added, but the representation of the inputs is enhanced. Higher-order representation of the input data can make the network easier to train. The joint or functional activations become directly available to the model. In some cases, a hidden layer is no longer needed. However, there are limitations to the network model. Many more input nodes must be processed to use the transformations of the original inputs. With higher-order systems, the problem is exacerbated. Yet, because of the finite processing time of computers, it is important that the inputs are not expanded more than is needed to get an accurate solution.

Functional-link networks were developed by Yoh-Han Pao and are documented in his book, *Adaptive Pattern Recognition and Neural Networks*. Pao draws a distinction between truly adding higher order terms in the sense that some of these terms represent joint activations versus functional expansion which increases the dimension of the representation space without adding joint activations. While most developers recognize the difference, researchers typically treat these two aspects in the same way. Pao has been awarded a patent for the functional-link network, so its commercial use may require royalty licensing.

4.1.6 Self-Organizing Map into Back-Propagation

A hybrid network uses a self-organizing map to conceptually separate the data before that data is used in the normal back-propagation manner. This map helps to visualize topologies and hierarchical structures of higher-order input spaces before they are entered into the feedforward, back-propagation network. The change to the input is similar to having an automatic functional-link input structure. This self-organizing map trains in an unsupervised manner. The rest of the network goes through its normal supervised training. The self-organizing map, and its unique approach to learning, is described in section 5.4.2

4.2 Networks for Classification

The previous section describes networks that attempt to make projections of the future. But understanding trends and what impacts those trends might have is only one of several types of applications. The second class of applications is classification. A network that can classify could be used in the medical industry to process both lab results and doctor-recorded patience symptoms to determine the most likely disease. Other applications can separate the "tire kicker" inquiries from the requests for information from real buyers.

4.2.1 Learning Vector Quantization

This network topology was originally suggested by Tuevo Kohonen in the mid 80's, well after his original work in self-organizing maps. Both this network and self-organizing maps are based on the Kohonen layer, which is capable of sorting items into appropriate categories of similar objects. Specifically, Learning Vector Quantization is a artificial neural network model used both for classification and image segmentation problems.

Topologically, the network contains an input layer, a single Kohonen layer and an output layer. An example network is shown in Figure 11. The output layer has as many processing elements as there are distinct categories, or classes. The Kohonen layer has a number of processing elements grouped for each of these classes. The number of processing elements per class depends upon the complexity of the input-output relationship. Usually, each class will have the same number of elements throughout the layer. It is the Kohonen layer that learns and performs relational classifications with the aid of a training set. This network uses supervised learning rules. However, these rules vary significantly from the back-propagation rules. To optimize the learning and recall functions, the input layer should contain only one processing element for each separable input parameter. Higher-order input structures could also be used.

Learning Vector Quantization classifies its input data into groupings that it determines. Essentially, it maps an n-dimensional space into an m-dimensional space. That is it takes n inputs and produces m outputs. The networks can be trained to classify inputs while preserving the inherent topology of the training set. Topology preserving maps preserve nearest neighbor relationships in the training set such that input patterns which have not been previously learned will be categorized by their nearest neighbors in the training data.

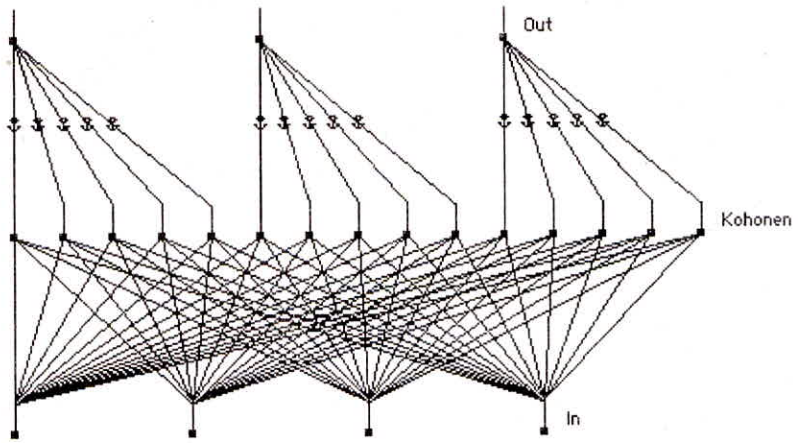


Figure 11: Learning Vector Quantization Network

In the training mode, this supervised network uses the Kohonen layer such that the distance of a training vector to each processing element is computed and the nearest processing element is declared the winner. There is only one winner for the whole layer. The winner will enable only one output processing element to fire, announcing the class or category the input vector belonged to. If the winning element is in the expected class of the training vector, it is reinforced toward the training vector. If the winning element is not in the class of the training vector, the connection weights entering the processing element are moved away from the training vector. This later operation is referred to as repulsion. During this training process, individual processing elements assigned to a particular class migrate to the region associated with their specific class.

During the recall mode, the distance of an input vector to each processing element is computed and again the nearest element is declared the winner. That in turn generates one output, signifying a particular class found by the network.

There are some shortcomings with the Learning Vector Quantization architecture. Obviously, for complex classification problems with similar objects or input vectors, the network requires a large Kohonen layer with many processing elements per class. This can be overcome with selectively better choices for, or higher-order representation of, the input parameters.

The learning mechanism has some weaknesses which have been addressed by variants to the paradigm. Normally these variants are applied at different phases of the learning process. They imbue a conscience mechanism, a boundary adjustment algorithm, and an attraction function at different points while training the network.

The simple form of the Learning Vector Quantization network suffers from the defect that some processing elements tend to win too often while others, in effect, do nothing. This particularly happens when the processing elements begin far from the training vectors. Here, some elements are drawn in close very quickly and the others remain permanently far away. To alleviate this problem, a conscience mechanism is added so that a processing element which wins too often develops a "guilty conscience" and is penalized. The actual conscience mechanism is a distance bias which is added to each processing element. This distance bias is proportional to the difference between the win frequency of an element and the average processing element win frequency. As the network progresses along its learning curve, this bias proportionality factors needs to be decreased.

The boundary adjustment algorithm is used to refine a solution once a relatively good solution has been found. This algorithm affects the cases when the winning processing element is in the wrong class and the second best processing element is in the right class. A further limitation is that the training vector must be near the midpoint of space joining these two processing elements. The winning wrong processing element is moved away from the training vector and the second place element is moved toward the training vector. This procedure refines the boundary between regions where poor classifications commonly occur.

In the early training of the Learning Vector Quantization network, it is some times desirable to turn off the repulsion. The winning processing element is only moved toward the training vector if the training vector and the winning processing element are in the same class. This option is particularly helpful when a processing element must move across a region having a different class in order to reach the region where it is needed.

4.2.2 Counter-propagation Network

Robert Hecht-Nielsen developed the counter-propagation network as a means to combine an unsupervised Kohonen layer with a teachable output layer. This is yet another topology to synthesize complex classification problems, while trying to minimize the number of processing elements and training time. The operation for the counter-propagation network is similar to that of the Learning Vector Quantization network in that the middle Kohonen layer acts as an adaptive look-up table, finding the closest fit to an input stimulus and outputting its equivalent mapping.

The first counter-propagation network consisted of a bi-directional mapping between the input and output layers. In essence, while data is presented to the input layer to generate a classification pattern on the output layer, the output layer in turn would accept an additional input vector and generate an output classification on the network's input layer. The network got its name from this counter-posing flow of information through its structure. Most developers use a uni-flow variant of this formal representation of counter-propagation. In other words. there is only one feedforward path from input layer to output layer.

An example network is shown in Figure 12. The uni-directional counter-propagation network has three layers. If the inputs are not already normalized before they enter the network., a fourth layer is sometimes added. The main layers include an input buffer layer, a self-organizing Kohonen layer, and an output layer which uses the Delta Rule to modify its incoming connection weights. Sometimes this layer is called a Grossberg Outstar layer.

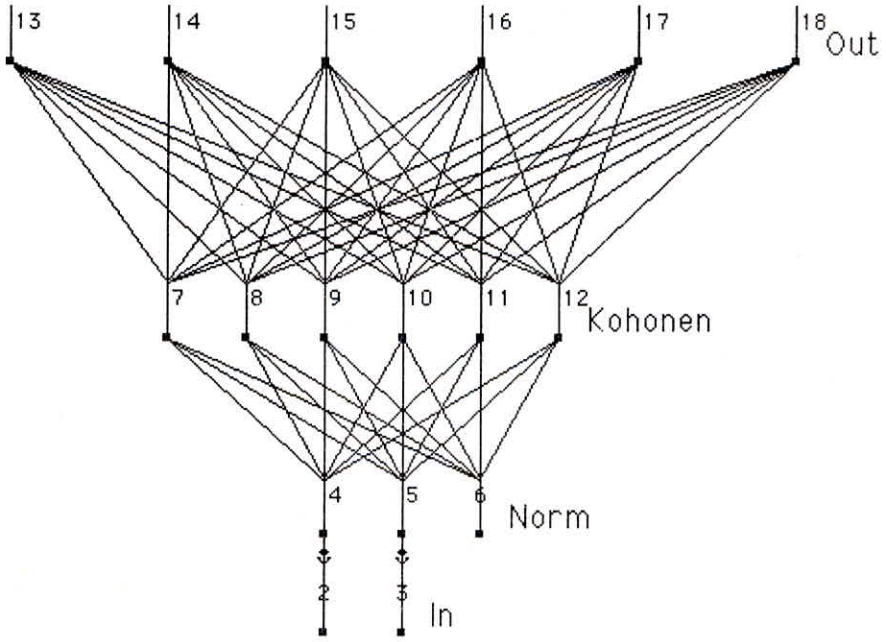


Figure 12: Counter-propagation Network

The size of the input layer depends upon how many separable parameters define the problem. With too few, the network may not generalize sufficiently. With too many, the processing time takes too long.

For the network to operate properly, the input vector must be normalized. This means that for every combination of input values, the total "length" of the input vector must add up to one. This can be done with a preprocessor, before the data is entered into the counter-propagation network. Or, a normalization layer can be added between the input and Kohonen layers. The normalization layer requires one processing element for each input, plus one more for a balancing element. This layer modifies the input set before going to the Kohonen layer to guarantee that all input sets combine to the same total.

Normalization of the inputs is necessary to insure that the Kohonen layer finds the correct class for the problem. Without normalization, larger input vectors bias many of the Kohonen processing elements such that weaker value input sets cannot be properly classified. Because of the competitive nature of the Kohonen layer, the larger value input vectors overpower the smaller vectors. Counter-propagation 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 of this report. This layer acts as a nearest neighbor classifier in that the processing elements 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 needs to be at least as many processing elements in the Kohonen layer as output classes. The Kohonen layer usually has many more elements than classes simply because additional processing elements provide a finer resolution between similar objects.

The output layer for counter-propagation is basically made up of processing elements 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 effected.

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

There is a problem which 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 processing element in the Kohonen layer to learn to take responsibility for two or more training inputs which belong to different classes. When this happens, the output of the network will be ambiguous for any inputs which activate this processing element. To alleviate this problem, the processing elements in the Kohonen layer could be pre-conditioned to learn only about a particular class.

4.2.3 Probabilistic Neural Network

The probabilistic neural network was developed by Donald Specht. His network architecture was first presented in two papers, *Probabilistic Neural Networks for Classification, Mapping or Associative Memory* and *Probabilistic Neural Networks*, released in 1988 and 1990, respectively. This network provides a general solution to pattern classification problems by following an approach developed in statistics, called Bayesian classifiers. Bayes theory, developed in the 1950's, takes into account the relative likelihood of events and uses a priori information to improve prediction. The network paradigm also uses Parzen Estimators which were developed to construct the probability density functions required by Bayes theory.

The probabilistic neural network uses a supervised training set to develop distribution functions within a pattern layer. These functions, in the recall mode, are used to estimate the likelihood of an input feature vector being part of a learned category, or class. The learned patterns can also be combined, or weighted, with the a priori probability, also called the relative frequency, of each category to determine the most likely class for a given input vector. If the relative frequency of the categories is unknown, then all categories can be assumed to be equally likely and the determination of category is solely based on the closeness of the input feature vector to the distribution function of a class.

An example of a probabilistic neural network is shown in Figure 13. This network has three layers. The network contains an input layer which has as many elements 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 processing element. And finally, the network contains an output layer, called the summation layer, which has as many processing elements as there are classes to be recognized. Each element in this layer combines via processing elements within the pattern layer which relate to the same class and prepares that category for output. Sometimes a fourth layer is added to normalize the input vector, if the inputs are not already normalized before they enter the network. As with the counter-propagation network, the input vector must be normalized to provided proper object separation in the pattern layer.

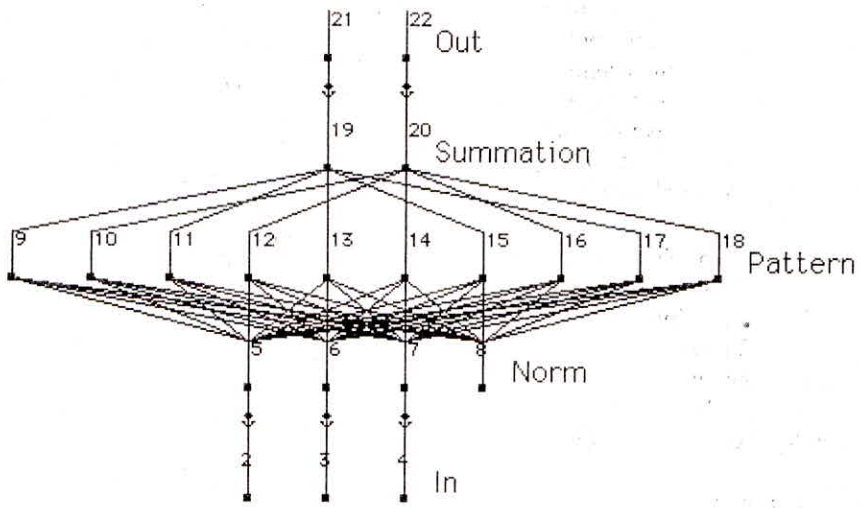


Figure 13: Probabilistic Neural Network

As mentioned earlier, the pattern layer represents a neural implementation of a version of a Bayes classifier, where the class dependent probability density functions are approximated using a Parzen estimator. This approach provides an optimum pattern classifier in terms of minimizing the expected risk of wrongly classifying an object. With the estimator, the approach gets closer to the true underlying class density functions as the number of training samples increases, so long as the training set is an adequate representation of the class distinctions.

In the pattern layer, there is a processing element for each input vector in the training set. Normally, there are equal amounts of processing elements for each output class. Otherwise, one or more classes may be skewed incorrectly and the network will generate poor results. Each processing element in the pattern layer is trained once. An element is trained to generate a high output value when an input vector matches the training vector. The training function may include a global smoothing factor to better generalize classification results. In any case, the training vectors do not have to be in any special order in the training set, since the category of a particular vector is specified by the desired output of the input. The learning function simply selects the first untrained processing element in the correct output class and modifies its weights to match the training vector.

The pattern layer operates competitively, where only the highest match to an input vector wins and generates an output. In this way, only one classification category is generated for any given input vector. If the input does not relate well to any patterns programmed into the pattern layer, no output is generated.

The Parzen estimation can be added to the pattern layer to fine tune the classification of objects, This is done by adding the frequency of occurrence for each training pattern built into a processing element. Basically, the probability distribution of occurrence for each example in a class is multiplied into its respective training node. In this way, a more accurate expectation of an object is added to the features which make it recognizable as a class member.

Training of the probabilistic neural network is much simpler than with back-propagation. However, the pattern layer can be quite huge if the distinction between categories is varied and at the same time quite similar is special areas. There are many proponents for this type of network, since the groundwork for optimization is founded in well known, classical mathematics.

4.3 Networks for Data Association

The previous class of networks, classification, is related to networks for data association. In data association, classification is still done. For example, a character reader will classify each of its scanned inputs. However, an additional element exists for most applications. That element is the fact that some data is simply erroneous. Credit card applications might have been rendered unreadable by water stains. The scanner might have lost its light source. The card itself might have been filled out by a five year old. Networks for data association recognize these occurrances as simply bad data and they recognize that this bad data can span all classifications.

4.3.1 Hopfield Network

John Hopfield first presented his cross-bar associative network in 1982 at the National Academy of Sciences. In honor of Hopfield's success and his championing of neural networks in general, this network paradigm is usually referred to as a Hopfield Network. The network can be conceptualized in terms of its energy and the physics of dynamic systems. A processing element in the Hopfield layer, will change state only if the overall "energy" of the state space is reduced. In other words, the state of a processing element will vary depending whether the change will reduce the overall "frustration level" of the network. Primary applications for this sort of network have included associative, or

content-addressable, memories and a whole set of optimization problems, such as the combinatoric best route for a traveling salesman.

The Figure 14 outlines a basic Hopfield network. The original network had each processing element operate in a binary format. This is where the elements compute the weighted sum of the inputs and quantize the output to a zero or one. These restrictions were later relaxed, in that the paradigm can use a sigmoid based transfer function for finer class distinction. Hopfield himself showed that the resulting network is equivalent to the original network designed in 1982.

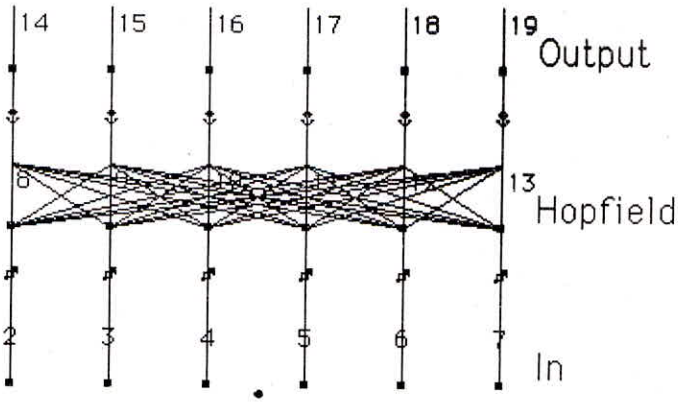


Figure 14: Hopfield Network

The Hopfield network uses three layers; an input buffer, a Hopfield layer, and an output layer. Each layer has the same number of processing elements. The inputs of the Hopfield layer are connected to the outputs of the corresponding processing elements in the input buffer layer through variable connection weights. The outputs of the Hopfield layer are connected back to the inputs of every other processing element except itself. They are also connected to the corresponding elements in the output layer. In normal recall operation, the network applies the data from the input layer through the learned connection weights to the Hopfield layer. The Hopfield layer oscillates until some fixed number of cycles have been completed, and the current state of that layer is passed on to the output layer. This state matches a pattern already programmed into the network.

The learning of a Hopfield network requires that a training pattern be impressed on both the input and output layers simultaneously. The recursive nature of the Hopfield layer provides a means of adjusting all of the connection weights. The learning rule is the Hopfield Law, where connections are increased when both the input and output of an Hopfield element are the same and the connection weights are decreased if the output does not match the input. Obviously, any non-binary implementation of the network must have a threshold mechanism in the transfer function, or matching input-output pairs could be too rare to train the network properly.

The Hopfield network has two major limitations when used as a content addressable memory. First, the number of patterns that can be stored and accurately recalled is severely limited. If too many patterns are stored, the network may converge to a novel spurious pattern different from all programmed patterns. Or, it may not converge at all.

The storage capacity limit for the network is approximately fifteen percent of the number of processing elements in the Hopfield layer. The second limitation of the paradigm is that the Hopfield layer may become unstable if the common patterns it shares are too similar. Here an example pattern is considered unstable if it is applied at time zero and the network converges to some other pattern from the training set. This problem can be minimized by modifying the pattern set to be more orthogonal with each other.

4.3.2 Boltzmann Machine

The Boltzmann machine is similar in function and operation to the Hopfield network with the addition of using a simulated annealing technique when determining the original pattern. The Boltzmann machine incorporates the concept of simulated annealing to search the pattern layer's state space for a global minimum. Because of this, the machine will gravitate to an improved set of values over time as data iterates through the system.

Ackley, Hinton, and Sejnowski developed the Boltzmann learning rule in 1985. Like the Hopfield network, the Boltzmann machine has an associated state space energy based upon the connection weights in the pattern layer. The processes of learning a training set full of patterns involves the minimization of this state space energy. Because of this, the machine will gravitate to an improved set of values for the connection weights while data iterates through the system.

The Boltzmann machine requires a simulated annealing schedule, which is added to the learning process of the network. Just as in physical annealing, temperatures start at higher values and decreases over time. The increased temperature adds an increased noise factor into each processing element in the pattern layer. Typically, the final temperature is zero. If the network fails to settle properly, adding more iterations at lower temperatures may help to get to a optimum solution.

A Boltzmann machine learning at high temperature behaves much like a random model and at low temperatures it behaves like a deterministic model. Because of the random component in annealed learning, a processing element can sometimes assume a new state value that increases rather than decreases the overall energy of the system. This mimics physical annealing and is helpful in escaping local minima and moving toward a global minimum.

As with the Hopfield network, once a set of patterns are learned, a partial pattern can be presented to the network and it will complete the missing information. The limitation on the number of classes, being less than fifteen percent of the total processing elements in the pattern layer, still applies.

4.3.3 Hamming Network

The Hamming network is an extension of the Hopfield network in that it adds a maximum likelihood classifier to the frond end. This network was developed by Richard Lippman in the mid 1980's. The Hamming network implements a classifier based upon least error for binary input vectors, where the error is defined by the Hamming distance. The Hamming distance is defined as the number of bits which differ between two corresponding, fixed-length input vectors. One input vector is the noiseless example pattern, the other is a pattern corrupted by real-world events. In this network architecture,

the output categories are defined by a noiseless, pattern-filled training set. In the recall mode any incoming input vectors are then assigned to the category for which the distance between the example input vectors and the current input vector is minimum.

The Hamming network has three layers. There is an example network shown in Figure 15. The network uses an input layer with as many nodes as there are separate binary features. It has a category layer, which is the Hopfield layer, with as many nodes as there are categories, or classes. This differs significantly from the formal Hopfield architecture, which has as many nodes in the middle layer as there are input nodes. And finally, there is an output layer which matches the number of nodes in the category layer.

The network is a simple feedforward architecture with the input layer fully connected to the category layer. Each processing element in the category layer is connected back to every other element in that same layer, as well as to a direct connection to the output processing element. The output from the category layer to the output layer is done through competition.

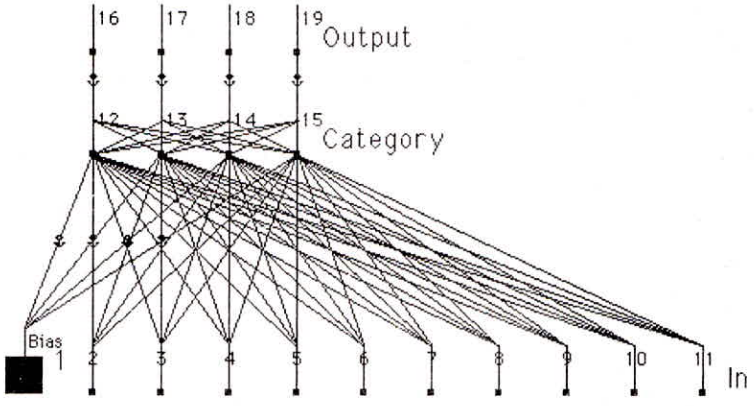


Figure 15: Hamming Network

The learning of a Hamming network is similar to the Hopfield methodology in that it requires a single-pass training set. However, in this supervised paradigm, the desired training pattern is impressed upon the input layer while the desired class to which the vector belongs is impressed upon the output layer. Here the output contains only the category output to which the input vector belongs. Again, the recursive nature of the Hopfield layer provides a means of adjusting all connection weights.

The connection weights are first set in the input to category layer such that the matching scores generated by the outputs of the category processing elements are equal to the number of input nodes minus the Hamming distances to the example input vectors. These matching scores range from zero to the total number of input elements and are highest for those input vectors which best match the learned patterns. The category layer's recursive connection weights are trained in the same manner as in the Hopfield network. In normal feedforward operation an input vector is applied to the input layer and must be presented long enough to allow the matching score outputs of the lower input to category subnet to settle. This will initialize the input to the Hopfield function in the category layer and

allow that portion of the subnet to find the closest class to which the input vector belongs. This layer is competitive, so only one output is enabled at a time.

The Hamming network has a number of advantages over the Hopfield network. It implements the optimum minimum error classifier when input bit errors are random and independent. So, the Hopfield with its random set up nature can only be as good a solution as the Hamming, or it can be worse. Fewer processing elements are required for the Hamming solution, since the middle layer only requires one element per category, instead of an element for each input node. And finally, the Hamming network does not suffer from spurious classifications which may occur in the Hopfield network. All in all, the Hamming network is both faster and more accurate than the Hopfield network.

4.3.4 Bi-directional Associative Memory

This network model was developed by Bart Kosko and again generalizes the Hopfield model. A set of paired patterns are learned with the patterns represented as bipolar vectors. Like the Hopfield, when a noisy version of one pattern is presented, the closest pattern associated with it is determined.

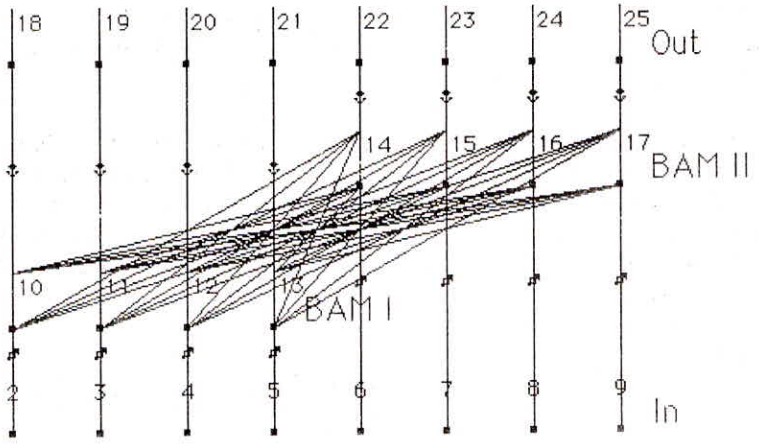


Figure 16: Bi-directional Associative Memory

A diagram of an example bi-directional associative memory is shown in Figure 16. It has as many inputs as output processing nodes. The two hidden layers are made up of two separate associated memories and represent the size of two input vectors. The two lengths need not be the same, although this example shows identical input vector lengths of four each. The middle layers are fully connected to each other. The input and output layers are for implementation purposes the means to enter and retrieve information from the network. Kosko original work targeted the bi-directional associative memory layers for optical processing, which would not need formal input and output structures.

The middle layers are designed to store associated pairs of vectors. When a noisy pattern vector is impressed upon the input, the middle layers oscillate back and forth until a stable equilibrium state is reached. This state, providing the network is not over trained, corresponds to the closest learned association and will generate the original training pattern on the output. Like the Hopfield network, the bi-directional associative memory

network is susceptible to incorrectly finding a trained pattern when complements of the training set are used as the unknown input vector.

4.3.5 Spatio-Temporal Pattern Recognition (Avalanche)

This network as shown in Figure 17 came out of Stephen Grossberg's work in the early 1970's. It basically was developed to explain certain cognitive processes for recognizing time varying sequences of events. In his work at the time he called this network paradigm an "Avalanche" network. Robert Hecht-Nielsen became interested in how this network could be applied to engineering applications. The outcome was the spatio-temporal pattern recognition network. Here, specific patterns, for example audio signals, are memorized and then used as a basis to classify incoming repetitive signals. This network has parameters which allow tuning to accommodate detection of time varying signals.

Here is a global bias term attached to each processing element. This term is used to normalize the overall activity in the network. It sets a variable threshold against which processing elements must compete, and insures that the best match wins. The learning paradigm for the network uses a variant of the Kohonen rule and adds a time varying component to the learning function, called the attack function. This function is also used in the recall mode, to provide latency to the history of signals passing through the network.

The primary application of spatio-temporal pattern networks appears to be in the area of recognizing repetitive audio signals. One group in General Dynamics has applied this network to classify types of ships based on the sounds their propellers make. Another characteristic of the network is that because of the slow decay of the attack function, even though the periodicity of the input signal varied by as much as a factor of two, the network was still able to correctly classify the propeller signals.

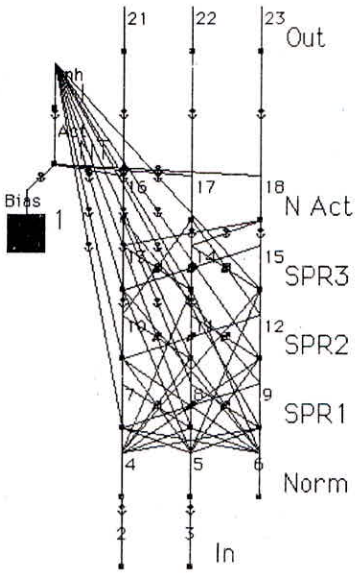


Figure 17: Spatio-temporal Pattern Network

4.4 Networks for Data Conceptualization

Another network type is data conceptualization. In many applications data is not just classified, for not all applications involve data that can fit within a class, not all applications read characters or identify diseases. Some applications need to group data that may, or may not be, clearly definable. An example of this is in the processing of a data base for a mailing list of potential customers. Customers might exist within all classifications, yet they might be concentrated within a certain age group and certain income levels. Also, in real life, other information might stretch and twist the region which contains the vast majority of potential buyers. This process is data conceptualization. It simply tries to identify a group as best as it can.

4.4.1 Adaptive Resonance Network

Developed by Stephen Grossberg in the mid 1970's, the network creates categories of input data based on adaptive resonance. The topology is biologically plausible and uses an unsupervised learning function. It analyses behaviorally significant input data and detects possible features or classifies patterns in the input vector.

This network was the basis for many other network paradigms, such as counter-propagation and bi-directional associative memory networks. The heart of the adaptive resonance network consists of two highly interconnected layers of processing elements located between an input and output layer. Each input pattern to the lower resonance layer will induce an expected pattern to be sent from the upper layer to the lower layer to influence the next input. This creates a "resonance" between the lower and upper layers to facilitate network adaption of patterns.

The network is normally used in biological modelling, however, some engineering applications do exist. The major limitation to the network architecture is its noise susceptibility. Even a small amount of noise on the input vector confuses the pattern matching capabilities of a trained network. The adaptive resonance theory network topology is protected by a patent held by the University of Boston.

4.4.2 Self-Organizing Map

Developed by Teuvo Kohonen in the early 1980's, the input data is projected to a two-dimensional layer which preserves order, compact sparse data, and spreads out dense data. In other words, if two input vectors are close, they will be mapped to processing elements that are close together in the two-dimensional Kohonen layer that represents the features or clusters of the input data. Here, the processing elements represent a two-dimensional map of the input data.

The primary use of the self-organizing map is to visualize topologies and hierarchical structures of higher-order dimensional input spaces. The self-organizing network has been used to create area-filled curves in two-dimensional space created by the Kohonen layer. The Kohonen layer can also be used for optimization problems by allowing the connection weights to settle out into a minimum energy pattern.

A key difference between this network and many other networks is that the self-organizing map learns without supervision. However, when the topology is combined

with other neural layers for prediction or categorization, the network first learns in an unsupervised manner and then switches to a supervised mode for the trained network to which it is attached.

An example self-organizing map network is shown in Figure 18. The self-organizing map has typically two layers. The input layer is fully connected to a two-dimensional Kohonen layer. The output layer shown here is used in a categorization problem and represents three classes to which the input vector can belong. This output layer typically learns using the delta rule and is similar in operation to the counter-propagation paradigm.

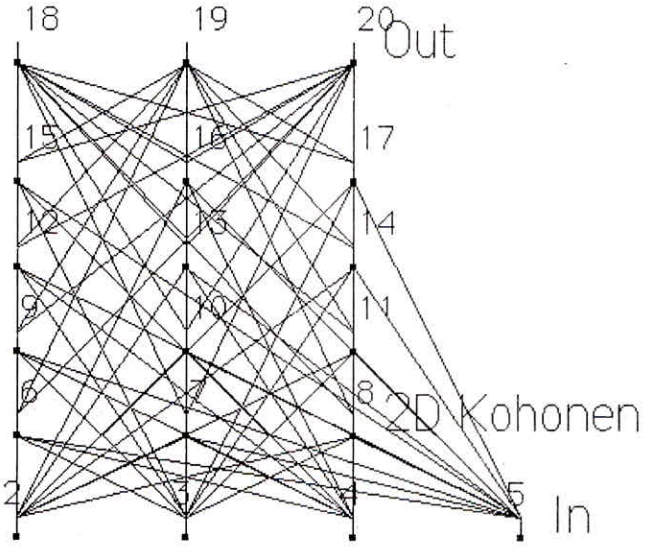


Figure 18: Self-organizing Map Network

The Kohonen layer processing elements each measure the Euclidean distance of its weights from the incoming input values. During recall, the Kohonen element with the minimum distance is the winner and outputs a one to the output layer, if any. This is a competitive win, so all other processing elements are forced to zero for that input vector. Thus the winning processing element is, in a measurable way, the closest to the input value and thus represents the input value in the Kohonen two-dimensional map. So the input data, which may have many dimensions, comes to be represented by a two-dimensional vector which preserves the order of the higher dimensional input data. This can be thought of as an order-perserving projection of the input space onto the two-dimensional Kohonen layer.

During training, the Kohonen processing element with the smallest distance adjusts its weight to be closer to the values of the input data. The neighbors of the winning element also adjust their weights to be closer to the same input data vector. The adjustment of neighboring processing elements is instrumental in preserving the order of the input space. Training is done with the competitive Kohonen learning law described in counter-propagation.

The problem of having one processing element take over for a region and representing too much input data exists in this paradigm. As with counter-propagation, this problem is solved by a conscience mechanism built into the learning function. The conscience rule depends on keeping a record of how often each Kohonen processing element wins and this information is then used during training to bias the distance measurement. This conscience mechanism helps the Kohonen layer achieve its strongest benefit. The processing elements naturally represent approximately equal information about the input data set. Where the input space has sparse data, the representation is compacted in the Kohonen space, or map. Where the input space has high density, the representative Kohonen elements spread out to allow finer discrimination. In this way the Kohonen layer is thought to mimic the knowledge representation of biological systems.

4.5 Networks for Data Filtering

The last major type of network is data filtering. An early network, the MADALINE, belongs in this category. The MADALINE removed the echoes from a phone line through a dynamic echo cancellation circuit. More recent work has enabled modems to work reliably at 4800 and 9600 baud through dynamic equalization techniques. Both of these applications utilize neural networks which were incorporated into special purpose chips.

4.5.1 Recirculation

Recirculation networks were introduced by Geoffrey Hinton and James McClelland as a biologically plausible alternative to back-propagation networks. In a back-propagation network, errors are passed backwards through the same connections that are used in the feedforward mechanism with an additional scaling by the derivative of the feedforward transfer function. This makes the back-propagation algorithm difficult to implement in electronic hardware.

In a recirculation network (Figure 19), data is processed in one direction only and learning is done using only local knowledge. In particular, the knowledge comes from the state of the processing element and the input value on the particular connection to be adapted. Recirculation networks use unsupervised learning so no desired output vector is required to be presented at the output layer. The network is auto-associative, where there are the same number of outputs as inputs.

This network has two layers between the input and output layers, called the visible and hidden layers. The purpose of the learning rule is to construct in the hidden layer an internal representation of the data presented at the visible layer. An important case of this is to compress the input data by using fewer processing elements in the hidden layer. In this case, the hidden representation can be considered a compressed version of the visible representation. The visible and hidden layers are fully connected to each other in both directions. Also, each element in both the hidden and visible layers are connected to a bias element. These connections have variable weights which learn in the same manner as the other variable weights in the network.

The learning process for this network is similar to the bi-directional associative memory technique. Here, the input data is presented to the visible layer and passed on to the hidden layer. The hidden layer passes the incoming data back to the visible, which in turn passes the results back to the hidden layer and beyond to the output layer. It is the second

pass through the hidden layer where learning occurs. In this manner the input data is recirculated through the network architecture.

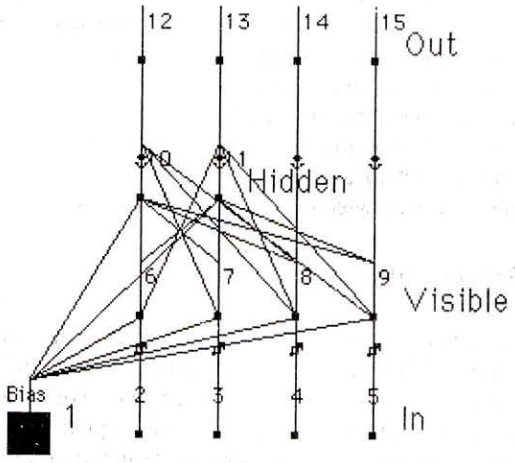


Figure 19: Recirculation Network

During training, the output of the hidden layer at the first pass is the encoded version of the input vector. The output of the visible layer on the next pass is the reconstruction of the original input vector from the encoded vector on the hidden layer. The aim of the learning is to reduce the error between the reconstructed vector and the input vector. This error is also reflected in the difference between the outputs of the hidden layer at the first and final passes since a good reconstruction will mean that the same values are passed to the hidden layer both times around. Learning seeks to reduce the reconstruction error at the hidden layer also.

In most applications of the network, an input data signal is smoothed by compressing then reconstructing the input vector on the output layer. The network acts as a low bandpass filter whose transition point is controlled by the number of hidden nodes.

5.0 How Artificial Neural Networks Are Being Used

Artificial neural networks are undergoing the change that occurs when a concept leaves the academic environment and is thrown into the harsher world of users who simply want to get a job done. Many of the networks now being designed are statistically quite accurate but they still leave a bad taste with users who expect computers to solve their problems absolutely. These networks might be 85% to 90% accurate. Unfortunately, few applications tolerate that level of error.

While researchers continue to work on improving the accuracy of their "creations," some explorers are finding uses for the current technology. In reviewing this state of the art, it is hard not to be overcome by the bright promises or tainted by the unachieved realities. Currently, neural networks are not the user interface which translates spoken works into instructions for a machine, but some day they will. Someday, VCRs, home security systems, CD players, and word processors will simply be activated by voice. Touch screen and voice editing will replace the word processors of today while bringing

spreadsheets and data bases to a level of usability pleasing to most everyone. But for now, neural networks are simply entering the marketplace in niches where their statistical accuracy is valuable as they await what will surely come.

Many of these niches indeed involve applications where answers are nebulous. Loan approval is one. Financial institutions make more money by having the lowest bad loan rate they can achieve. Systems that are "90% accurate" might be an improvement over the current selection process. Indeed, some banks have proven that the failure rate on loans approved by neural networks is lower than those approved by some of their best traditional methods. Also, some credit card companies are using neural networks in their application screening process.

This newest method of seeking the future by analyzing past experiences has generated its own unique problems. One of those problems is to provide a reason behind the computer-generated answer, say as to why a particular loan application was denied. As mentioned throughout this report, the inner workings of neural networks are "black boxes." Some people have even called the use of neural networks "voodoo engineering." To explain how a network learned and why it recommends a particular decision has been difficult. To facilitate this process of justification, several neural network tool makers have provided programs which explain which input through which node dominates the decision making process. From that information, experts in the application should be able to infer the reason that a particular piece of data is important.

Besides this filling of niches, neural network work is progressing in other more promising application areas. The next section of this report goes through some of these areas and briefly details the current work. This is done to help stimulate within the reader the various possibilities where neural networks might offer solutions, possibilities such as language processing, character recognition, image compression, pattern recognition among others.

5.1 Language Processing

Language processing encompasses a wide variety of applications. These applications include text-to-speech conversion, auditory input for machines, automatic language translation, secure voice keyed locks, automatic transcription, aids for the deaf, aids for the physically disabled which respond to voice commands, and natural language processing.

Many companies and universities are researching how a computer, via ANNs, could be programmed to respond to spoken commands. The potential economic rewards are a proverbial gold mine. If this capability could be shrunk to a chip, that chip could become part of almost any electronic device sold today. Literally hundreds of millions of these chips could be sold.

This magic-like capability needs to be able to understand the 50,000 most commonly spoken words. Currently, according to the academic journals, most of the hearing-capable neural networks are trained to only one talker. These one-talker, isolated-word recognizers can recognize a few hundred words. Within the context of speech, with pauses between each word, they can recognize up to 20,000 words.

Some researchers are touting even greater capabilities, but due to the potential reward the true progress, and methods involved, are being closely held. The most highly touted, and demonstrated, speech-parsing system comes from the Apple Corporation. This network, according to an April 1992 Wall Street Journal article, can recognize most any person's speech through a limited vocabulary.

This works continues in Corporate America (particularly venture capital land), in the universities, and in Japan.

5.2 Character Recognition

Character recognition is another area in which neural networks are providing solutions. Some of these solutions are beyond simply academic curiosities. HNC Inc., according to a HNC spokesman, markets a neural network based product that can recognize hand printed characters through a scanner. This product can take cards, like a credit card application form, and put those recognized characters into a data base. This product has been out for two and a half years. It is 98% to 99% accurate for numbers, a little less for alphabetical characters. Currently, the system is built to highlight characters below a certain percent probability of being right so that a user can manually fill in what the computer could not. This product is in use by banks, financial institutions, and credit card companies.

Odin Corp., according to a press release in the November 4, 1991 Electronic Engineering Times, has also proved capable of recognizing characters, including cursive. This capability utilizes Odin's propriatory Quantum Neural Network software package called, QNspec. It has proven uncannily successful in analyzing reasonably good handwriting. It actually benefits from the cursive stroking.

The largest amount of research in the field of character recognition is aimed at scanning oriental characters into a computer. Currently, these characters requires four or five keystrokes each. This complicated process elongates the task of keying a page of text into hours of drudgery. Several vendors are saying they are close to commercial products that can scan pages.

5.3 Image (data) Compression

A number of studies have been done proving that neural networks can do real-time compression and decompression of data. These networks are auto associative in that they can reduce eight bits of data to three and then reverse that process upon restructuring to eight bits again. However, they are not lossless. Because of this losing of bits they do not favorably compete with more traditional methods.

5.4 Pattern Recognition

Recently, a number of pattern recognition applications have been written about in the general press. The Wall Street Journal has featured a system that can detect bombs in luggage at airports by identifying, from small variances, patterns from within specialized sensor's outputs. Another article reported on how a physician had trained a back-propagation neural network on data collected in emergency rooms from people who felt that they were experiencing a heart attack to provide a probability of a real heart attack

versus a false alarm. His system is touted as being a very good discriminator in an arena where priority decisions have to be made all the time.

Another application involves the grading of rare coins. Digitized images from an electronic camera are fed into a neural network. These images include several angles of the front and back. These images are then compared against known patterns which represent the various grades for a coin. This system has enabled a quick evaluation for about $15 as opposed to the standard three-person evaluation which costs $200. The results have shown that the neural network recommendations are as accurate as the people-intensive grading method.

Yet, by far the biggest use of neural networks as a recognizer of patterns is within the field known as quality control. A number of automated quality applications are now in use. These applications are designed to find that one in a hundred or one in a thousand part that is defective. Human inspectors become fatigued or distracted. Systems now evaluate solder joints, welds, cuttings, and glue applications. One car manufacturer is now even prototyping a system which evaluates the color of paints. This system digitizes pictures of new batches of paint to determine if they are the right shades.

Another major area where neural networks are being built into pattern recognition systems is as processors for sensors. Sensors can provide so much data that the few meaningful pieces of information can become lost. People can lose interest as they stare at screens looking for "the needle in the haystack." Many of these sensor-processing applications exist within the defense industry. These neural network systems have been shown successful at recognizing targets. These sensor processors take data from cameras, sonar systems, seismic recorders, and infrared sensors. That data is then used to identify probable phenomenon.

6.0 Recurrent Neural Network

Recurrent neural networks (RNN) are fundamentally different from feedforward architectures in the sense that they not only operate on an input space but also on an internal *state* space – a trace of what already has been processed by the network. The human brain is a recurrent neural network (RNN) or feedback neural network: a network of neurons with feedback connections. From training examples they can learn to map input sequences to output sequences. In principle they can implement almost arbitrary sequential behavior, which makes them promising for adaptive robotics, speech recognition, music composition, attentive vision, and many other applications. RNNs or feedback networks are biologically more plausible and computationally more powerful than other adaptive models such as Hidden Markov Models (no continuous internal states), feedforward networks and Support Vector Machines (no internal states at all).

How RNN is different from feedforward neural network

The network on the figure 20 is a simple feed forward network. A recurrent neural network (RNN) is a modification to this architecture to allow for temporal classification, as shown in Figure 21. In this case, a ``context" layer is added to the structure, which retains information between observations. At each timestep, new inputs are fed into the RNN. The previous contents of the hidden layer are passed into the context layer. These then feed back into the hidden layer in the next time step.

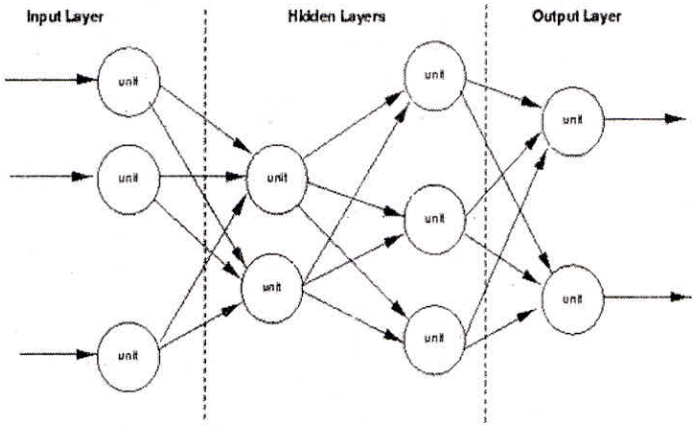


Figure 20: Feed Forward Network

In an algorithm similar to the backpropagation algorithm, called back propagation through time (BPTT), the weights of the hidden layers and context layers are set. To do classification, post processing of the outputs from the RNN is performed; so, for example, when a threshold on the output from one of the nodes is observed, we register that a particular class has been observed.

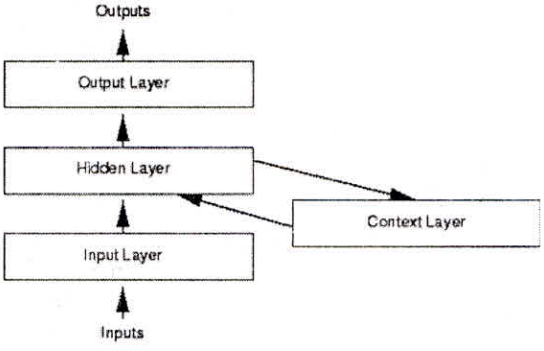


Figure 21: A simple Recurrent Neural Network

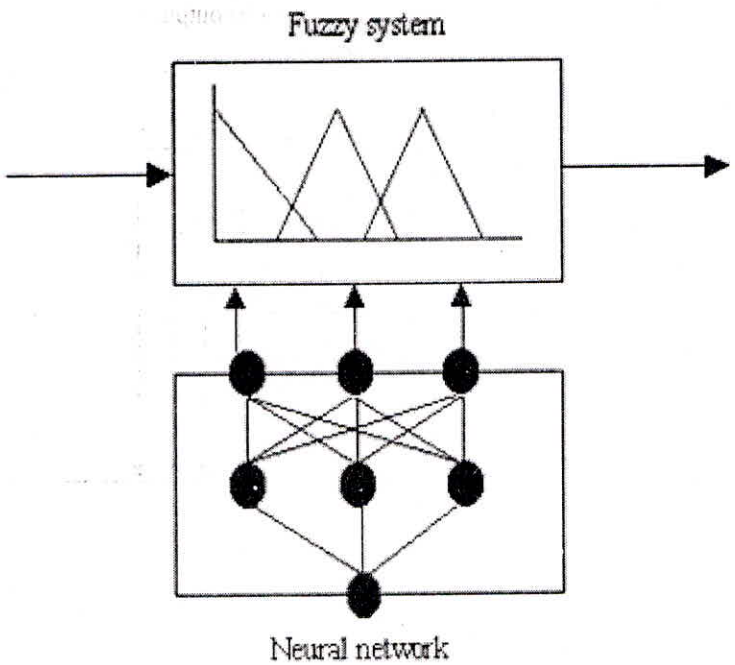
Recurrent neural networks suffer from many of the same problems as HMMs (Hidden Markov Model), namely:

- There are many parameters. These include parameters such as the number of units in the hidden layer, the appropriate structure, the correct parameter adjustment algorithm (there are many alternatives to backpropagation), the learning rate, the encoding and more. Well-formed techniques for deciding appropriate values for these parameters are in the development stage.
- Their efficacy for learning long connected sequences is doubtful. They do seem capable of learning short sequences (tens of frames), such as learning individual phonemes, rather than whole words.

7.0 Neural Networks and Fuzzy Systems

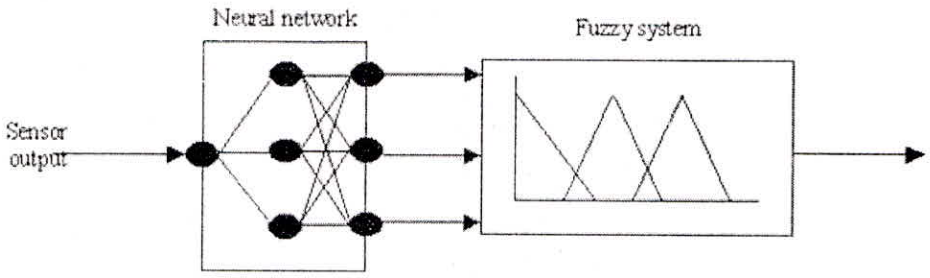
The basic idea of combining fuzzy systems and neural networks (Fig. 22) is to design an architecture that uses a fuzzy system to represent knowledge in an interpretable manner and the learning ability of a neural network to optimize its parameters. The drawbacks of both of the individual approaches - the black box behavior of neural networks, and the problems of finding suitable membership values for fuzzy systems - could thus be avoided. A combination can constitute an interpretable model that is capable of learning and can use problem-specific prior knowledge. Therefore, neuro-fuzzy methods are especially suited for applications, where user interaction in model design or interpretation is desired. Discussions and an overview of current approaches can be found, e.g. Lin, and Lee, 1996; Nauck et al, 1997; Close et al., 2001.

Neuro-fuzzy Systems, the most widely researched of all the hybrid systems at the present time, are a combination of neural networks and fuzzy-logic. Fuzzy logic provides a structure within which the learning ability of neural networks is employed. In this field there are a number of possible uses. Firstly, neural networks can be used to generate the membership functions for a fuzzy system and to tune them. The neuro fuzzy systems connected in series and parallel systems are shown in fig 23 and 24



Figures 22: Combining Fuzzy with Neural Network

The series system would be used if the sensor output is not suitable for direct connection to the input of the fuzzy system. Post-processing systems also exist, in which the output of a fuzzy system is not suitable for direct connection to external devices, and therefore a neural network provides an interface which performs a mapping which could not easily be carried out using analytical techniques.



Figures 23: Series Neuro-fuzzy systems

The neural network fine tunes the output of the fuzzy system according to what it has learned about users' personal preferences from the fine adjustments they have previously made. The parallel network system used for fine tuning the output is shown in fig. 24

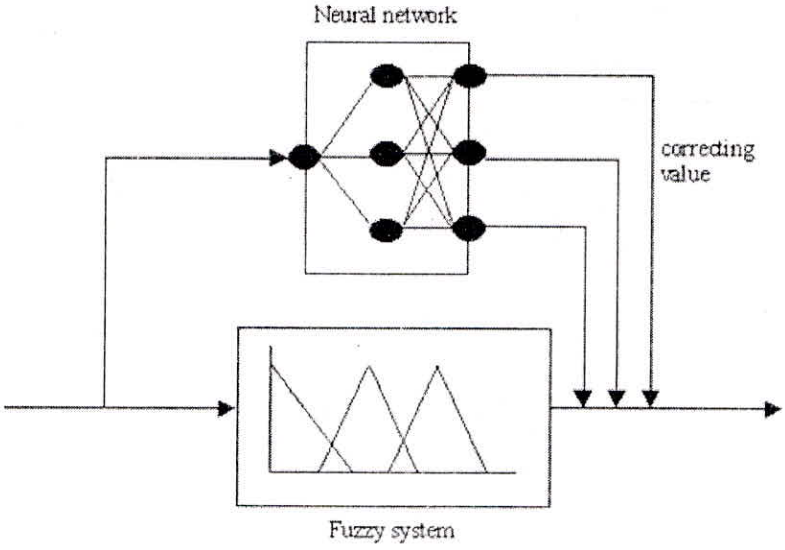


Figure 24: Parallel neural network-fuzzy system for fine-tuning an output

Adaptive Network-based Fuzzy Inference System, is the commonly used platform for neuro-fuzzy model development in which neural networks are used to implement a fuzzy inference system. A fuzzy inference system consists of three components. Firstly, a rule base contains a selection of fuzzy rules. Secondly, a database defines the membership functions used in the rules and, finally, a reasoning mechanism carries out the inference procedure on the rules and given facts. The concept of fuzzy reasoning is straightforward. The truth of a proposition A infers the truth of a proposition B by the implication

$A \rightarrow B$

For example, if A represents "the banana is yellow" and B represents "the banana is ripe", then if "the banana is yellow" it is inferred that "the banana is ripe". Fuzzy reasoning then allows the inference that if "the banana is more or less yellow" then "the banana is more or less ripe".

Neuro-fuzzy systems have become popular in several fields. Control is a notable example - particularly space and aviation applications, where auto-pilots aim to mimic human ability to make reasoned judgements. ***Lee et al.***, describe a system for face recognition in which various features are extracted from the face and fuzzified to make them less sensitive to variation of features of the same person. The fuzzified features are then applied to a neural network for the recognition process.

8.0 ANN APPLICATIONS IN WATER RESOURCES

General

The development of ANN began about 60 yrs ago (McCulloch and Pitts, 1943), inspired the way neurons in brain work and analyze a perception. However due to various reasons it never became much popular among the engineering and scientific communities, until in 1986 when Rumelhart (1986) discovered the "back-propagation algorithm". However the work done by Hopfield in 1982 in the field of iterative auto-associable neural networks was also very influential in ANN computing techniques. From that year (i.e 1986) onwards there was a tremendous increase in ANN applications in various scientific and engineering fields such as: biology, neurophysiology, physics, biomedical engineering, electrical engineering, computer science engineering, acoustics, robotics, cybernetics, image processing etc. Regular applications of ANN to hydrology started in early 90's only.

One of the earliest problems to be dealt in hydrology using ANN was the rainfall-runoff modeling process. Some of the reasons for a great interest in ANN among hydrologists is that: (1) ANNs can be used to solve large complex problems very easily even when minimum data is available, which is the case most of the time in many hydrologic problems. (2) ANN is a type of heuristic procedure i.e. information regarding input and output is enough and the understanding of the process itself is not very water like: hydrology, hydraulics, groundwater engineering, water quality modeling etc.

Before going into the applications of ANN in WRE a brief review about various problem solving approaches in WRE, is discussed:

Empirical Methods/Models: These are based on some thumb rules and are location specific i.e they are not generic. They can be best described as black box models which give an output for a given input without bothering about the process of conversion of input to output. Eg: Lumped models.

Geomorphology based models/methods: These are the models which are an improvement over empirical models, in that, they try to develop mathematical equations connecting various geographical parameters while making appropriate assumptions. Eg: Distributed models.

Physical Models: These are the most accurate of the three models as they try to capture the real physics of the process. They involve (numerical) solution for a system of partial differential equations. However these models suffer from the problems such as: identification, estimability and uniqueness of parameter estimation.

Based on the above description of the models, ANN would be considered as an empirical method. Important aspects of ANN modeling are:

1. Selection of input and output variables
2. Collecting and processing data
3. Designing an ANN
4. Training and cross training
5. Model validation

Strength of ANN:

1. They recognize the relationship between input and output variables without explicit physical considerations.
2. They work well even when the training sets contain noise and measurement errors
3. They are able to adapt to the solutions over the time to compensate for changing circumstances
4. Once trained are easy to use

Limitations of ANN:

1. Requirement of data of good quality and quantity
2. Lack of physical concepts and relationships
3. The fact that there is no standardized way of selecting network architecture

Now we will discuss ANN applications in various fields of WRE. The topic can be broadly classified into various sub-categories like:

1. Estimation of precipitation and evaporation
2. Applications in ground water
3. Water quality modeling
4. Rainfall-Runoff modeling
5. Stream flows modeling

8.1 ANN for Estimating Precipitation and Evaporation

Precipitation and evaporation in hydrology are highly non-linear and random processes. They exhibit a lot of spatial and temporal variations. ***French et al.,*** (1992) used a three-layer feedforward ANN with back-propagation to forecast rainfall intensity fields at a lead time of 1 hour with the current field as input. The authors compared ANN generated results with those from persistence and forecasting models. Their results suggested that ANNs performed slightly better than these models during the training stage after a suitable architecture had been identified. But their performance over the testing data set was not satisfactory. They concluded that the ability of an ANN to generalize the underlying rule was strongly dependent on selecting a large enough hidden buyer.

Tohma and Igata (1994) employed a three-layer ANN to estimate rainfall fields based on visible and infrared remote sensing cloud images in the coastal region of south-western Hokkaido and in a heavy rainfall area of Hokkaido, Japan. They reported that ANNs

could map the relationship between remotely sensed images of clouds and rainfall intensities and provide short-term forecasts of rainfall.

Navone and Ceccatto (1994) have used an ANN model to predict summer monsoon rainfall (SMR) over India. The inserted the stochastic and dynamic models into ANN framework. The resulting hybrid network was shown to perform 40%more accurately than the best linear statistical method using the same data.

Hsu et al., (1996, 1997) developed a modified counter-propagation ANN for transforming satellite infrared images to rainfall rates over a specified area. The results indicated that ANNs provided a good estimation of rainfall and yielded some insights into the functional relationships between the input variables and the rainfall rate.

Zhang et al., (1997) proposed that ANNs need to be employed in groups when the transformation from the input to the output space is complex. By this, the authors were successful in making half-hourly rainfall estimates. ***Kuligowski and Barros*** (1998) present an ANN approach for short-term precipitation prediction. Their model uses a feed forward architecture. Compared with a persistence model, the proposed ANN model showed significant improvement for short-term precipitation prediction.

Kumar et al., (2002) investigated the utility of artificial neural networks (ANNs) for estimation of daily grass reference crop evapo-transpiration (ETo) and compared the performance of ANNs with the conventional method (Penman–Monteith) used to estimate ETo. The results revealed that the single hidden layer ANNs were sufficient to account for the nonlinear relationship between climatic variables and corresponding ETo. Improvement in performance ceased for higher learning cycles and PEs in the hidden layer. The results also suggested that given the lysimeter measured ETo as a target, ANN predicted ETo could be better than the PM method. However the generalization of ANN in this area is to be studied further.

Sudheer et al., (2003) examined the potential of artificial neural networks (ANN) in estimating the actual crop evapotranspiration (ET) from limited climatic data. The results from the study indicated that crop evapo-transpiration can be successfully estimated using limited data through the ANN approach. The authors claimed that a unique advantage of the ANN approach in estimating ET was that it eliminates the need for identifying a reference crop and it requires only limited climatic data. However, the study only used data from a single crop for a limited period and further studies using more data would be required to strength the conclusions.

Slavisa et al., (2003) developed a sequentially adaptive radial basis function (RBF) network for the forecasting of reference evapotranspiration (ET*o*). The sequential adaptation of parameters and structure was achieved using an extended Kalman filter. The results suggested that adaptive RBF networks are a promising approach to forecasting reference evapotranspiration. Even better results may be expected with further improvement of the RBF networks.

Keskin and Terzi (2006) proposed ANNs as an alternative approach of evaporation estimation for Lake Eğirdir. The study had three objectives: (1) to develop ANN models to estimate daily pan evaporation from measured meteorological data;(2) to compare the ANN models to the Penman model; and (3) to evaluate the potential of ANN models. They analyzed that ANN models have higher R^2 and lower MSE values for both the

training and testing data sets than the Penman method. Also the performance of the ANN model with air and water temperature and solar radiation inputs suggested that the evaporation could be estimated from easily available data using the ANN approach.

8.2 ANN Applications in Ground Water Engineering

It is difficult to separate ground water and water quality as different sections. Many articles have addressed both these topics to some extent (Rogers and Dowla 1994; Roger et al. 1995). ***Aziz and Wong*** (1992) illustrated the use of ANNs for determining aquifer parameter values from normalized drawdown data obtained from pumping tests. This study grew on the pattern recognition of an ANN based on aquifer test data. Using draw-downs as inputs transmissivity T, storativity S and r/B ratio were estimated. Both confined and leaky confined aquifers were considered. The values of aquifer parameters compared well with results using traditional methods.

Rizzo and Daughtery (1994) introduced the idea of neural krigging for characterization of aquifer properties. The authors, based on their study for estimating K, pointed out that neural krigging produced unbiased estimates of the K values at unmeasured locations. They concluded that ANNs could be a useful tool in geo-hydrology when applied to specific problems of aquifer characterization.ANNs when combined with GA, were showed to accelerate the search process for estimating a few optimal pumping strategies among a vast number of possible pumping patterns (Rogers, 1992; Rogers & Dowla, 1994; Johnson & Rogers, 1995; Rogers et al. 1995). However it can provide meaningful solutions only over the problem dimensions defined by initial model runs used for training. Once the scope of problem changes, like increase in management time frame or the addition of new perspective well locations, training must be repeated with a new information.

Yang et al., (1997) utilized ANNs to predict water table elevations in subsurface drained farmlands. Daily rainfall, potential evaporation and previous water table locations were used selected as input to ANN. The output was current location of water table. They found that ANN could predict the water table elevation levels satisfactorily after training using observed values.

Coulibaly et al., (2001) modeled the fluctuations of ground water levels using RNN. Three types of functionally different artificial neural network (ANN) models are calibrated using a relatively short length of groundwater level records and related hydrometeorlogical data to simulate water table fluctuations in the Gondo aquifer, Burkina Faso. Input delay neural network (IDNN) with static memory structure and globally recurrent neural network (RNN) with inherent dynamical memory are proposed for monthly water table fluctuations modeling. The simulation performance of the IDNN and the RNN models is compared with results obtained from two variants of radial basis function (RBF) networks, namely, a generalized RBF model (GRBF) and a probabilistic neural network (PNN). Overall, simulation results suggest that the RNN is the most efficient of the ANN models tested for a calibration period as short as 7 years. The results of the IDNN and the PNN are almost equivalent despite their basically different learning procedures. The GRBF performs very poorly as compared to the other models. Furthermore, the study shows that RNN may offer a robust framework for improving water supply planning in semiarid areas where aquifer information is not available. This study has significant implications for groundwater management in areas with inadequate groundwater monitoring network.

da Silva et al., (2000) used Artificial Neural Network (ANN) to estimate the depth of the dynamical water level of groundwater wells in relation to water flow, operation time and rest time. The conventional estimation of aquifer dynamical behavior involves understanding of the relationship between the structural parameters associated with it. These structural parameters depend on various factors such as soil properties and hydrologic and geologic aspect. The authors reported that the ability of artificial neural networks on complex nonlinear functions realization makes it attractive to identify and estimate dynamical behavior of underground aquifers.In the study, feedforward ANNs were used to map the relationships between the variables associated with the process of identification of aquifer dynamical behavior. The learning algorithm used to compute the weights of the network was the backpropagation and it corresponds to the parameter estimation stage.

The first network (ANN-I) has ten neurons in the hidden layer and it is responsible for the computation of the aquifer operation level. The training data for ANN-1 were directly obtained from experimental measurements. The second network (ANN-2) is responsible for the computation of the aquifer dynamical level and it is composed by two hidden layers with both having ten neurons. For this network, the training data were also obtained from experimental measurements. As observed in Figure 1, the ANN-I output is provided as an input parameter to the ANN-2. Therefore, the computation of the aquifer dynamical level takes into account the aquifer operation level, the exploration flow and operation time. After training process of the neural networks, they were used for estimation of the aquifer dynamical level. From the study the authors concluded that proposed approach can be efficiently used in the similar kinds of problem with great accuracy level.

Coppola Jr. et al., (2003) proposed an ANN approach for predicting transient water levels in a multilayer groundwater system under variable state, pumping and climatic conditions. A study was carried over the less than 2.5 months validation period using significantly fewer inputs and it was observed that ANN achieved a much higher degree of accuracy than the numerical flow model with far less development efforts. For this application a relatively short historical record of groundwater and climate information was sufficient for achieving accurate water level predictions with ANN. ANN can be easily reinitialized with existing water levels in real time since re-initialization is a problematic part in numerical models.

Raj et al., (2004) showed that a trained ANN can be used to solve the complex problem of unknown groundwater pollution source identification. However this requires a number of observation wells & the location of wells are also vital. One of the limitations of their study was in terms of large identification errors. More rigorous evaluation is necessary before the applicability of the ANNs approach is established.

Dixon (2005) applied neuro-fuzzy techniques in predicting ground-water vulnerability. Neuro-fuzzy modeling is an approach where the fusion of neural networks and fuzzy logic find their strengths. These two techniques complement each other. The neuro-fuzzy approaches employ heuristic learning strategies derived from the domain of neural networks theory to support the development of a fuzzy system. It is possible to completely map neural networks knowledge to fuzzy logic. A marriage between neural networks and fuzzy logic techniques should help overcome the short comings of both techniques. Delineation of vulnerable areas and selective applications of contaminants can minimize contamination of groundwater. However, assessment of groundwater vulnerability or delineation of critical monitoring zones (areas) is not easy since

contamination depends upon numerous, complex interacting parameters and uncertainty is inherent in all methods of assessing groundwater vulnerability. Hence, reliably and cost effectively modeling groundwater vulnerability from non-point sources (NPS) at a regional scale remains a major challenge. This study addresses this challenge by proposing a new methodology that predicts groundwater vulnerability using neuro-fuzzy techniques with a geographic information system (GIS) after extensive sensitivity analysis.

In recent years, several hydrological studies have used neural networks, fuzzy logic, and neuro-fuzzy techniques to make predictions. However, very few of these research studies undertook extensive sensitivity analysis. Sensitivity analysis could provide insights into creating the ideal combination of model parameters for the neuro-fuzzy systems that could then produce reliable and meaningful ground water vulnerabiiity maps with minimum pre-processing and computation time. Therefore, the overall objective of this work was to examine the sensitivity of neuro-fuzzy models used to predict groundwater vulnerability in a spatial context by integrating GIS and neuro-fuzzy techniques. The specific objectives were to assess the sensitivity of neuro-fuzzy models by varying (i) shape of the fuzzy sets, (ii) number of fuzzy sets, and (iii) learning and validation parameters (including rule weights).

Daliakopoulos et al., (2005) used artificial neural networks for groundwater level forecasting. Neural networks have proven to be an extremely useful method of empirical forecasting of hydrological variables. In this paper an attempt was made to identify the most stable and efficient neural network configuration for predicting groundwater level in the Messara Valley. The groundwater in the area has been steadily decreasing since the late 1980s due to overexploitation due to intensive irrigation. A total of seven different ANN configurations were tested in terms of optimum results for a prediction horizon of 18 months.

From the results of the study it can also be inferred that the Levenberg–Marquardt algorithm is more appropriate for this problem since the RNN also performs well when trained with this method. Moreover, combining two or more methods of prediction should also be considered as in our case the FNN-LM method tended to underestimate events when the rest of the methods overestimated them. In general, the results of the case study are satisfactory and demonstrate that neural networks can be a useful prediction tool in the area of groundwater hydrology. Most importantly, this paper presents indications that neural networks can also be applied in cases where the datasets manifest trends and shifts and the desired output has lies outside of the range of previously introduced input, as shown by the results.

8.3 ANN in Water Quality Modeling

Water quality is influenced by many factors such as flow rate, contaminant load, medium of transport, water levels, initial conditions and other site-specific parameters. The estimation of such variables is often a complex and nonlinear problem, making it suitable for ANN application. ***Maier and Dandy*** (1996) illustrated the utility of ANNs for estimating salinity at the Murray Bridge on the River Murray in South Australia. In this study, the inputs to the ANN model were daily salinity values, and water levels and flows at upstream stations and at antecedent times whereas network output was the 14-day-advance forecast of river salinity. It was observed that the average percentage errors of the independent 14 day forecasts for four different years of data varied from 5.3 to 7.0%.

The authors also concluded that the impact of using different learning rates and different network geometries was relatively minor.

Rogers (1992) and ***Rogers and Dowla*** (1994) employed an ANN, which was trained by a solute transport model, to perform optimization studies in ground-water remediation. A multilayer feedforward ANN was trained using the back-propagation training algorithm. The methodology was applied to a Superfund site by ***Rogers et al.,*** (1993) and ***Johnson and Rogers*** (1995). They concluded that ANNs, combined with a genetic algorithm, result in robust and flexible tools that can be used for planning effective strategies in ground-water remediation. They concluded that ANNs, combined with a genetic algorithm, result in robust and flexible tools that can be used for planning effective strategies in ground-water remediation.

Morshed and Kaluarachchi (1998) used an ANN to estimate the saturated hydraulic conductivity and the grain size distribution parameter for application in the problem of free product recovery. They also concluded that the search process in the parameter space could be accelerated when the ANN was guided by a genetic algorithm.

Basheer and Najjar (1995) used a three-layer artificial neural network to predict the breakthrough time in a fixed-bed adsorption system. Using a systematic analysis, the authors identified three inputs as being the most influential in determining the breakthrough time. These were influent concentration, specific weight of the adsorbent, and the particle diameter of the porous bed material. The authors found ANN predictions to be reliable as long as the inputs were within the range of the data sets.

Ray and Klindworth (1996) lay a blueprint for addressing the problem of agriculture chemical assessment in the rural private wells in Illinois using neural networks. They envisioned that important inputs would be depth to the aquifer material, well depth, land topography in the vicinity of the well, distance of potential contaminant sources from the well, and timing of precipitation with respect to pesticide application. They also discussed how data would be collected for such an application and commented about the utility of ANNs in such applications.

Sandhu and Finch (1996) used ANNs to relate flow conditions and gate positions in the Sacramento San Joaquin Delta to salinity levels in the interior and along the boundary of the delta. ANNs were further used to estimate flow in the Sacramento River to meet salinity standards. They found simulation models too slow and the commonly used statistical models to be inadequate, and they concluded that neural networks would be suitable for this application.

Hutton et al., (1996) used neural networks to enhance the capability of an existing model for predicting trihalomethane (THM) formation and specification by including variable reaction conditions. Sensitivity analyses showed that ANNs were predicting the right trends of TPH chemistry. The authors concluded that ANN models predict THM formation species and total concentrations in delta waters in an adequate manner.

Nagy et al., (2002) developed an artificial neural model to estimate the natural sediment discharge in rivers in terms of sediment concentration. Their study details the application of ANNs to the problem of sediment discharge estimation. The authors showed that the neural networks model can be successfully applied for the sediment transport when other approaches cannot succeed due to the uncertainty and the stochastic nature of the sediment movement. They listed in their paper various points regarding the advantages in

use of ANN for sediment discharge estimation. Some of them are: (1) The networks, can overcome the stochastic nature of the sediment movement more than any other distinct formula. (2) The ANN can accept any number of effective variables as input parameters. (3) The presented ANN model is constructed by using only field river data, and it has no boundary conditions in application. (4) Site engineers can calculate sediment discharge using the ANN without prior knowledge of the sediment transport theories. However one of the limitations of the study was that the model cannot estimate accurately the sediment concentration for data out of the range of the learning pattern data.

Lindsay et al., (2002) showed use of the measured complex permittivity of electrolyte solutions for predicting ionic species and concentration is investigated using ANNs. The performance of the ANNs developed in their paper demonstrated that there is potential for using complex permittivity data to detect the invasion of metal contaminants into groundwater, in terms of contaminant species and concentration. As per the authors, the nonlinear modeling capabilities of neural networks can play an important role in this research. They concluded that: (1) The artificial neural networks developed were able to accurately classify cationic and anionic species (2) The MLP1 (multilayer perceptrons) correctly classified 83.8% of the total number of cationic contaminants in the training set and 83.3% in the test cases. The MLP1 also demonstrated superior ability to distinguish between samples of pure water and cation containing solutions. The MLP3 correctly classified 97.5% of the total number of anionic contaminants in the training set and 97.9% of the test cases. The MLP3 was able to correctly classify all but one pure-water sample (3) Both MLP2 and RFB1 (radial basis functions) were able to explain much of the variability in cationic and anionic concentrations, respectively; however, MLP2 was less accurate at higher concentrations.

Markus, M. et al., (2003) developed three ANN models to predict weekly nitrate-N concentrations in the Sangamon River near Decatur, Illinois, based on past weekly precipitation, air temperature, discharge, and past nitrate-N concentrations. Preliminary analysis indicated that during training and testing stages, the ANN models were more accurate than the corresponding linear regression models. More complex ANN architecture could potentially provide more insight into the basic cause–effect relationships and possibly produce more accurate weekly predictions. Among the three ANN models used for weekly nitrate concentration forecasting N1 (one input i.e nitrate concentration), NQPT1 (four inputs i.e nitrate N, discharge Q, precipitation P and temperature T), and NQPT2 (i.e , N(t), Q(t), Q(t-1), P(t), P(t-1), T(t) and T(t-1) to predict N(t+1)) model NQPT1 was the most accurate.

Suen and Eheart (2003) applied ANN for estimating nitrate concentration in a Midwestern river, i.e., the Upper Sangamon River in Illinois. Back-propagation neural networks (BPNNs) and radial basis function neural networks (RBFNNs) were compared as to their effectiveness in water quality modeling. It was concluded that, all ANNs trained by the odd-even method outperformed the mechanistic SWAT model and traditional regression analysis. RBFNN, which used a fuzzy min-max network to perform the clustering and multivariate regression to construct the neural network, was much faster than the BPNN approach, which used the gradient descent method in its training procedure. However, the authors noted that since ANN is like any other regression model, it is therefore incapable of, or poorly capable of, extension to cases other than those for which it was trained. Further work is needed to assess whether the findings reported by the authors are robust for impounded and other un-impounded rivers in the Midwest and elsewhere. The specter of climate change also has implications for the frequency of high-nitrate events, and is a good candidate for future research.

Chaves et al., (2004) adopted artificial intelligence techniques for operation of storage reservoir for water quality by using optimization. Modeling water quality presents a great degree of complexity, for example, due to the great number of parameters to be defined, non-linearity of the processes involved and few available data. Time dependence for reservoir water quality model is not considered in the developed model due to lack of appropriate data and regarding the purpose of analysis. On the other hand, not considering time dependence makes the combination of the ANN model for water quality and the SDP model more straightforward, as the ANN model can be used with the backward calculation recommended for SDP.

Five quality parameters are simulated with the ANN model: DO, BOD, TP, TN and chlorophyll (CHA). As these parameters have some interdependence among themselves and the various input variables, it is assumed that a single ANN model would best reflect these interrelations. The number of hidden units is defined through trial-and-error. Note that with limited data available for training and validation, it is recommended to avoid using a large number of hidden nodes, avoiding the problem of over-fitting. The storage reservoir system is successfully optimized accounting for the uncertainties related to input loads. Moreover, stochastic characteristics of inflow are properly handled through the use of Markov chain process.

Mishra et al., (2004) showed the use of qualitative and quantitative information in neural networks for assessing agricultural chemical contamination of domestic wells. It was observed that exact estimation of concentrations was unrealistic with NN models since the authors had sparse knowledge regarding the degree of involvement of parameters with the level of contamination.

Azmathullah et al., (2005) developed a neural network model for estimation of scour downstream of a ski-jump bucket. The network predictions were said to be more satisfactory than those given by traditional regression equations because of low errors and high correlation coefficients. However the feed forward back propagation used by them, took a long time to train the neural networks. The input of bucket radius, lip angle, sediment size, and tail water depth were found to be necessary in addition to that of unit discharge and height of fall (as practiced in traditional formulas), if accurate predictions are to be desired. Further research based on the type of rock bed, classified as per rock quality designation, and rock mass rating by using artificial neural network (ANN) and adaptive network based inference system (ANFIS) is underway.

Tayfur and Singh (2005) predicted longitudinal dispersion coefficient in natural streams using ANNs using flow variables and channel geometric characteristics. The satisfactory predictions of the measured data from streams having different geometric and flow characteristics revealed that the developed ANN model was superior to the existing theoretical and empirical equations. The ANN model, developed in their study, made no assumption with regard to stream geometry or flow dynamics in the stream. From sensitivity analysis the authors concluded the following points:

1. If the data on shear velocity, flow velocity, depth, and channel width are available, then the ANN model successfully predicts the wide ranging values of the dispersion coefficient of natural streams of different geometries. However, if one has only the discharge data then one can use that data in the ANN model to satisfactorily predict the more frequently encountered low values of the dispersion coefficient.

2. If geometric characteristics of the channel shape parameter and sinuosity are used along with the flow velocity in the input vector, then the ANN model satisfactorily predicts the dispersion coefficient.
3. If the relative shear velocity is used as the only input variable, then the ANN model can yield satisfactory predictions of the dispersion coefficient in more frequently encountered narrower channels.
4. The geometric characteristics, when used along with the relative shear velocity, can significantly improve the performance of the ANN model in predicting the longitudinal dispersion coefficient in natural streams.

Schmid and Koskiaho (2006) reported a study on application of artificial neural networks (ANNs) to the modeling of dissolved oxygen in the Finnish wetland pond of Hovi. It was concluded that the multilayer perceptron (MLP) class of models showed adequate predictive abilities in a very complex ambient with processes evolving at a wide range of spatiotemporal scales. It was demonstrated that the class of ANNs employed (i.e MLPs) were able to learn and generalize the mechanism of convective oxygen transport in the wetland pond under study.

8.4 ANN in Rainfall-Runoff Estimation

Determining the relationship between rainfall and runoff for a watershed is one of the most important problems faced by hydrologists and engineers. The relationship between rainfall and runoff is highly nonlinear and complex. In addition to rainfall, runoff is dependent on numerous factors such as initial soil moisture, land use, watershed geomorphology, evaporation, infiltration, distribution, duration of the rainfall, and so on. Research activities in this aspect havebeen quite revealing, and they can be broadly classified into two categories. The first category of studies are those where ANNs were trained and tested using existing models (e.g., Smith and Eli 1995; Shamseldin 1997). These studies may be viewed as providing a ''proof of concept'' analysis for ANNs. They have laid the foundations for future ANN use by demonstrating they are indeed capable of replicating model behavior, provided sufficient data is available for training.

Most ANN-based studies fall into the second category, those that have used observed rainfall-runoff data. Frequently, supplementary inputs such as temperature, snowmelt equivalent, and historical stream flows have been included. In such instances, comparisons with other empirical or conceptual models have also been provided. These studies provide a more comprehensive evaluation of ANN performance and are capable of establishing ANNs as viable tools for modeling rainfall-runoff.

A number of researchers have investigated the potential of neural networks in modeling watershed runoff based on rainfall inputs. In a preliminary study, ***Halff et al.***, (1993) designed a three-layer feedforward ANN using the observed rainfall hyetographs as inputs and hydrographs recorded by the U.S. Geological Survey (USGS) at Bellvue,Washington, as outputs. A sequence of 25 normalized 5 min rainfalls were applied as inputs to predict the runoff. This study opened up several possibilities for rainfall-runoff applications using neural networks.

Hjelmfelt and Wang (1993) developed a neural network based on the unit hydrograph theory. Using linear superposition, a composite runoff hydrograph for a watershed was developed by appropriate summation of unit hydrograph ordinates and runoff excesses. To implement this in a neural network framework, the number of units in the input and hidden layer was kept the same. The inputs to the ANN were sequences of rainfall.

Instead of the threshold function, a ramp transfer function corresponding to the rainfall Φ-index was used for the hidden layer. The output layer calculated a weighted sum of the rainfall excesses. The resulting network was shown to reproduce the unit hydrograph better than the one obtained through the standard gamma function representation. In a later study, ***Hjelmfelt and Wang*** (1996) compared this method with a regular three layered artificial network with back-propagation. The authors concluded that a regular network could not reproduce the unit hydrograph very well and was more susceptible to noise than a network whose architecture was more suited for unit hydrograph computations.

In an application using two neural networks, ***Zhu et al.,*** (1994) predicted upper and lower bounds on the flood hydrograph in Butter Creek, New York. Off-line predictions were made when present flood data were not available and estimates had to be based on rainfall data alone. On-line predictions were based on both rainfall and previous flood data. Data for ANN testing and validation were generated from a nonlinear storage model. Model performance was strongly influenced by the training data set. The authors found that, while the ANN did well during interpolation, predictions made by ANNs outside the range of the training data set were not encouraging. The process of trying to make ANNs adaptive was computationally very demanding, because the entire training process needed to be repeated with each new data pair. As the lead time for forecasting increased, ANN performance deteriorated. By comparison, ANNs were found to be marginally better than fuzzy inference-based techniques.

Smith and Eli (1995) applied a back-propagation neural network model to predict peak discharge and time to peak over a hypothetical watershed. By representing the watershed as a grid of cells, it was possible for the authors to incorporate the spatial and temporal distribution information of rainfall into the ANN model. The peak discharge and the time to peak corresponding to each rainfall pattern were computed using a linear and nonlinear reservoir model and served as target outputs for the ANN model. Many such patterns formed the training set. For single-storm events, the peak discharge and the time to peak were predicted well by the neural network, both during training and testing. The authors were less successful for multiple- storm events. One reason cited for this was the insufficient number of nodes in the output layer. In a separate application dealing with multiple storms, Smith and Eli (1995) represented the entire hydrograph by a Fourier series with 21 coefficients, rather than just two attributes as in single-storm events. The ANN output layer now consisted of 21 nodes corresponding to the Fourier coefficients. Using this method, the authors found the prediction of the entire hydrograph to be very accurate for multiple storm events.

The issue of enhancing the training speed using a three-layer network was addressed by ***Hsu et al.,*** (1995) and ***Gupta et al.,*** (1997). These studies advocated the linear least squares simplex (LLSSIM) algorithm, which partitions the weight space to implement a synthesis of two training strategies. The authors applied this technique to daily rainfall-runoff modeling of the Leaf River Beam near Collins, Mississippi. The performance of neural networks was compared with the linear ARMAX time series model and the conceptual SACSMA model. The study showed that ANN was performing better than the other two methods with predictions from ANN being more close to observed values on greater number of occasions. Gupta et al. (1997) concluded that the LLSSIM is likely to be a better training algorithm than back-propagation or conjugate gradient techniques, especially in the absence of a good initial guess of weights.

In another related study over the Leaf River Basin, ***Hsu et al.,*** (1997) used a three-layer feedforward ANN and a recurrent ANN to model daily rainfall-runoff. They concluded that the feedforward ANN needed a trial-and-error procedure to find the appropriate number of time-delayed input variables to the model and also was not suitable to distributed watershed modeling. On the other hand, the recurrent ANN was able to provide a representation of the dynamic internal feedback loops in the system, eliminating the need for lagged inputs and resulting in a compact weight space. However, both ANNs performed equally well at runoff prediction.

Carriere (1996) developed a virtual runoff hydrograph system that employed a recurrent back-propagation artificial neural network to generate runoff hydrographs. The author concluded from thei study that the neural network could predict runoff hydrographs accurately, with good agreement between the observed and predicted values.

In a study by ***Minns and Hall*** (1996), data for network training consisted of model results from one storm sequence, and two such sequences were generated for testing. Each storm sequence was generated using a Monte Carlo procedure that preserved predetermined storm characteristics. For each such storm sequence, the corresponding runoff sequence was constructed using a simple nonlinear model for flood estimation (called RORB) that allowed for different levels of nonlinearly in the response. A three-layer network with back-propagation was used. It was found that ANN performance was hardly influenced by level of nonlinearity, with performance deteriorating only slightly for high levels of nonlinearity. However, when the network was required to predict ''out of range'' of the standardized values, the performance dropped significantly, suggesting that ANNs are not very good extrapolators.

Haykin (1994) showed that design of a supervised neural network might be pursued in a number of different ways. While the back-propagation algorithm for the design of a multilayer perceptron (under supervision) may be viewed as an application of stochastic approximation, radial-basis function (RBF) networks can be viewed as a curve-fitting problem in a high-dimensional space.

Bonafe et al., (1994) assessed the performance of a neural network in forecasting daily mean flow from the upper Tiber River basin in central Italy. The previous discharge, daily precipitation, daily mean temperature, total rainfall of the previous five days, and mean temperature over the previous ten days were selected as ANN inputs. They concluded that the ANN was able to yield much better performances than ARMA models.

Mason et al., (1996) used RBF networks for accelerating the training procedure as compared with regular back-propagation techniques. Data were generated using the Simulation Program for Interactive Drainage Analysis (SPIDA) model. The network output was runoff based on inputs consisting of time, rainfall intensity, cumulative rainfall, and derivative of rainfall intensity. The authors concluded that, while RBF networks did provide for faster training, such networks require the solution of a linear system of equations that may become ill conditioned, especially if a large number of cluster centers are chosen.

Jayawardena and Fernando (1995, 1996) and ***Fernando and Jayawardena*** (1998) also used RBF methods for flood fore-casting. They illustrated the application of (RBF) artificial neural networks using an orthogonal least squares algorithm (OLS) to model the rainfall-runoff process. The input nodes contained three antecedent discharges and two

rainfall values—that is, $Q(t\text{-}1)$, $Q(t\text{-}2)$, $Q(t\text{-}3)$, $R(t\text{-}2)$, and $R(t\text{-}3)$. The output was the discharge at the current hour, $Q(t)$. Both a multiple layer perceptron (MLP) neural network and a RBF network were developed and compared with the statistical ARMAX model. Even though both the RBF and MLP networks performed well, it was found that RBF networks could be trained much faster than MLP networks using back propagation. Both networks performed better than theARMAX model.

Shamseldin (1997) compared ANNs with a simple linear model, a season-based linear perturbation model, and a nearest neighbor linear perturbation model. The results suggested that the neural networks generally performed better than the other models during training and testing.

In an effort to relate runoff to precipitation, snow and temperature, and previous streamflows, ***Tokar and Markus*** (1997) employed ANNs to predict monthly flows on the Fraser River near Granby, Colordo, and daily flows on the Raccoon Creek near Bayard, Iowa. The WATBAL and the SAC-SMA models were used as alternative tools for comparison purposes over the two watersheds, respectively. For the Fraser River, the ANN produced better results than the WATBAL model. In the case of Raccoon Creek, the best neural network was chosen among four alternatives and produced comparable results to the conceptual SAC-SMA model.

Dawson and Wilby (1998) used a three-layer back-propagation network to determine runoff over the catchments of the Rivers Amber and Mole. ANN inputs were past flows and averages of past rainfall and flow values. The ANN output consisted of predicting future flows at 15 min intervals up to a lead time of six hours. Their results show that ANNs performed about as well as an existing forecasting system that required more information. When compared with actual flows, the ANNs appeared to overestimate low flows for the Mole River.

Tokar and Johnson (1999) reported that ANN models provided higher training and testing accuracy when compared with regression and simple conceptual models. Their goal was to forecast daily runoff for the Little Patuxent River, Maryland, with daily precipitation, temperature, and snowmelt equivalent serving as inputs. They found that the ANN that was trained on wet and dry data had the highest prediction accuracy.

Thandavesvara and Sajikumar (2000) classified various river basins using ANN. The usefulness of the ART-II in clustering the basins into different homogeneous groups was demonstrated. ART is an unsupervised competitive network that is used to cluster the analog input pattern into different groups based on the proximity of each pattern with others. ART follows the incremental learning and, hence, takes care of the well-known stability-plasticity dilemma (Carpenter and Grossberg 1987). It was found that the network can cluster the data into proper geographical regions if the data set represents changes in the characteristics among the geographical regions. The RMSE and NRMSE indicated that the ART-II clustering improves the performance of the MLP network prediction. Hence, it was concluded that ART-II network could be used to identify the homogeneous groups in geographical space as well as in data hyperspace. Furthermore, it was suggested that the data from more basin numbers have to be employed to increase the reliability of the prediction of the runoff model.

Elshorbagy et al., (2000) did a performance evaluation of artificial neural networks for runoff prediction. Three techniques were adopted for comparison. They were: Linear

Regression Analysis (LRA), Nonlinear Regression Analysis (NRA) and ANN. Six testing experiments were conducted. The results of the six experimental tests indicated that the performance of ANN-based models was better but dependent on the data input structure. The authors pointed out that while using the ANN based models, one should be cautious to see whether the data set encompasses the entire data patterns or not. The results showed that ANN-based models show better predictionability than the NRA models for cases where unsufficient amount of data is encountered during the training phase. However, in cases of small training data sets, even the LRA models (with their large ability to generalize) may prove suitable candidates for consideration.

Lihua Xiong et al., (2001) used the first-order Takagi–Sugeno fuzzy system to develop a non-linear combination of rainfall-runoff models. With a plethora of watershed rainfall-runoff models available for flood forecasting and more than adequate computing power to operate a number of such models simultaneously, we ca n now combine the simulation results from the different models to produce the combination forecasts. In this paper, the first-order Takagi–Sugeno fuzzy system is introduced and explained as the fourth combination method (besides other three combination methods tested earlier, i.e. the simple average method (SAM), the weighted average method (WAM), and the neural network method (NNM)) to combine together the simulation results of five different conceptual rainfall-runoff models in a flood forecasting study on eleven catchments. The comparison of the forecast simulation efficiency of the first-order Takagi–Sugeno combination method with the other three combination methods demonstrates that the first-order Takagi–Sugeno method is just as efficient as both the WAM and the NNM in enhancing the flood forecasting accuracy. Considering its simplicity and efficiency, the first-order Takagi–Sugeno method is recommended for use as the combination system for flood forecasting.

Liong et al., (2001) formulated a derivation of Pareto front (The plot of the objective functions whose non-dominated vectors are in the Pareto optimal set is called the Pareto front) using GA and ANN. The paper presents a new genetic algorithm (GA) based calibration scheme, accelerated convergence GA (ACGA), which generates a limited number of points on the Pareto front. A neural network (NN) is then trained to compliment ACGA in the derivation of other desired points on the Pareto front by mimicking the relationship between the ACGA-generated calibration parameters and the model responses. ACGA has been proposed in the study and mainly differs from SGA (Goldberg 1989) in (1) the initial population generator; and (2) the chromosomes selection scheme. Results show that ACGA has the following advantages over other GAbased schemes such as SGA, VEGA, MOGA, and NSGA:

- The initial chromosomes cover a wider spectrum of the search space.
- The convergence rate is faster.
- The resultant Pareto front is more optimal.

Hydrologists and engineers need methods to disaggregate hourly rainfall data into subhourly increments for many hydrologic and hydraulic engineering applications. Burian et al. (2001) presented a training model of ANN to perform such a rainfall disaggregation. The research presented in the paper evaluated the influence on performance of several ANN model characteristics and training issues including data standardization, geographic location of training data, quantity of training data, number of training iterations, and the number of hidden neurons in the ANN. Results from this study suggested that data from rainfall-gauging stations within several hundred kilometers of the station to be disaggregated were adequate for training the ANN rainfall disaggregation model. Further, the authors found that the number of training iterations, the limits of data standardization, the number of training data sets, and the number of

hidden neurons in the ANN exhibited varying degrees of influence over the ANN model performance.

François Anctil et al., (2003) studied the short-term forecasting improvement afforded by the inclusion of low-frequency inputs to artificial neural network (ANN) rainfall-runoff models that are first optimized by using only fast response components, i.e. using stream flow and rainfall as inputs. Ten low-frequency ANN input candidates are considered: the potential evapotranspiration, the antecedent precipitation index (API_i, i=7, 15, 30, 60, and 120 days) and a proposed soil moisture index time series (SMI_A, for A=100, 200, 400 and 800 mm). As the ANNs considered are for use in real-time lead-time-L forecasting, forecast performance is expressed in terms of the persistence index, rather than the conventional Nash–Sutcliffe index. The API_i are the non-decayed moving average precipitation series, while the SMI_A are calculated through the soil moisture accounting reservoir of the lumped conceptual rainfall-runoff model GR4J. Results, based on daily data of the Serein and Leaf rivers, reveal that only the SMI_A time series are useful for one-day-ahead stream flow forecasting, with both the potential evapotranspiration and the API_itime series failing to improve the ANN performance.

Gwo-Fong Lin and Lu-Hsien Chen (2003) used the radial basis function network (RBFN) to construct a rainfall-runoff model, and the fully supervised learning algorithm is presented for the parametric estimation of the network. The fully supervised learning algorithm has advantages over the hybrid-learning algorithm that is less convenient for setting up the number of hidden layer neurons. The number of hidden layer neurons can be automatically constructed and the training error then decreases with increasing number of neurons. The early stopping technique that can avoid over-fitting is adopted to cease the training during the process of network construction. The proposed methodology is finally applied to an actual reservoir watershed to find the one- to three-hour ahead forecasts of inflow. The result shows that the RBFN can be successfully applied to build the relation of rainfall and runoff.

Jain and Indurthy (2003) did a comparative analysis of event-based rainfall-runoff modeling techniques using deterministic, statistical, and artificial neural networks. Results from two UH models, four regression models, and two ANN models were reported. The performance of each model structure was evaluated using common performance criteria. The results obtained in the study demonstrated that (1) the ANN models were able to consistently outperform the conventional models in terms of certain performance criteria, barring a few exceptions; (2) the ANN models are able to provide a better representation of an event-based rainfall-runoff process as compared to the conventional models; and (3) the multiple hidden layer 10–12–14–1 ANN model was the best model among all of the models developed in this study. A comparative analysis of all of the modeling techniques revealed that the ANN was the most suitable technique, the deterministic UH method was the next best technique, and statistical regression may not be very suitable for the purpose of modeling an event-based rainfall-runoff process.

Kralisch et al., (2003) used neural network approach for the optimisation of watershed management. Neural networks have been used in the area of hydrologic modeling since around 1995. In most of these approaches the underlying hydrologic processes are considered as 'black box' systems where inputs (e.g. antecedent rainfall and flow) and outputs (usually flow) are the inputs and outputs of the catchment as a whole. In contrast, the present approach does presuppose a detailed scg-based model of the catchment's physical characteristics and requires some sort of pre-processing. wasmod (water and substance simulation model) was used for both purposes. Its computations form the basis

for a good initialization of the topology, the weights, and the activation functions of the neural net used for the optimization.
Another field where neural net technology is applied is that of multi-objective optimization. in this case neural networks are used to find solutions to optimization problems with multiple, possibly competing, objectives. The system that was aimed at can be considered as a combination of both—neural net based hydrological modeling and multi-objective optimization. it must be able (i) to adequately represent the hydrologic processes on a field scale (e.g. in order to justify fertilization regulations for the affected farmers); and (ii) to search for a configuration of input parameters that implies lowest costs (where costs represent a whole bundle of competing objectives, like water quality and expenses for compensation payments).

Several studies had used artificial neural networks (ANNs) to estimate local or regional precipitation/rainfall on the basis of relationships with coarse-resolution atmospheric variables. None of these experiments satisfactorily reproduced temporal intermittency and variability in rainfall. ***Olsson et al.,*** (2004) attempted to improve performance by using two approaches: (1) couple two NNs (neural networks) in series, the first to determine rainfall occurrence, and the second to determine rainfall intensity during rainy periods; and (2) categorize rainfall into intensity categories and train the NN to reproduce these rather than the actual intensities. The results indicated that (1) two NNs in series may greatly improve the reproduction of intermittency; (2) longer data series are required to reproduce variability; (3) intensity categorization may be useful for probabilistic forecasting; and (4) overall performance in this region is better during winter and spring than during summer and autumn.

Jieyun Chen and Barry J. Adams (2005) integrated artificial neural networks with conceptual models in rainfall-runoff modeling. Based on this integrated approach, the spatial variation of rainfall, the heterogeneity of watershed characteristics and their impacts on runoff can be investigated by the development of a semi-distributed form of conceptual rainfall-runoff models. As a result, in each subcatchment, the runoff generation and water budget among different runoff components including surface runoff and groundwater can be simulated with consideration of the spatially distributed model parameters and rainfall inputs.

In the runoff routing, instead of a linear superposition of the routed runoff from all subcatchments in the formation of total runoff output at the entire watershed outlet as traditionally performed in a semi-distributed form of conceptual models, artificial neural networks as effective tools in nonlinear mapping are employed to explore nonlinear transformations of the runoff generated from the individual subcatchments into the total runoff at the entire watershed outlet. The feasibility of this new approach has been demonstrated in the study based on the selection of three different types of conceptual rainfall-runoff models. The verification results from the three conceptual models indicate that the approach of integrating artificial neural networks with conceptual models presented in this paper shows promise in rainfall-runoff modeling.

Traditional conceptual rainfall–runoff models in the lumped form are usually developed without consideration of the spatial variation of rainfall and the heterogeneity of the watershed geo-morphological nature. As an improvement to traditional conceptual models of the lumped form, ***Chen and Adams*** (2006) proposed a semi-distributed form of the Tank model coupled with artificial neural networks (ANNs). As a result, authors were able to investigate the effect of spatial variations of rainfall and model parameters by dividing the entire catchment into a number of subcatchments and applying the

spatially varied rainfall inputs and parameters to each subcatchment. Furthermore, in contrast to the linear summation commonly used in watershed routing that usually regards the total simulated runoff at the entire catchment outlet as a linear superposition of the routed runoff from all individual subcatchments, artificial neural networks were employed to explore nonlinear transformations of the runoff generated from the individual subcatchments into the total runoff at the entire watershed outlet.

Garbrecht (2006) investigated the performance of three artificial neural network (ANN) designs that accounted differently for the effects of seasonal rainfall and runoff variations for monthly rainfall-runoff simulation on an 815 km^2 watershed in central Oklahoma. The three designs were: (1) Design 1 consisted of total rainfall for the current month and for each of the previous two months; (2) In Design 2, a separate and independent ANN was developed for each calendar month, thereby accounting explicitly for seasonal variations and (3) In Design 3, the ANN considered rainfall and runoff data from three consecutive months for the training phase, yet for the simulation phase only rainfall-runoff data for the center months were used to drive the ANN. Following conclusions were drawn from the study:

1) ANN design had a significant impact on rainfall-runoff simulation performance and deserves careful consideration when setting up an ANN
2) Design 1 gave better simulated results than Design 2.
3) Base flow during low rainfall months was not well simulated by any ANN design because of the weak relationship between rainfall and base flow during dry months
4) Design 3 and Design 2 produced similar rainfall-runoff simulation results with Design 2 being somewhat simpler and displaying overall better performance.
5) All three ANN designs systematically underpredicted high and overpredicted low runoff values.
6) Base flow during low rainfall months was not well simulated by any ANN design because of the weak relationship between rainfall and base flow during dry months.

Tayfur and Singh (2006) presented the development of artificial neural network (ANN) and fuzzy logic (FL) models for predicting event-based rainfall runoff and tested these models against the kinematic wave approximation (KWA). The results showed that measured event based rainfall-runoff peak discharge data from laboratory flume and experimental plots were satisfactorily predicted by the ANN, FL, and KWA models. Similarly, all the three models satisfactorily simulated event-based rainfall-runoff hydrographs from experimental plots with comparable error measures. ANN and FL models also satisfactorily simulated a measured hydrograph from a small watershed 8.44 km^2 in area. The authors concluded that:

1. The models need to be recalibrated with sufficient site specific measured data when applied at larger scales such as large watersheds;
2. ANN and FL models are based on the observations of the physical system and consequently they require sufficiently long historical data for describing the process under consideration
3. The KWA equation is valid across different scales. It only requires the recalibration of the model parameters pertinent to the specific area.
4. It is easier to construct ANN and FL models than KWA model.

Carcano et al., (2005) attempted to simulate potential scenarios in Rainfall-Runoff (R-R) transformation at daily scale, mainly perceived for the control and management of water resources, using feed-forward multilayer perceptrons (MLP) and, subsequently, Jordan Recurrent Neural Networks (JNN). In the report the author indicated that training a complete RNN involves a high computational cost and can lead to stability problems,

(several simplified models have been proposed in the literature, which can be trained by conventional back –propagation). In the study Jordan neural networks (JNN) had been considered; where, recurrency is accomplished through the following updating rule:

$$X_i(t) = \alpha \cdot X_i(t-1) + Y_i(t-1)$$

where X_i is the *i-th* component of the additional vector X, built at given time t from previous outputs i Y and α is the strength coefficient to be chosen by the user. Their procedure entails two different approaches: one with "rainfall memory effect" where Y_i (through the functional dependence) are rainfall data and a_1 is the strength coefficient, and another one with runoff memory effect with a_2 coefficient, where Y_i are network's calculated discharges; and traditional recurrent procedure occurs.

8.5 ANN Applications in Modeling Stream Flows

Streamflows are often treated as estimates of runoff from watershed. In some studies, streamflow prediction was an intermediate goal. In one of the earlier applications involving streamflows, Kang et al., (193) used ANNs and autoregressive moving average models to predict daily and hourly streamflows in the Pyung Chang River basin in Korea. This preliminary study concluded that ANNs are useful tools for forecasting streamflows.

In a more detailed study along similar lines, ***Karunanithi et al.***, (1994) were interested in estimating streamflows at an ungauged site on the Huron River in Michigan, based on data from USGS stream gauging stations located 30 km upstream and 20 km downstream of the sampling site. Neural networks were found to better predict high stream flow events, while both methods predicted low streamflows fairly well. These authors stated that NNs were capable of adapting their complexity to accommodate temporal changes in historical streamflow records. They also found that including another gauging station that supposedly had little or no effect on streamflows at the gauging site caused the performance of the regression technique to deteriorate, while the ANN performance was not affected. The authors claimed that ANNs are likely to be more robust when noisy data is present in the inputs. The authors found lag time to be important in predicting streamflows. This reflects the longer memory associated with streamflows. The authors did not use any statistical techniques to evaluate the lag time and include it in the network architecture.

Markus et al., (1995) used ANNs with the back-propagation algorithm to predict monthly streamflows at the Del Norte gauging station in the Rio Grande Basin in Southern Colorado. The inputs used were snow water equivalent alone, or snow water equivalent and temperature. They used Periodic Transfer Functions (PTFs) to predict stream flows based on similar inputs as an alternative form of prediction. They looked at forecast bias and root mean square error for assessing model performance. The results indicated that both ANNs and PTFs did a good job of predicting stream flows, and that including temperature as input improved model performance.

Poff et al., (1996) used ANNs to evaluate the changes in stream hydrograph from hypothetical climate change scenarios based on precipitation and temperature changes. The synthetic daily hydrograph was generated based on historic precipitation and temperature as inputs. They studied two streams in the northern United States under different hydro-climatological factors. Three classes of hydrological variables of interest were derived from ANN-generated streamflow output. Mean flow conditions were

composed of mean daily discharge, coefficient of variation of daily flow, and predictability of daily flow. High flow conditions included flood frequency, flood predictability, and flood-free period. The ANNs were particularly geared towards modeling these kinds of hydrologic variables.

Muttiah et al., (1997) also used the cascade-correlation algorithm in their efforts to predict two-year peak discharge from watersheds all over the continental United States. An interesting goal of this work was to investigate the possibility of a single model that could predict peak discharges from local to regional-sized watersheds. They wanted to use data that was easily available from GIS databases. Therefore, network inputs consisted of the log of the drainage basin area, elevation, average slope, and average annual precipitation. The authors claim that ANNs showed some improvement over the standard regression techniques employed by the USGS. Using input vector reduction techniques based on the cascade-correlation method, the authors concluded that drainage area and basin elevations could be used for predicting two-year peak discharges.

Stream rating curves often exhibit hysteresis, with the stagedischarge relationship being different for rising and receding stages. A single relationship is inadequate, while using two separate relationships leads to problems of separation. ***Tawfik et al.,*** (1997) used ANNs, with a saturating linear transfer function, to predict flow discharges at two locations over the Nile River using the stage, H, and the rate of change of stage, dH/dt, as network inputs. ANNs were shown to predict discharge without exhibiting the separation problem associated with a method that uses different regression relationships for the rising and receding portions based on when dH/dt changes sign.

These studies indicate that ANNs have achieved some success in streamflow prediction, particularly when these are desired over a certain range of streamflow values. They have been used for obtaining quick and reliable forecasts. It has been shown to be superior to regression techniques and time series models. Many of the comments presented about rainfall-runoff modeling are applicable to the problem of streamflow prediction as well. A major limitation appears to be in trying to design robust prediction techniques over a wide range of streamflows. It is generally believed that the hydrology of high and low flow events is different from events that are perceived as being normal. Future efforts should be directed towards designing ANNs to account for these different scenarios, in order to represent both normal and extreme conditions.

Kumar et al., (2004) has developed two different networks, namely the feed forward network and the recurrent neural network for forecasting a hydrologic time series. The feed forward network is trained using the conventional back propagation algorithm with many improvements and the recurrent neural network is trained using the method of ordered partial derivatives. The selected ANN models were used to train and forecast the monthly flows of a river in India, with a catchment area of 5189 km^2 up to the gauging site. The trained networks are used for both single step ahead and multiple step ahead forecasting. A comparative study of both networks indicates that the recurrent neural networks performed better than the feed forward networks. In addition, the size of the architecture and the training time required were less for the recurrent neural networks. The recurrent neural network gave better results for both single step ahead and multiple step ahead forecasting. Hence recurrent neural networks are recommended as a tool for river flow forecasting.

9.0 Case Studies

The application of ANN in different fields of water resources was briefly reviewed in section 8. This section describe in detail the application of ANN for reservoir operation with two different case studies.

9.1 Neuro-fuzzy paradigm for reservoir operation

Neuro-fuzzy is a noun that looks like an adjective. Unfortunately, it is also used as an adjective. This system is a kind of mapping problem. It derives fuzzy rules from data to mapping the given set of input-output samples. Recent algorithms add means for validating and editing the trained fuzzy rules, even in the absence of expert knowledge (Panigrahi et. al. 2000). These developments have produced 'learning' fuzzy systems that can automatically extract a set of fuzzy rules from a given data set, without the requirement to initialize the system with known rules at the outset. There are added advantages in what is known as Neuro-fuzzy technique wherein the network produced can be simplified when they search for the best model to represent the data set automatically. Hence its performance is tested for the problem or reservoir operation.

The prediction involved combination of expert's opinion with historical data. These approaches were adopted in this paper with the objective of enhancing the accuracy of prediction in the problem of reservoir operation. The case study used to demonstrate is that of deriving operational policy for the river Vaigai in south of Tamil Nadu in India. The training, testing and validation sets in the model consisted of flow data from 1969-1993 and 1994-1997.

9.1.1 Study area and Database

The model applies to Vaigai basin (N 9° 15'- 10° 20' and E 77° 10' – 79° 15') which is located in south of Tamil Nadu in India (Fig. 25). The catchment area is 2253 sq.km and water spread area 25.9 sq.km. The reservoir is getting water from catchments of two states namely, Tamil Nadu and Kerala. Maximum storage capacity of the dam is 193.84 Mm^3. The period of study is of measured historical data from 1969-'70 to 1997-'98. Flow data from water year 1969-'70 to 1993-'94 is used for training the neural network and developing fuzzy rules. Data from water year 1994-'95 to 1997-'98 is used for validation of the model.

9.1.2 Model Development

Adapted neuro fuzzy inference system (ANFIS) model which was coded using Matlab (Matlab, 1999) version 6.1 was used to create, train and test the fuzzy system for reservoir operation. The Sugeno-type inference systems, based on training data were automatically tuned. Instead of simply using the data to choose the membership function parameters, they were checked to see how they could be chosen among their types using the fuzzy logic toolbox applications. The architechture of ANFIS model is presented in Figure 26. The steps involved in the development of themodel include, construction of membership functions for inflow, storage, demand and release, formulation of fuzzy rules, implication and defuzzification. The data from water year 1969-70 to 1997-98 were used for the study. The total data sets were divided into training (90%) and testing set (10%). In the present study ANFIS model was developed to capture the pattern of operation of reservoir over the past 25 years (300 data pairs) in each time period to make

decision for irrigation releases during the each time period. ANFIS model was solved through Matlab.

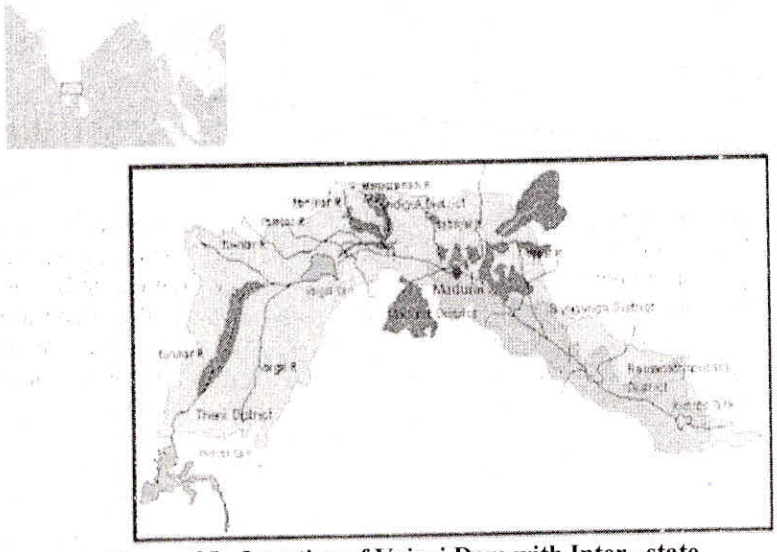


Figure 25: Location of Vaigai Dam with Inter –state Transfer of water from Periyar Dam

ANFIS maps inputs through input membership functions and associated parameters, and then through output membership functions and associated parameters to output to interpret the input/output map. The fuzzy inference system used in this model is shown in Figure 26. This has time period, inflow, initial storage and demand as input fuzzy variable and release as output fuzzy variable. The fuzzy variable time period had *period 1, period 2, period 3 and period 4*, inflow had *very low, low, medium, high and very high* as fuzzy sets, initial storage had *very low, low, medium, high and very high* as fuzzy sets, demand had *low, medium and high* as fuzzy sets and release had *very low, low, medium, high and very high* as fuzzy sets. The parameters associated with the membership functions would change through the learning process. The developed model is used for deriving operation policies for Vaigai system.

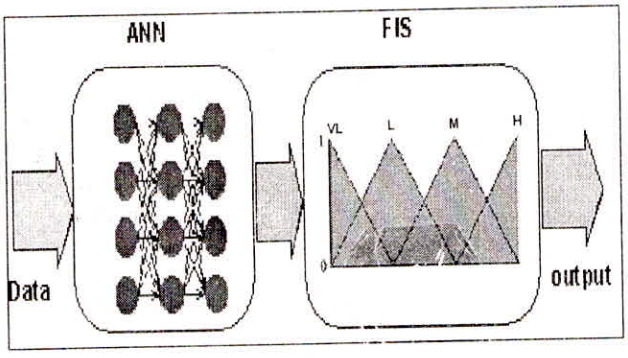


Figure 26: Architecture of Neuro-Fuzzy Model for reservoir operation

9.1.3 Results and discussions

ANFIS model performed pretty good against the test data with error value of 0.03093. When the performance of checking data were plotted against training data, checking error was minimum (5.8107) unto certain point. Then it jumped up to higher value and maintained at a higher value of 7.2359 until final epoch number 15. The performance of ANFIS is compared with the actual releases made for the validation period and is shown in figure 27. This jump represented the point of model over fitting. Hence the ANFIS model parameters associated with the minimum checking error (just prior to this jump point) were selected.

The reservoir operation policy is derived for given inflow, initial storage and demand using the ANFIS code developed through this study. In some cases, data were collected using noisy measurements, and the training data were not representative of all the features of the data that could be presented to the model. That was the reason why model validation is important in any type of modeling. The type of model validation that takes place with the option of checking for model over fitting, and the argument called as checking data set were carried out.

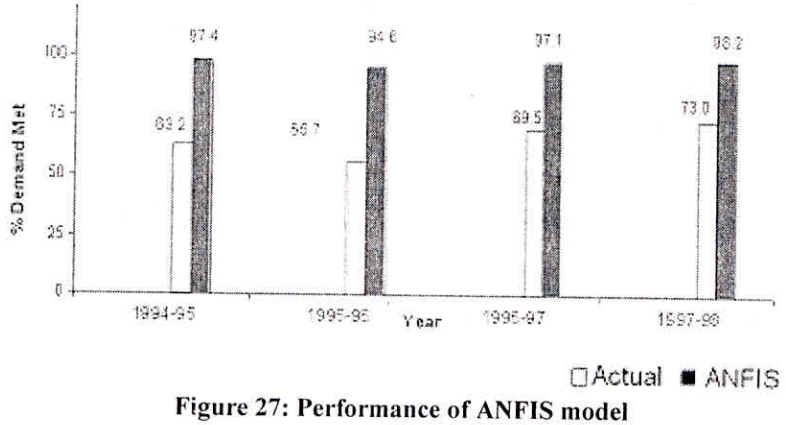


Figure 27: Performance of ANFIS model

9.1.4 Validation of the Model

The input vectors from input/output data sets which were not used to train FIS, were presented to the trained FIS model, to see how well the FIS model predicted the corresponding data set output values. The validation inputs for the year 1994-1997 were given to this interactive model to get the output release. The plot on ANFIS release over target release is also plotted. These results suggested that Neuro-Fuzzy algorithms have the potential to significantly improve usual classification methods for the use in reservoir operation

9.1.5 Conclusions

The results of these models for various conditions of initial storage and inflows can be mapped on to the neuro-fuzzy system. In other words one can apply this fuzzy inference

to a system for which a collection of input/output data are already there or for modeling or model-following or some similar scenario. There is no need to have a predetermined model structure based on characteristics of variables in the system. Rather than choosing the parameters associated with a given membership function arbitrarily, these parameters could be chosen so as to tailor the membership functions to the input/output data. This is the reason for adopting Neuro-adaptive learning techniques incorporated into the ANFIS in the Fuzzy logic toolbox.

9.2 A Neural Network Model for Reservoir Operation

The objectives of the study were set as follows:

1) To develop a stochastic dynamic programming model for derivation of reservoir operation policies, and
2) Comparatively evaluate the performance of the developed neural network model with actual operation and also with that of the standard operating policy.

9.2.1 System for Study

The Sri Ram Sagar Project (SRSP) situated in Andhra Pradesh state, India is considered for the study. This system was built to utilise the Godavari river water for irrigation. The project site is located on the river Godavari at latitude 8°58' N and longitude 78 °20' E at Pochampad village in Nizamabad District in Andhra Pradesh state, India.

The capacity of the reservoir at maximum water level (MWL) is 3443 Mm^3. The capacity of the reservoir at dead storage level (DSL) is 873 Mm^3. There are three canals taking off from the reservoir named as the Saraswati, Laxmi, Kakatiya. The Saraswati canal is 47 km length and its capacity is 42.47 cumec. The Laxmi canal is 3.5 km length and its capacity is 14.13 cumec. The Kakatiya canal is 284 km having a capacity of 274.7 cumec. The total area to be irrigated is nearly 4,10,000 hectares. The schematic diagram of the reservoir system is shown **in Fig. 28**

The following data were collected for the study at Sri Ram Sagar Project, Pochampad village in Andhra Pradesh.

1. Monthly inflows for the 26 years (June 1970 to Jun_ 2006)
2. Daily data of following were collected for 15 years (June 1991 - June 2006)
    i) Reservoir storage level
    ii) Inflows into the reservoir.
    iii) Releases through the three canals
3. Evaporation losses and daily data on maximum and minimum daily temperatures and daily rainfall were also collected for a period often years (1997-2006).

9.2.2 Methodology

The methodology proposed in this study consists of two phases; in the first phase, a

stochastic dynamic programming model was developed to determine the optimal final storage volumes for each of initial storage and inflow ranges for all the time periods. In the second phase, a neural network model was developed with inputs as current time period, initial storage, and inflow in the current time period, inflow in the previous time period and the output as the corresponding optimal release which was obtained in the first phase. The performance of the developed neural network model was compared with the standard operating policy and also with the actual releases. Here only the neural network modeling part is explained.

9.2.3 Training of ANN model

In training of neural network phase, enough training samples including the inputs and target output are required. To generate the inputs and corresponding outputs either the actual data may be used or the data may be derived from the existing optimization techniques, such as linear programming, dynamic programming, non-linear programming and simulation methods may be used in this step. For the study, the stochastic dynamic programming model was developed and used to derive optimal releases. The initial storages, inflows in both current period and in previous period were feed to neural network model as input to derive corresponding releases as output patterns.

The program for the neural network model was developed using MATLAB. MATLAB *(matrix laboratory)* is an interactive program with scientific and engineering numeric calculations. The developed ANN model for every month has four input nodes, 15 hidden nodes and one output node. The optimal releases obtained from the stochastic dynamic programming for each of initial storage, inflow in the current period, inflow in the previous period and current time period were given as the set of input pattern and, the corresponding releases as output patterns to the neural network model.

Let Q_{ij}, S_{ij}, t, and R_{ij} denote inflow, storage, time period and release in period in the j^{th} sample, respectively. $Q_{i-1,j}$ is the inflow in the previous period for j^{th} sample. Release is non-linear and uncertain and it is function of inflow, storage. To the neural network model set of correspond Q_{ij}, S_{ij}, t, $Q_{i-1,j}$ are given as input patterns and the corresponding R_{ij} values are given as output patterns. The major task is to determine the training parameters and the weights of connections between the input layer to hidden layer and hidden layer to output layer. 1

Most neural networks are constructed using the sigmoidal unit with a logistic activation function. The output response of a typical sigmoid unit is bounded over a 0.0 to 1.0 range. Thus the input and output data need to be normalized to the range of 0.0 to 1.0 using the known maximum value of the series. In this study, inflows, releases and storage's were normalized to the range of 0.0 to 1.0 using the following equations before

feeding into the neural network.

$$Q_{ij} = \frac{Q_u}{Q_{ij}^{max}}$$

$$S_{ij} = \frac{S_u}{S_{ij}^{max}}$$

$$R_{ij} = \frac{R_{ij}}{R_{ij}^{max}}$$

where

S_{ij}^{max} = maximum storage level during the time period i

Q_{ij}^{max} = maximum value of inflow during the time period i

R_{ij}^{max} = maximum release during the time period i

Neural networks are trained with a set of typical input/output pairs called training sets. The final weight vector of a successfully trained neural network represents its knowledge about the problem. Thus, at the beginning of the training of the network, weights are initialized with a set of uniform random values. In this study, the network weights are initialized with a set of uniform random values drawn between -0.5 to 0.5. During training, the weights are adjusted so as to reduce the residual error of the training set. The training parameters namely learning rate (α), momentum factor (β) may either kept constant or allowed to vary in order to facilitate faster convergence. In this study, variation of parameters is allowed during learning and resulted in faster convergence. The values of parameters α and β were set initially to 0.1 and 0.5 respectively, and improved by a step of 0.01 and 0.1 respectively during training up to desired convergence.

The number of hidden nodes is usually greater than number of input nodes. It can be seen that as the number of nodes increases, the mean sum squared error is decreased. An optimal value of 15 nodes in the hidden layer is adopted for study. The neural network adopted in this study can be denoted as [4 x 15 x 1] that corresponds to number of input nodes, hidden nodes, and output nodes respectively. The network was trained by the back propagation method. The interconnected weights between the input layer to hidden layer and the hidden layer to output layer are initialized by random numbers and the program was executed until the sum squared error between the actual outputs and desired outputs is minimum. Now trained neural network can be used to derive the optimal releases, given that the time period, initial storage, inflow in time period, and inflow in proceeding time period.

After the network was trained, the neural network was used to derive the releases for two years (June 2004-May 2006). The initial storage at beginning of month June 2004, inflow for the month of June 2004 and inflow for preceding month May 2004 inflow and time period were fed as input to the neural network model and the release was obtained and this release is used to compute the final storage for the month June. This final storage is given as initial storage of month July and the other data for the month July were fed to the neural network and release for July month was obtained. This procedure was repeated for all the months over the period of two years. To compare the performance of neural network model the releases were also derived using the standard operating policy (Loucks et al., 1981) for the same two years. The standard operating rule can be described as follows. For every season it is required to determine the release as a function both of target releases and of water availability, defined as the sum of the, inflow and the storage at the beginning of the season.

Three classes of rules can be defined for the standard operating policy as follows:

1. If the water availability is insufficient to meet the target requirements then the rule assumes that the reservoir will be emptied in order to try to meet the demand.
2. If there is enough water to meet the target demand, then the rule depicts that release will be equal to the required demand of water.
3. If the available water minus the demand exceeds storage capacity, then all water in excess of storage capacity must be spilled.

The releases for the same two years were also derived using the standard operating policy. The releases derived from neural network, and standard operating model with that of actual operation are presented in Table 4 for two years. It can be seen that the neural network model resulted in more releases thus enabling more area being brought under irrigation. On the other hand, the standard operating policy resulted in comparatively less release which may be due to the fact that the release rules of SOP do not take into account the future requirements. Fig 29 shows the comparison of the derived releases from the three methods.

Table 4 Comparison of Annual Release in (Mm^3)

Year	Month	Actual operation release (Mm^3)	Standard operating policy release (Mm^3)	Neural network model release (Mm^3)
2004	*Jun*	84	74	102
	Jul	142	163	185
	Aug	348	377	370
	Sep	328	352	350
	Oet	284	335	324
	Nav	305	325	295
	Dec	214	317	326

2005	Jan	138	101	126
	Feb	137	179	159
	Mar	128	14	98
	Apr	15	15	33
	May	15	16	68
	Jun	64	17	62
	July	287	263	316
	Aug	302	377	381
	Sep	285	352	344
	Oct	102	135	112
	Nov	108	117	178
	Dec	301	317	355
2006	Jan	215	336	297
	Feb	302	288	302
	Mar	153	187	178
	Apr	21	18	36
	May	35	15	56

Table 5 shows the annual release for the two years obtained from standard operating policy and neural network model, and actual releases. It may be noted that the standard operating policy resulted in less releases in during the non-monsoon periods which resulted in more deficit. It can also be seen that the neural network model resulted releases that would result in of90% of the demand for the period of June 2004-May 2005. For period June 2005- May 2006, the release could meet 97% of the demand. On the other hand, SOP resulted in releases equal to 84% and 90% the actual operation for both the years showed that about 80% of the demand could only met during the two years under study. Thus it is evident that the releases resulted from the neural network model is superior to both releases derived from standard operating policy and that of actual operation.

Table 5 Annual Release from SOP and ANN Model

	Standard policy operation		Neural network model		Actual operation	
In %	Annual release	Annual Deficit	Annual release	Annual Deficit	Annual release	Annual Deficit
June 04-May 05	2268	16	2436	10	2138	20
June 05-May 06	2422	10	2617	3	2175	19

9.2.4 Conclusions

The major conclusions were arrived from this study are as follows.

(i) The neural network model with inputs as current time period, initial storage for that period, inflow in the current period and previous period and releases as outputs found to be a suitable combination for derivation of releases.

(ii) The number of nodes in the hidden layer plays an important role in the convergence of the outputs from the neural networks towards the actual values. For the reservoir system studied 15 nodes in the hidden layers resulted in faster convergence.

(iii) When the training parameters namely learning rate and momentum factor are allowed to vary within the iterations resulted in faster convergence rather than assuming constant values throughout the training.

(iv) Optimal releases resulted from neural network model when compared to that of actual releases and releases derived from standard operating policy models reveals that almost 95% of the area that of the existing area under the command can be irrigated if the release decisions from the neural network model are followed.

(v) Standard operating policy model considers only the availability of the water in the current period to decide the releases and do not consider the requirements for later periods. The study also showed that even through standard operating policy model meet the demand of monsoon period with out much deficit, it failed to meet the demand in non-monsoon season.

(vi) The study proved that the combination of stochastic dynamic programming model and neural network model is an effective tool for release decision-making, given the conditions of the reservoir system namely storage and possible inflows. Since the trained neural network has the complete knowledge about the relationship between' the input variables and output variables, it can be used to derive the releases within a reasonable time rather than solving an optimization model. The developed neural network model is useful for reservoir operators in the decision making process.

The neural network model has been found to be effective for determining the releases from a single reservoir system. The feasibility of neural network model for release decision making in a multi-purpose single reservoir system, multipurpose multi-reservoir system need to be studied.

10.0 Summary

In summary, artificial neural networks are one of the promises for the future in computing. They offer an ability to perform tasks outside the scope of traditional processors. They can recognize patterns within vast data sets and then generalize those patterns into recommended courses of action. Neural networks learn, they are not programmed. Yet, even though they are not traditionally programmed, the designing of neural networks does require a skill. It requires an "art." This art involves the understanding of the various network topologies, current hardware, current software tools, the application to be solved, and a strategy to acquire the necessary data to train the network. This art further involves the selection of learning rules, transfer functions, summation functions, and how to connect the neurons within the network.

Then, the art of neural networking requires a lot of hard work as data is fed into the system, performances are monitored, processes tweaked, connections added, rules modified, and on and on until the network achieves the desired results. These desired results are statistical in nature. The network is not always right. It is for that reason that neural networks are finding themselves in applications where humans are also unable to always be right. Neural networks can now pick stocks, cull marketing prospects, approve loans, deny credit cards, tweak control systems, grade coins, and inspect work.

Yet, the future holds even more promises. Neural networks need faster hardware. They need to become part of hybrid systems which also utilize fuzzy logic and expert systems. It is then that these systems will be able to hear speech, read handwriting, and formulate actions. They will be able to become the intelligence behind robots who never tire nor become distracted. It is then that they will become the leading edge in an age of "intelligent" machines.

11.0 References

Azmathullah, Md. H., Deo, M. C., and Deolalikar, P. B. (2005). "Neural Networks for Estimation of Scour Downstream of a Ski-Jump Bucket". *Journal of Hydraulic Engineering*. Vol. 131(10), 898-908.

Basheer, I. A., and Najjar, Y. M. (1995). ''Designing and analyzing fixedbed adsorption systems with artificial neural networks.'' *Journal of Envir. Syst.*, 23(3), 291–312.

Bonafe, A., Galeati, G., and Sforna, M. (1994). ''Neural networks for daily mean flow forecasting.'' *Hydr. Engrg. Software V*, W. R. Blain and K. L. Katsifarakis, eds., Computational Mechanics Publications, Southampton, U.K., 1, 131–138.

Carcano, E. C., Bartolini, P., and Muselli, M. (2005). "Recurrent Neural Networks in Rainfall –Runoff modeling at daily scale." *Proceedings of the DANOLD (Device Applications of Nonlinear Dynamics) Conference.*

Carriere, P., Mohaghegh, S., and Gaskari, R. (1996). ''Performance of a virtual runoff hydrograph system.'' *Journal of Water Resour. Plng. and Mgmt.*,ASCE, 122(6), 421–427.

Chaves,P., Tsukatan, T., and Kojiri, T., (2004). "Operation of storage reservoir for water quality by using optimization and artificial intelligence techniques." *Mathematics and computers in simulation.* 67 (4-5), 419-432.

Chen, J., and Adams, B. J. (2006). "Semidistributed Form of the Tank Model Coupled with Artificial Neural Networks" *Journal of Hydrologic Engineering*. Vol. 11(5), 408-417

Coppola Jr., E., Szidarovszky, F., Poulton, M., and Charles, E., (2003). "Artificial Neural Network Approach for Predicting Transient Water Levels in a Multilayered Groundwater System under Variable State, Pumping, and Climate Conditions." *Journal of Hydrologic Engineering*. Vol. 8(6), 348-360.

Coulibaly, P., Anctil, F., Aravena, R., and Bobee, B. (2001). "Artificial neural network modeling of water table depth fluctuations." *Water Resources Research*. Vol. 37, No 4, 885-896.

da Silva, I. N., Nilton J. Saggioro and Jose A. Cagnon (2000). "*Using Neural Networks for Estimation of Aquifer Dynamical Behavior.*" IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN'00)-Volume 6, 203-208.

Daliakopoulos, I. N., Paulin Coulibaly and Ioannis K. Tsanis, (2005). "Groundwater level forecasting using artificial neural networks." *Journal of Hydrology*. 309(1-4), 229-240.

Dawson, C. W., and Wilby, R. (1998). ''An artificial neural network approach to rainfall-runoff modeling.'' *Hydrological Sci.*, 43(1), 47–66.

Dixon, B., (2005). "Applicability of neuro-fuzzy techniques in predicting ground-water vulnerability: a GIS-based sensitivity analysis." *Journal of Hydrology*. 309(1-4), 17-38.

Elshorbagy, A., Simonovic, S. P., and Panu, U. S. (2000). "Performance Evaluation Of Artificial Neural Networks For Runoff Prediction." *Journal of Hydrologic Engineering.* Vol. 5(4), 424-427.

Fernando, D. A. K., and Jayawardena, A. W. (1998). ''Runoff forecasting using RBF networks with OLS algorithm.'' *Journal of Hydrologic Engrg.*, ASCE, 3(3), 203–209.

Francois Anctil, Claude Miche, Charles Perrin and Vazken Andréassian , (2004). "A soil moisture index as an auxiliary ANN input for stream flow forecasting." *Journal of Hydrology.* 286(1-4), 155-167.

French, M. N., Krajewski, W. F., and Cuykendal, R. R. (1992). ''Rainfall forecasting in space and time using a neural network.'' *Journal of Hydrol.*, Amsterdam, 137, 1–37.

Garbrecht, J. D. (2006). "Comparison of Three Alternative ANN Designs for Monthly Rainfall-Runoff Simulation." *Journal of Hydrologic Engineering.* Vol. 11(5), 502-505.

Gorrini, V., and Bersini, H., (1994). "Recurrent Fuzzy Systems", In: *Proceedings of the 3rd Conference on Fuzzy Systems (FUZZ-IEEE 94),* IEEE, Orlando.

Govindaraju Rao, S. (2000). "Artificial Neural Networks in Hydrology II: Hydrologic Applications." ASCE Task Committee on Application of Artificial Neural Networks in Hydrology, *Journal of Hydrologic Engineering.* Vol. 5(2), 124-137.

Gupta, H. V., Ksu, K., and Sorooshian, S. (1997). ''Superior training of artificial neural networks using weight-space partitioning.'' *Proc., IEEE Int. Conf. on Neural Networks,* Institute of Electrical and Electronics Engineers, New York.

Gwo-Fong Lin and Lu-Hsien Chen, (2004). "A non-linear rainfall-runoff model using radial basis function network." *Journal of Hydrology.* 289(1-4), 1-8.

Haykin, S. (1994). *Neural networks: a comprehensive foundation.* Mac- Millan, New York.

Haykin, S. (1994). *Neural Networks,* Prentice Hall, New Jersey.

Hjelmfelt, A. T., and Wang, M. (1993a). ''Artificial neural networks as unit hydrograph applications.'' *Proc., Engrg. Hydrol.*, ASCE, New York, 754–759.

Hjelmfelt, A. T., and Wang, M. (1993b). ''Runoff simulation using ANN.'' *Proc., 4th Int. Conf. in the Application of Artificial Intelligence to Civ. and Struct. Engrg.: NN and Combinatorial Optimization in Civ. and Struct. Engrg.*, B. H. V. Topping and A. I. Khan, eds., Civl-Comp Ltd., Edinburgh, U.K., 517–522.

Hjelmfelt, A. T., and Wang, M. (1993c). ''Runoff hydrograph estimation using artificial neural networks.'' *Proc., ASAE Conference*, American Society of Agricultural Engineers, St. Joseph, Mich.

Hsu, K., Gao, X., Sorooshain, S., and Gupta, H. V. (1997). "Precipitation estimation from remotely sensed information using artificial neural networks." *Journal of Appl. Meteorology*, 36(9), 1176–1190.

Hsu, K., Gupta, H. V., and Soroochian, S. (1997). "Application of a recurrent neural network to rainfall-runoff modeling." *Proc., Aesthetics in the Constructed Envir.*, ASCE, New York, 68–73.

Hsu, K., Gupta, H. V., and Sorooshian, S. (1995). "Artificial neural network modeling of the rainfall-runoff process." *Water Resour. Res.*, 31(10), 2517–2530.

Hsu, K., Gupta, H. V., Sorooshian, S., and Gao, X. (1996). "An artificial neural network for rainfall estimation from satellite infrared imaginary." *Applications of Remote Sensing in hydrol., Proc., 3rd Int. Workshop, NHRI Symp. No. 17*, NASA, Greenbelt, Md.

Hutton, P. H., Sandhu, N., and Chung, F. I. (1996). "Predicting THM formation with artificial neural networks." *Proc., North Am. Water and Envir. Conf.*, ASCE, New York, 3557–3556.

Jain, A., and Prasad Indurthy, S. K. V. (2003). "Comparative Analysis of Event-based Rainfall-runoff Modeling Techniques—Deterministic, Statistical, and Artificial Neural Networks." *Journal of Hydrologic Engineering.* Vol. 8(2), 93-98

Jayawardena, A. W., and Fernando, D. A. K. (1995). "ANN in hydrometeorological modeling." *Proc., 4th Int. Conf. in the Application of Artificial Intelligence to Civ. and Struct. Engrg.: Devel. in NN and Evolutionary Computing for Civ. and Struct. Engrg.*, B. H. V. Topping and A. I. Khan, eds., Civil-Comp, Ltd., Edinburgh, U.K., 115–120.

Jayawardena, A. W., and Fernando, D. A. K. (1996). "Comparison of multi-layer perceptron and radial basis function network as tools for flood forecasting." *Proc., North Am. Water and Envir. Conf.*, ASCE, new York, 457–458.

Jieyun Chen and Barry J. Adams, (2005). "Integration of artificial neural networks with conceptual models in rainfall-runoff modeling." *Journal of Hydrology.* 318(1-4), 232-249.

Karunanithi, N., Grenney, W. J., Whitley, D., and Bovee, K. (1994). "Neural networks for river flow prediction." *Journal of Comp. in Civ. Engrg.*, ASCE, 8(2), 201–220.

Keskin, M. E., and Terzi., O., (2006). "Artificial Neural Network Models of Daily Pan Evaporation." *Journal of Hydrologic Engineering.* Vol. 11(1), 65-70.

Klose, A., Nürnberger, A., Nauck, D., and Kruse, R., (2001). "Data Mining with Neuro-Fuzzy Models". In: A. Kandel, H. Bunke, M. Last (Eds.), *Data Mining and Computational Intelligence*, 1-36, Physica-Verlag, 1-36.

Kralisch, S., Fink, M., Flügel, W. A., and Beckstein, C., (2003). "A neural network approach for the optimization of watershed management." *Environmental modeling & software*. 18(8-9), 815-823.

Kuligowski, R. J., and Barros, A. P. (1998). ''Experiments in short-term precipitation forecasting using artificial neural networks.'' *Monthly Weather Rev.*, 126(2), 470–482.

Kumar, D. N., Raju, R. S., and Sathish, T. (2004). "Riverflow Forecasting using Recurrent Neural Network." *Water Resources Management*. Vol.18, No 2, 143-161.

Kumar, M., Raghuwanshi, N. S., Singh, R., Wallender, W. W., and Pruitt, W. O. (2002). "Estimating Evapotranspiration using Artificial Neural Network." *Journal of Irrigation and Drainage Engineering*. Vol. 128(4), 224-233.

Lihua Xiong, Asaad Y. Shamseldin and Kieran M. O'Connor, (2001). "A non-linear combination of the forecasts of rainfall-runoff models by the first-order Takagi–Sugeno fuzzy system." *Journal of Hydrology*. 245(1-4), 196-217.

Lin, C. T., and Lee, C. C. (1996). *Neural Fuzzy Systems. A Neuro-Fuzzy Synergism to Intelligent Systems,* Prentice Hall, New York.

Lindsay, J. B., Julie Q. Shang., and Kerry Rowe, R. (2002). "Using Complex Permittivity and Artificial Neural Networks for Contaminant Prediction." *Journal of Environmental Engineering*. Vol. 128(8), 740-747.

Liong, S. Y., Soon-Thiam Khu, and Weng-Tat Chan. (2001). "Derivation Of Pareto Front With Genetic Algorithm And Neural Network." *Journal of Hydrologic Engineering*. Vol. 6(1), 52-61.

Loucks, D. P., Stedinger, J. R. and Haith, D.A. (1981), "Water Resources System Planning and Analysis", Prantice Hall, Englewood Cliffs, New Jersey.

Markus, M., Christina W.-S. Tsai, and Demissie, M. (2003). "Uncertainty of Weekly Nitrate-Nitrogen Forecasts Using Artificial Neural Networks." *Journal of Environmental Engineering*. Vol. 129(3), 267-274.

Markus, M., Salas, J. D., and Shin, H.-K. (1995). ''Predicting streamflows based on neural networks.'' *Proc., 1st Int. Conf. on Water Resour. Engrg.*, ASCE, New York, 1641–1646.

Mason, J. C., Price, R. K., and Tem'me, A. (1996). ''A neural network model of rainfall-runoff using radial basis functions.'' *Journal of Hydr. Res.*, Delft, The Netherlands, 34(4), 537–548.

Matlab (1999) *Fuzzy Logic Toolbox, User's guide*, The Mathworks, Inc., Mass.

Minns, A. W., and Hall, M. J. (1996). ''Artificial neural networks as rainfall-runoff models.'' *Hydrologic Sci.*, 41(3), 1996.

Mishra, A., Ray, C. P. E., and Dana W. Kolpin. (2004). "Use of Qualitative and Quantitative Information in NeuralNetworks for Assessing Agricultural Chemical Contamination of Domestic Wells." *Journal of Hydrologic Engineering*.Vol. 9(6), 502-511.

Morshed, J., and Kaluarachchi, J. J. (1998). "Parameter estimation using artificial neural network and genetic algorithm for free-product and recovery." *Water Resour. Res.*, 34(5), 1101–1113.

Muttiah, R. S., Srinivasan, R., and Allen, P. M. (1997). "Prediction of two-year peak stream discharges using neural networks." *Journal of Am. Water Resour. Assoc.*, 33(3), 625–630.

Nagy, H. M., Watanabe, K., and Hirano, M. (2002). "Prediction of Sediment Load Concentration in Rivers using Artificial Neural Network Model." *Journal of Hydraulic Engineering.* Vol. 128(6), 588-595.

Nauck, D., Klawonn, F., and Kruse, R., (1997). *Foundations of Neuro-Fuzzy Systems,* Wiley, Chichester.

Navone, H. D., and Ceccatto, H. A. (1994). "Predicting Indian monsoon rainfall: a neural network approach." *Climate Dyn.*, 10, 305–312.

Olsson, J., Uvo, C. V., Jinno, K.., Kawamura, A., Nishiyama, K., Koreeda, N., Nakashima, T., and Morita, O. (2004). "Neural Networks for Rainfall Forecasting by Atmospheric Downscaling." *Journal of Hydrologic Engineering.* Vol. 9(1), 1-12.

Panigrahi, D. P and Mujumdar, P. P (2000) Reservoir Operation Modeling with Fuzzy logic, *Water Resources Management*, Vol.14, 89-109.

Poff, N. L., Tokar, S., and Johnson, P. (1996). "Steam hydrological and ecological responses to climate change assessed with an artificial neural network." *Limnol. and Oceanog.*, 41(5), 857–863.

Raj Mohan Singh., Bithin Datta., and Ashu Jain. (2004). "Identification of Unknown Groundwater Pollution Sources Using Artificial Neural Networks." *Journal of Water Resources Planning and Management.* Vol. 130(6), 506-514.

Ray, C., and Klindworth, K. K. (1996). "Use of artificial neural networks for agricultural chemical assessment of rural private wells." *Proc., North Am. Water and Envir. Conf.*, ASCE, New York, 1687–1692.

Rizzo, D. M., and Dougherty, D. E. (1994). "Characterization of aquifer properties using artificial neural networks: neural kriging." *Water Resour. Res.*, 30(2), 483–497.

Rogers, L. L. (1992). "Optimal groundwater remediation using artificial neural network and the genetic algorithm," PhD disertation, Stanford University, Stanford, Calif. *Journal Of Hydrologic Engineering / April 2000 / 137.*

Rogers, L. L., and Dowla, F. U. (1994). "Optimization of groundwater remediation using artificial neural networks with parallel solute transport modeling." *Water Resour. Res.*, 30(2), 457–481.

Rogers, L. L., Dowla, F. U., and Johnson, V. M. (1995). "Optimal fieldscale groundwater remediation using neural networks and the genetic algorithm." *Envir. Sci. and Technol.*, 29(5), 1145–1155.

Rogers, L. L., Johnson, V. M., and Dowla, F. U. (1993). "Network dissection of neural networks used in optimal groundwater remediation." *Proc., 2nd USA/CIS Joint Conf. on Envir. Hydrol. and Hydrogeology*, American Institute of Hydrology, Arlington, Va.

Rojas, R. (1993). *Neural Networks - A Systematic Introduction,* Springer-Verlag, Berlin, New York.

Sandhu, N., and Finch, R. (1996). "Emulation of DWRDSM using arti- ficial neural networks and estimation of Sacramento River flow from salinity." *Proc., North Am. Water and Envir. Conf.*, ASCE, New York, 4335–4340.

Schmid, B. H., and Koskiaho, J. (2006). "Artificial Neural Network Modeling of Dissolved Oxygen in a Wetland Pond: The Case of Hovi, Finland." *Journal of Hydrologic Engineering.* Vol. 11(2), 188-192.

Shamseldin, A. Y. (1997). "Application of a neural network technique to rainfall-runoff modeling." *Journal of Hydrol.*, Amsterdam, 199(1997), 272– 294.

Slavisa Trajkovic., Branimir Todorovic., and Miomir Stankovic. (2003). "Forecasting of Reference Evapotranspiration by Artificial Neural Networks." *Journal of Irrigation and Drainage Engineering.* Vol. 129(6), 454-457.

Smith, J., and Eli, R. N. (1995). "Neural-network models of rainfallrunoff process." *Journal of Water Resour. Plng. and Mgmt.*, ASCE, 121(6), 499–508.

Sudheer, K. P., Gosain, A. K., and Ramasastri, K. S. (2003). "Estimating Actual Evapotranspiration from Limited Climatic Data Using Neural Computing Technique." *Journal of Irrigation and Drainage Engineering.* Vol. 129(3), 214-218.

Suen, J. P., and Wayland, E. J. (2003). "Evaluation of Neural Networks for Modeling Nitrate Concentrations in Rivers." *Journal of Water Resources Planning and Management.* Vol. 129(6), 505-510.

Tawfik, M., Ibrahim, A., and Fahmy, H. (1997). "Hysteresis sensitive neural network for modeling rating curves." *Journal of Comp. in Civ. Engrg.*, ASCE, 11(3), 206–211.

Tayfur, G., and Singh, V. P. (2005). "Predicting Longitudinal Dispersion Coefficient in Natural Streams by Artificial Neural Network." *Journal of Hydraulic Engineering.* Vol. 131(11), 991-1000,

Tayfur, G., and Singh, V. P., (2006). "ANN and Fuzzy Logic Models for Simulating Event-Based Rainfall-Runoff." *Journal of Hydraulic Engineering.* Vol. 132(12), 1321-1330.

Thandaveswara, B. S., and Sajikumar, N. (2000). "Classification Of River Basins Using Artificial Neural Network." *Journal of Hydrologic Engineering.* Vol. 5(3), 290-298.

Tohma, S., and Igata, S. (1994). "Rainfall estimation from GMS imagery data using neural networks." *Hydraulic engineering software V*, Vol. 1, W. R. Blain and K. L. Katsifarakis, eds., Computational Mechanics, Southampton, U.K., 121–130.

Tokar, A. S., and Johnson, P. A. (1999). ''Rainfall-runoff modeling using artificial neural networks.'' *J. Hydrologic Engrg.*, ASCE, 4(3), 232–239.

Tokar, A. S., and Markus, M. (1997). ''Artificial neural networks and conceptual models in water management of small basins in the central United States.'' *Proc., 3rd Int. Conf. on FRIEND*, International Association of Hydrological Sciences, Wallingford, U.K.

Yang, C. C., Prasher, S. O., Lacroix, R., Sreekanth, S., Patni, N. K., and Masse, L. (1997). ''Artificial neural network model for subsurfacedrained farmland.'' *Journal of Irrig. and Drain. Engrg.*, ASCE, 123(4), 285–292.

Zhang, S. P., Watanabe, H., and Yamada, R. (1994). ''Prediction of daily water demands byt neural networks.'' *Stochastic and statistical method in hydrology and environmental engineering*, Vol. 3, K. W. Hipel et al., eds., Kluwer, Dordrecht, The Netherlands, 217–227.

Zhu, M., Fujita, M., and Hashimoto, N. (1994). ''Application of neural networks to runoff prediction.'' *Stochastic and statistical method in hydrology and environmental engineering*, Vol. 3, K. W. Hipel et al., eds., Kluwer, Dordrecht, The Netherlands, 205–216.

http://koti.mbnet.fi/~phodju/nenet/NeuralNetworks/NeuralNetworks.html

INDIAN NATIONAL COMMITTEE ON HYDROLOGY (INCOH)

(IHP National Committee of India for UNESCO)

Constituted by the Ministry of Water Resources in 1982

INCOH Activities Related to UNESCO's IHP-VI Program

India is actively participating in IHP-VI activities and a detailed program has been chalked out in accordance with IHP-VI themes towards preparation of reports, taking up research studies, organisation of seminars/symposia at national and regional level, and promotion of hydrological education in the country. It is envisaged to participate in all the relevant and feasible programs identified under the various focal areas of IHP-VI themes as given below.

India's participation in IHP-VI program

Theme	Selected Focal Area
1. Global Changes and Water Resources	Integrated assessment of water resources in the context of global land based activities and climate change
2. Integrated Watershed and Aquifer Dynamics	Extreme events in land and water resources
3. Land Habitat Hydrology	Dry lands
4. Water and Society	Raising public awareness on water interactions
5. Water Education and Training	Continuing education and training for selected target groups

INCOH Publications

Publication of Jalvigyan Sameeksha Journal

To disseminate information and promote hydrological research in the country, INCOH brings out the Journal '*Jalvigyan Sameeksha*' (Hydrology Review Journal). The papers published in the Journal are by invitation only. The Journal is widely circulated amongst major organisations and agencies dealing with water resources.

Publication of State of Art Reports

In pursuance of its objectives to periodically update the research trends in different branches of hydrology, state of art reports authored by experts identified by INCOH from various institutes and organisations in India, are published regularly. These reports are circulated free of cost to central and state government agencies including academic and research organisations.

Printed at : GUPTA PRINTING PRESS, ROORKEE
Ph. : (01332) 271422 (O) **Mob.** : 9412021708