ARTIFICIAL NEURAL NETWORKS TECHNOLOGY

A DACS State-of-the-Art Report

Contract Number F30602-89-C-0082
(Data & Analysis Center for Software)

ELIN: A011

August 20 1992

Prepared for:

Rome Laboratory
RL/C3C
Griffiss AFB, NY 13441-5700

Prepared by:

Dave Anderson and George McNeill

Kaman Sciences Corporation
258 Genesse Street
Utica, New York 13502-4627

TABLE OF CONTENTS

List of Figures

List of Tables

1.0 Introduction and Purpose

This report is intended to help the reader understand what Artificial Neural Networks are, how to use them, and where they are currently being used.

Artificial Neural Networks are being touted as the wave of the future in computing. They are indeed self learning mechanisms which don't require the traditional skills of a programmer. But unfortunately, misconceptions have arisen. Writers have hyped that these neuron-inspired processors can do almost anything. These exaggerations have created disappointments for some potential users who have tried, and failed, to solve their problems with neural networks. These application builders have often come to the conclusion that neural nets are complicated and confusing. Unfortunately, that confusion has come from the industry itself. An avalanche of articles have appeared touting a large assortment of different neural networks, all with unique claims and specific examples. Currently, only a few of these neuron-based structures, paradigms actually, are being used commercially. One particular structure, the feedforward, back-propagation network, is by far and away the most popular. Most of the other neural network structures represent models for "thinking" that are still being evolved in the laboratories. Yet, all of these networks are simply tools and as such the only real demand they make is that they require the network architect to learn how to use them.

This report is intended to help that process by explaining these structures, right down to the rules on how to tweak the "nuts and bolts." Also this report discusses what types of applications are currently utilizing the different structures and how some structures lend themselves to specific solutions.

In reading this report, a reader who wants a general understanding of neural networks should read sections 2, 3, 6, 7 and 8. These sections provide an understanding of neural networks (section 2), their history (section 3), how they are currently being applied (section 6), the tools to apply them plus the probable future of neural processing (section 7), and a summary of what it all means (section 8). A more serious reader is invited to delve into the inner working of neural networks (section 4) and the various ways neural networks can be structured (section 5).

2.0 What are Artificial Neural Networks?

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.

2.1 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 is 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. They do it in a primitive way. But for the software engineer who is trying to solve problems, neural computing was never about replicating human brains. It is about machines and a new way to solve problems.

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 2.2.1 shows the relationship of these four parts.

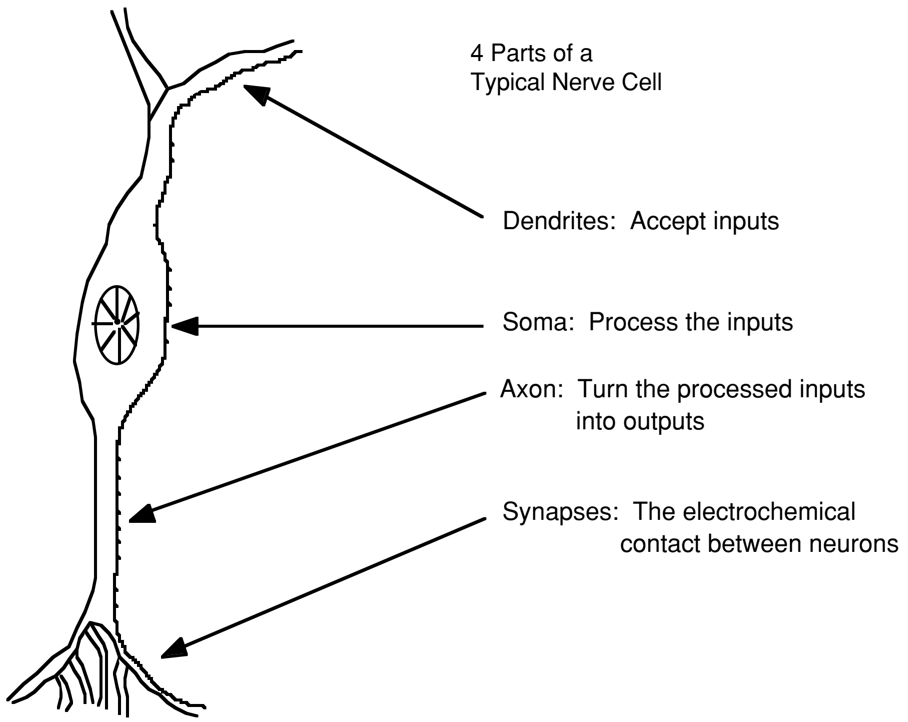
4 Parts of a
Typical Nerve Cell
Dendrites: Accept inputs
Soma: Process the inputs
Axon: Turn the processed inputs into outputs
Synapses: The electrochemical contact between neurons

Figure 2.2.1 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 2.2.2 shows a fundamental representation of an artificial neuron.

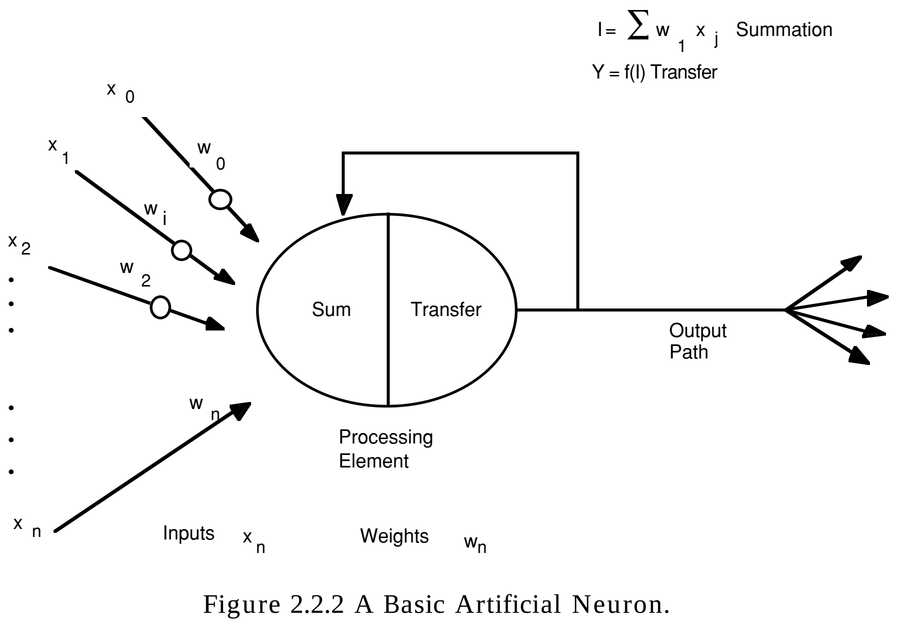

Figure 2.2.2 A Basic Artificial Neuron.

In Figure 2.2.2, 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.

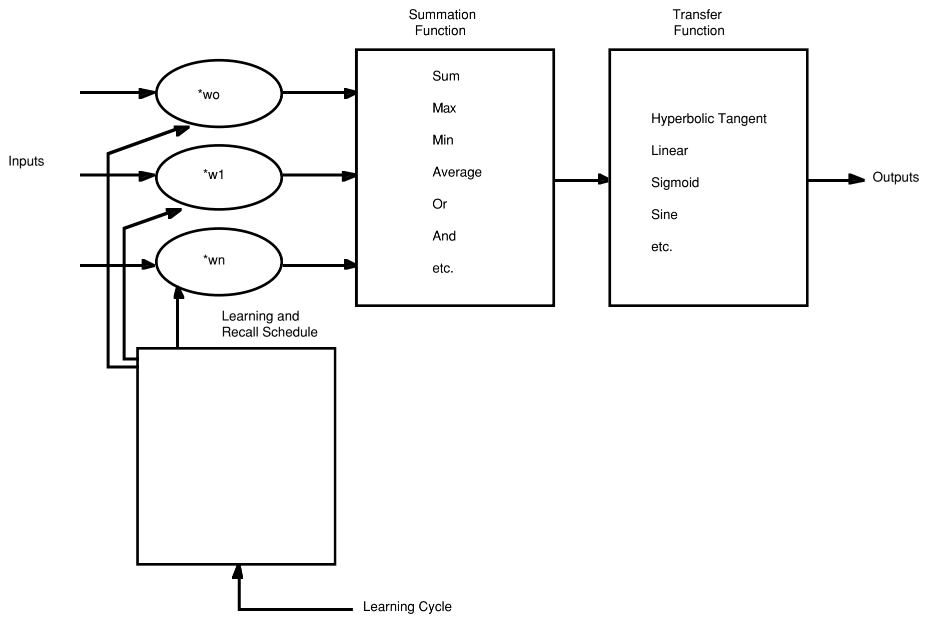

Figure 2.2.3 A Model of a "Processing Element".

In Figure 2.2.3, 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 2.2.4.

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.

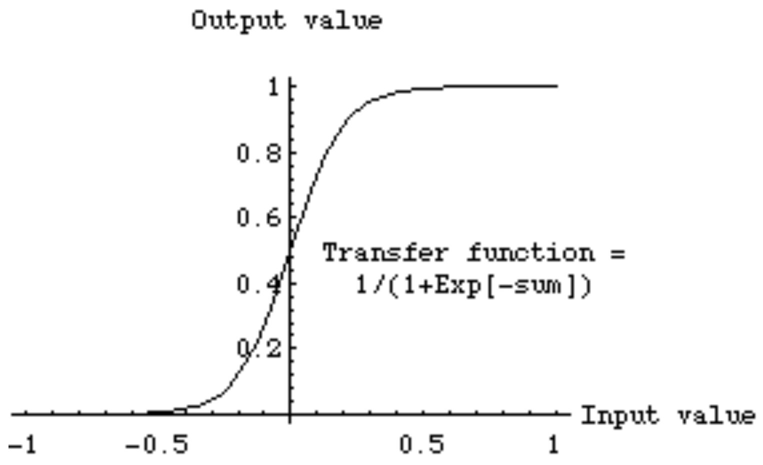

Figure 2.2.4 Sigmoid Transfer Function.

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.

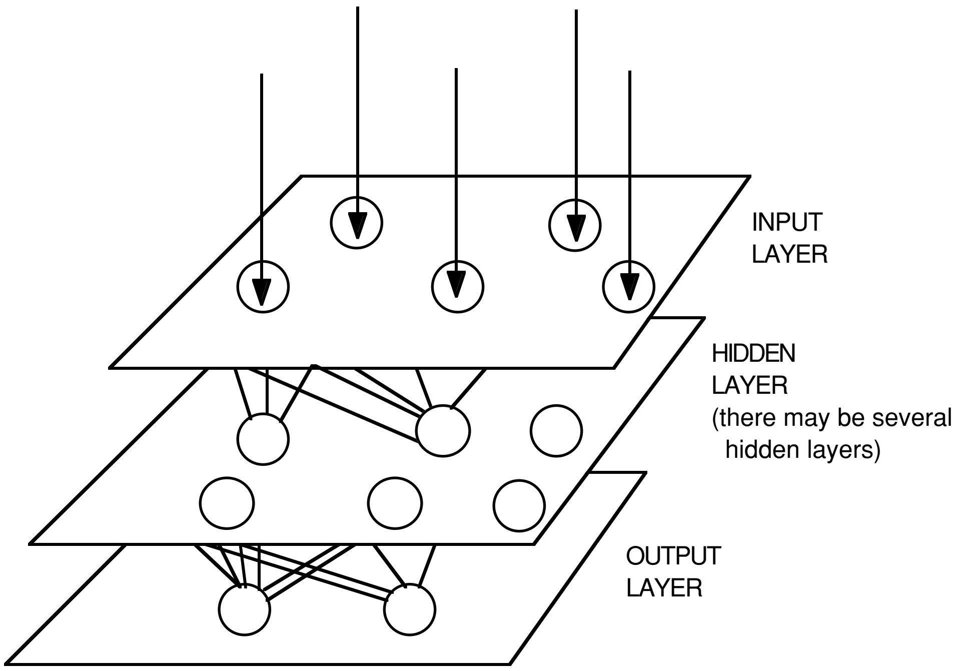

Figure 2.4.1 A Simple Neural Network Diagram.

Basically, all artificial neural networks have a similar structure or topology as shown in Figure 2.4.1. In that structure some of the neurons interfaces 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 receive 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 feedforward path to the output. (Note: in section 5 the drawings are reversed, inputs come into the bottom and outputs come out the top.)

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 2.4.2.

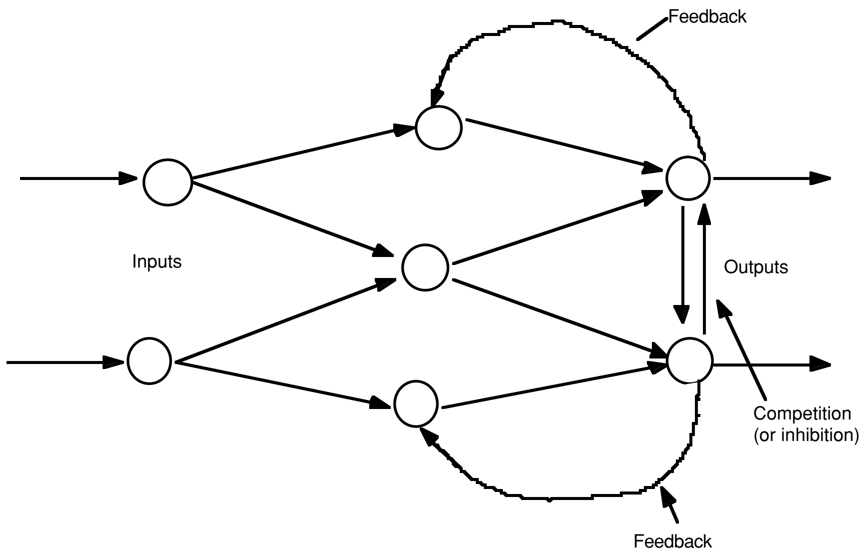

Figure 2.4.2
Simple Network with Feedback and Competition.

The way that the neurons are connected to each other has a significant impact on the operation of the network. In the larger, more professional software development packages the user is allowed to add, delete, and control these connections at will. By "tweaking" parameters these connections can be made to either 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 nonsignificant 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 overtrained. 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 2.6.1 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.

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

Table 2.6.1 Comparison of Computing Approaches.

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 networks is contained in Table 2.6.2.

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
Memory	Separate	The same
Processing Approach	Processes problem one rule at a 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
Use of Neurobiology in design	None	Moderate
Programming	Through a rule based shell; complicated	Self-programming; but network must be properly set up
Ability to be fast	Requires big processors	Requires multiple custom-built chips

Table 2.6.2 Comparisons of Expert Systems and Neural Networks.

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.

Neural systems do exact their own demands. They do require their implementor 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 his 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 History of Neural Networks

The study of the human brain is thousands of years old. With the advent of modern electronics, it was only natural to try to harness this thinking process. The first step toward artificial neural networks came in 1943 when Warren McCulloch, a neurophysiologist, and a young mathematician, Walter Pitts, wrote a paper on how neurons might work. They modeled a simple neural network with electrical circuits.

Reinforcing this concept of neurons and how they work was a book written by Donald Hebb. The *Organization of Behavior* was written in 1949. It pointed out that neural pathways are strengthened each time that they are used.

As computers advanced into their infancy of the 1950s, it became possible to begin to model the rudiments of these theories concerning human thought. Nathanial Rochester from the IBM research laboratories led the first effort to simulate a neural network. That first attempt failed. But later attempts were successful. It was during this time that traditional computing began to flower and, as it did, the emphasis in computing left the neural research in the background.

Yet, throughout this time, advocates of "thinking machines" continued to argue their cases. In 1956 the Dartmouth Summer Research Project on Artificial Intelligence provided a boost to both artificial intelligence and neural networks. One of the outcomes of this process was to stimulate research in both the intelligent side, AI, as it is known throughout the industry, and in the much lower level neural processing part of the brain.

In the years following the Dartmouth Project, John von Neumann suggested imitating simple neuron functions by using telegraph relays or vacuum tubes. Also, Frank Rosenblatt, a neuro-biologist of Cornell, began work on the Perceptron. He was intrigued with the operation of the eye of a fly. Much of the processing which tells a fly to flee is done in its eye. The Perceptron, which resulted from this research, was built in hardware and is the oldest neural network still in use today. A single-layer perceptron was found to be useful in classifying a continuous-valued set of inputs into one of two classes. The perceptron computes a weighted sum of the inputs, subtracts a threshold, and passes one of two possible values out as the result. Unfortunately, the perceptron is limited and was proven as such during the "disillusioned years" in Marvin Minsky and Seymour Papert's 1969 book *Perceptrons*.

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.

Unfortunately, these earlier successes caused people to exaggerate the potential of neural networks, particularly in light of the limitation in the electronics then available. This excessive hype, which flowed out of the academic and technical worlds, infected the general literature of the time. Disappointment set in as promises were unfilled. Also, a fear set in as writers began to ponder what effect "thinking machines" would have on man. Asimov's series on robots revealed the effects on man's morals and values when machines where capable of doing all of mankind's work. Other writers created more sinister computers, such as HAL from the movie 2001.

These fears, combined with unfulfilled, outrageous claims, caused respected voices to critique the neural network research. The result was to halt much of the funding. This period of stunted growth lasted through 1981.

In 1982 several events caused a renewed interest. John Hopfield of Caltech presented a paper to the national Academy of Sciences. Hopfield's approach was not to simply model brains but to create useful devices. With clarity and mathematical analysis, he showed how such networks could work and what they could do. Yet, Hopfield's biggest asset was his charisma. He was articulate, likeable, and a champion of a dormant technology.

At the same time, another event occurred. A conference was held in Kyoto, Japan. This conference was the US-Japan Joint Conference on Cooperative/Competitive Neural Networks. Japan subsequently announced their Fifth Generation effort. US periodicals picked up that story, generating a worry that the US could be left behind. Soon funding was flowing once again.

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. Currently most neural network development is simply proving that the principal works. This research is developing neural networks that, due to processing limitations, take weeks to learn. To take these prototypes out of the lab and put them into use requires specialized chips. Companies are working on three types of neuro chips - digital, analog, and optical. Some companies are working on creating a "silicon compiler" to generate a neural network Application Specific Integrated Circuit (ASIC). These ASICs and neuron-like digital chips appear to be the wave of the near future. Ultimately, optical chips look very promising. Yet, it may be years before optical chips see the light of day in commercial applications.

4.0 Detailed Description of Neural Network Components and How They Work

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.

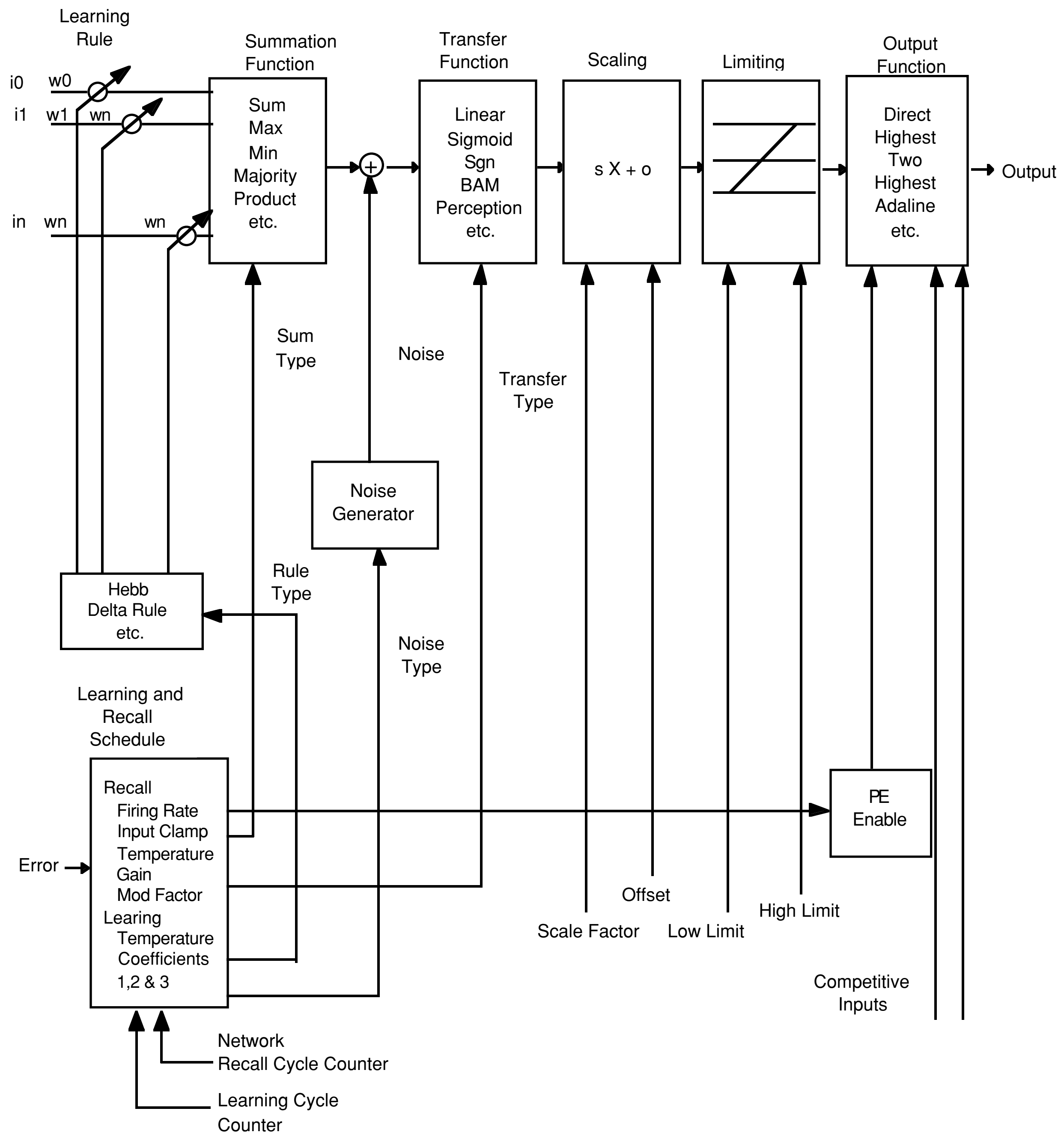

Figure 4.0.1 Processing Element.

Neural computing is about machines, not brains. It is the process of trying to build processing systems that draw upon the highly successful designs naturally occuring in biology. This linkage with biology is the reason that there is a common architectural thread throughout today's artificial neural networks. Figure 4.0.1 shows a model of an artificial neuron, or processing element, which embodies a wide variety of network architectures.

This figure is adapted from NeuralWare's simulation model used in NeuralWorks Profession II/Plus. NeuralWare sells a artificial neural network design and development software package. Their processing element model shows that networks designed for prediction can be very similar to networks designed for classification or any other network category. Prediction, classification and other network categories will be discussed later. The point here is that all artificial neural processing elements have common components.

4.1 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 muliplying 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. See Figure 4.1.1 for sample transfer functions.

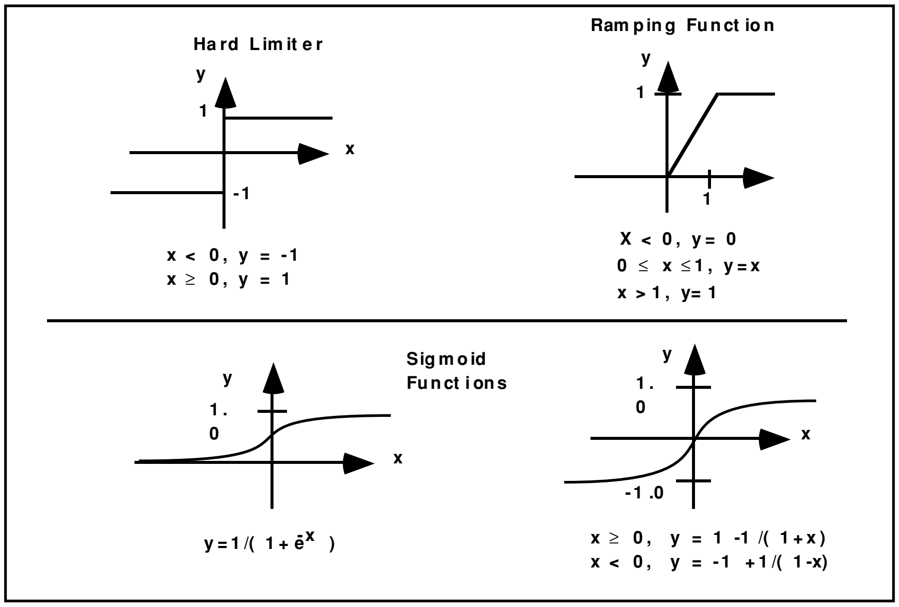

Figure 4.1.1 Sample Transfer Functions.

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.

NASA just announced a network topology which uses what it calls a temperature coefficient in a new feed-forward, 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.

4.2 Teaching an Artificial Neural Network

4.2.1 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 an 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 an 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.

4.2.2 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.

4.2.3 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.

4.2.4 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.

5.0 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, of 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 network architectures and their best matches to specific applications.

Basically, most applications of neural networks fall into the following five categories:

- prediction
- classification
- data association
- data conceptualization
- data filtering

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-propagation	Use input values to predict some output (e.g. pick the best stocks in the stock market, predict the weather, identify people with cancer risks)
Classification	- Learning vector quantization - Counter-propagation -Probabalistic neural network	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 -Spatio-temporal pattern recognition	Like classification but it also recognizes data that contains errors (e.g. not only identify the characters that were scanned but also identify when the scanner wasn'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 data base 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 5.0.1 Network Selector Table.

Table 5.0.1 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.

5.1 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.

5.1.1 Feedforward, 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 learn model for complex, multi-layered networks. This network is used more than all others 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 maximum 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 5.0.1. (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.

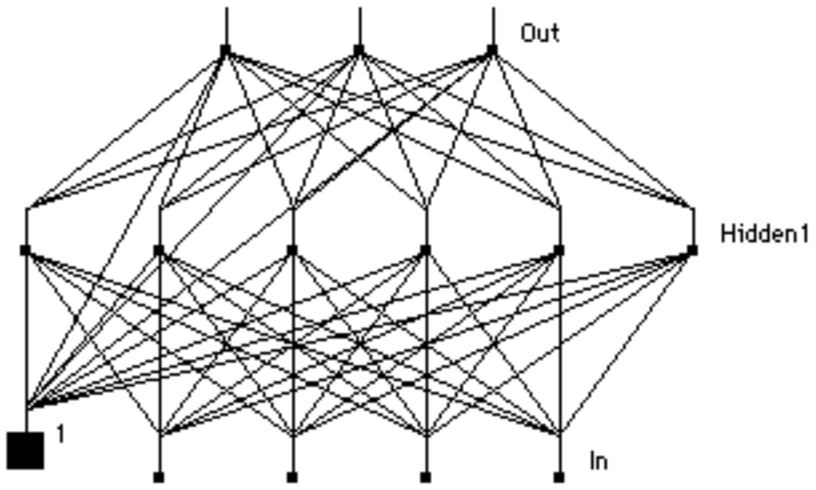

Figure 5.0.1 An Example Feedforward Back-propagation Network.

The number of layers and the number of processing elements per layer are important decisions. These parameters to a feedforward, back-propagation topology are also the most ethereal. They are the "art" 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 to their problems.

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 layer(s). 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 networks. Different error functions, transfer functions, and even the modifying method of the derivative of the transfer function can be used. The concept of "momentum error" 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 cancelled 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 "epoch." 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 system 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.

5.1.2 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'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 learning 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 "heuristic" 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 intelligent 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 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 preformed. 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 "curvature of the error surface" 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.

5.1.3 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.

5.1.4 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.

5.1.5 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.

5.1.6 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

5.2 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.

5.2.1 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 5.2.1. 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.

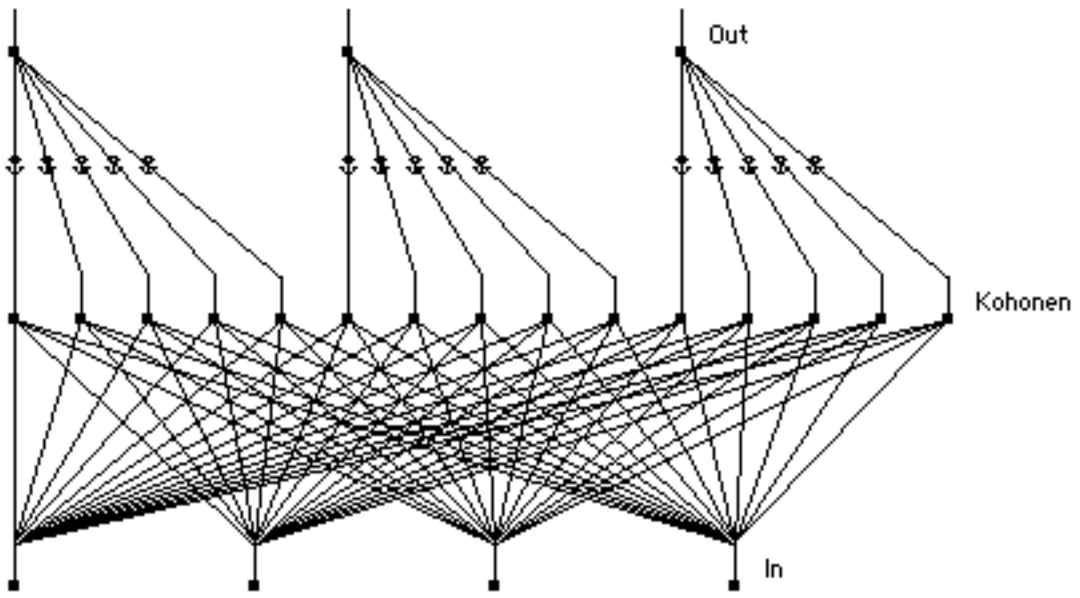

Figure 5.2.1. An Example Learning Vector Quantization Network.

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 mechanisms 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 effects 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.

5.2.2 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 5.2.2. 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.

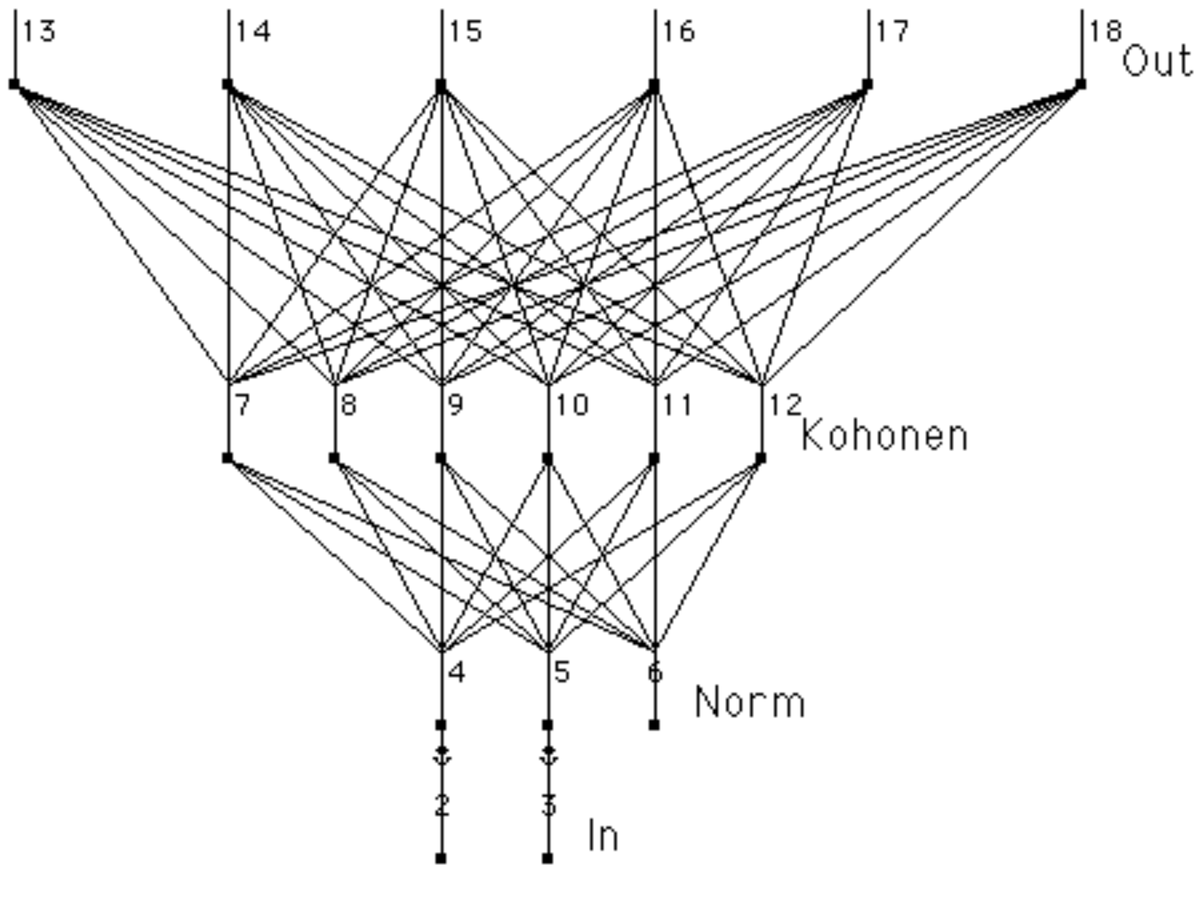

Figure 5.2.2. An Example Counter-propagation Network.

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.

5.2.3 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 5.2.3. 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.

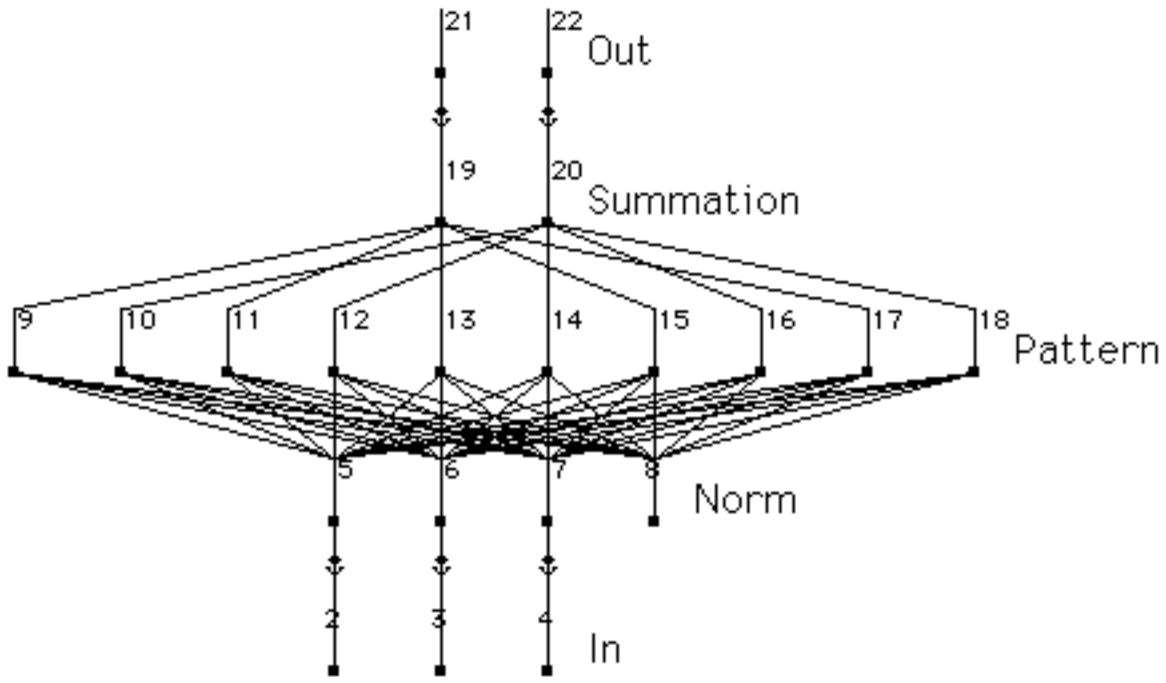

Figure 5.2.3. A Probabilistic Neural Network Example.

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.

5.3 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.

5.3.1 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 5.3.1 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.

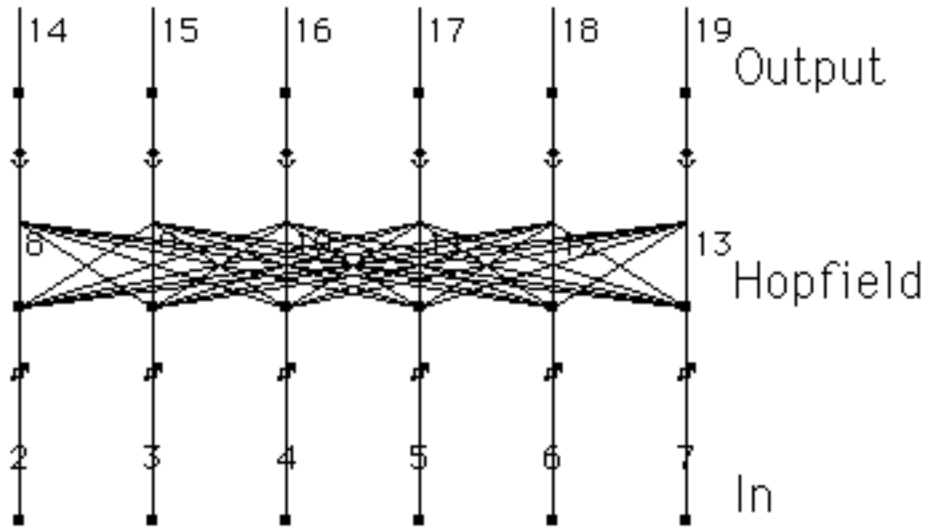

Figure 5.3.1. A Hopfield Network Example.

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.

5.3.2 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.

5.3.3 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 5.3.2. 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.

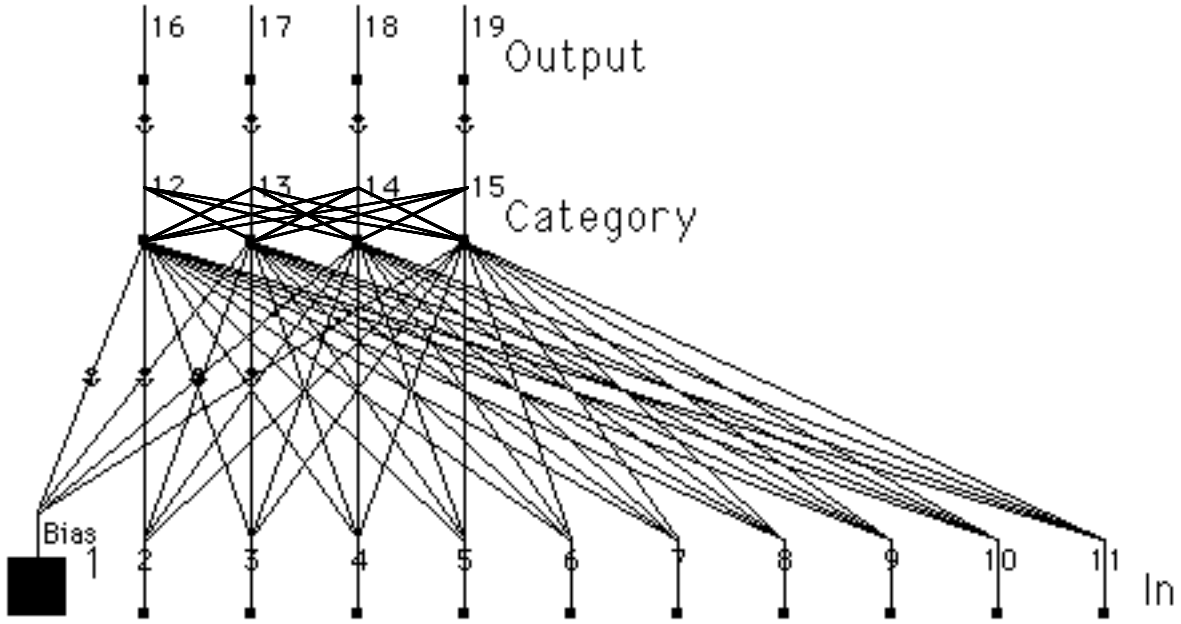

Figure 5.3.2. A Hamming Network Example.

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.

5.3.4 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.

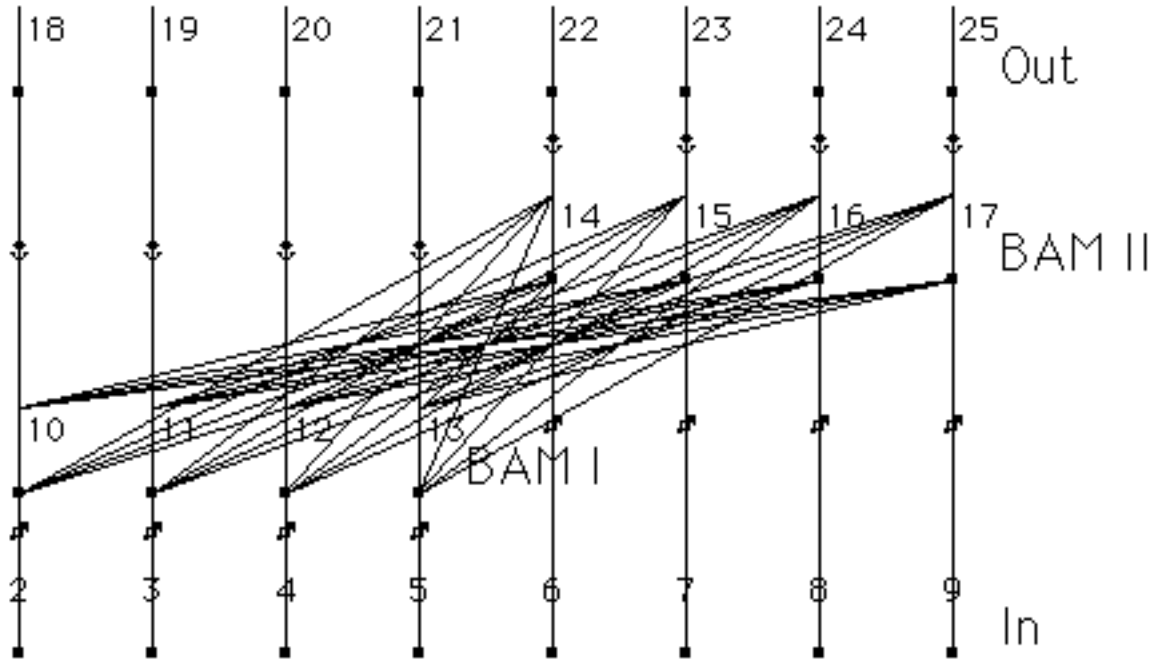

Figure 5.3.4. Bi-directional Associative Memory Example.

A diagram of an example bi-directional associative memory is shown in Figure 5.3.4. 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 examples 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.

5.3.5 Spatio-Temporal Pattern Recognition (Avalanche).

This network as shown in Figure 5.3.5 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.

There 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 a 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.

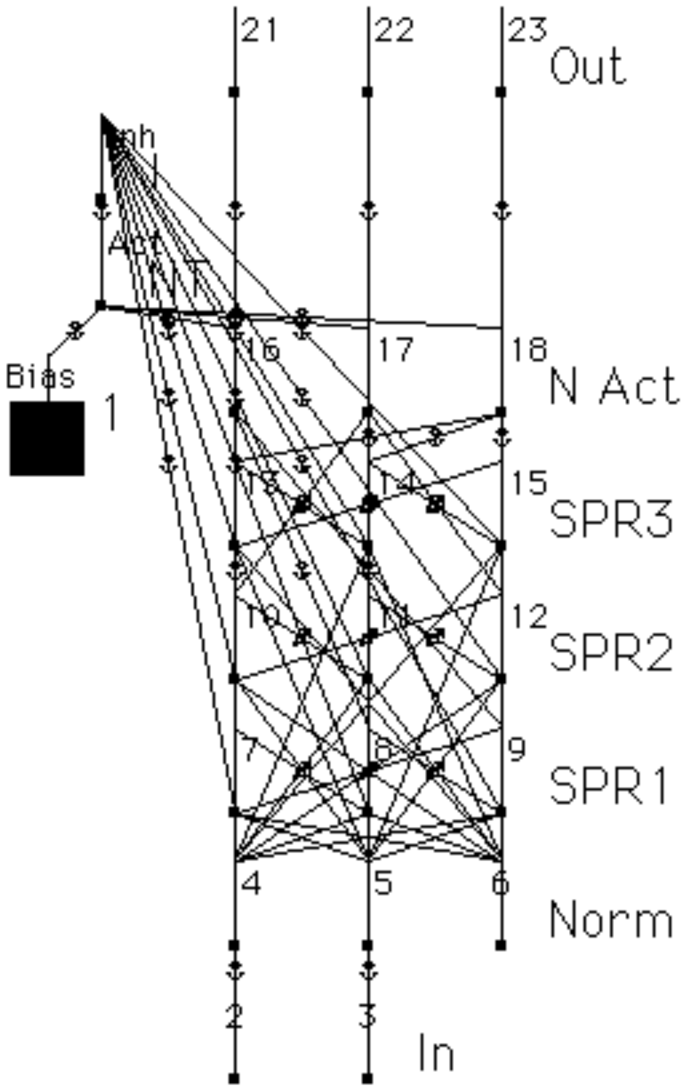

Figure 5.3.5. A Spatio-temporal Pattern Network Example.

5.4 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.

5.4.1 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.

5.4.2 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, compacts sparce 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 5.4.2. 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.

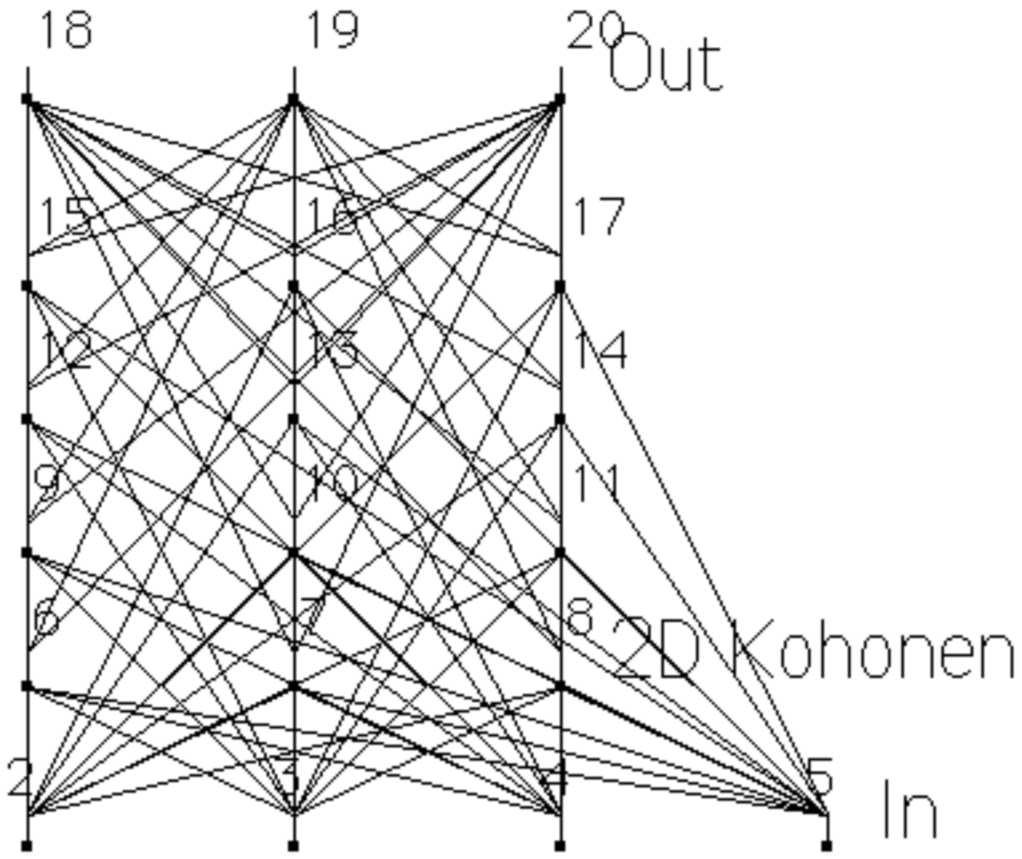

Figure 5.4.2. An Example Self-organizing Map Network.

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.

5.5 <u>Networks for Data Filtering</u>

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.

5.5.1 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, 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.

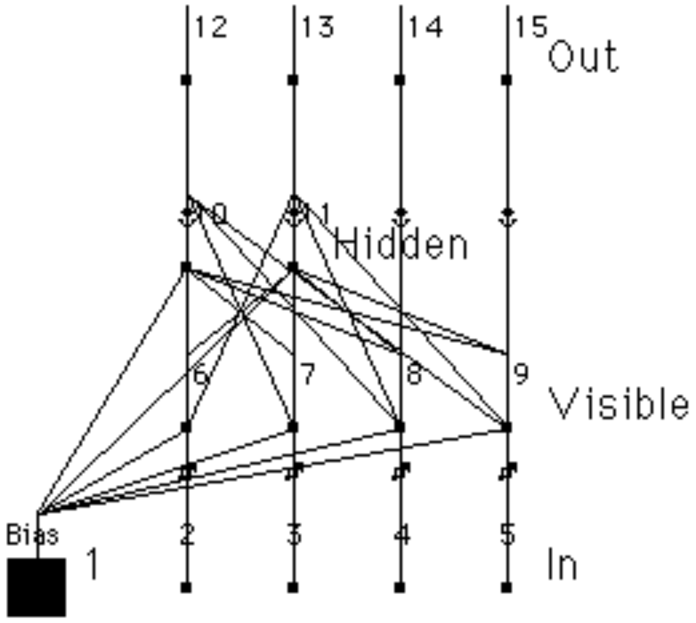

Figure 5.5.1. An Example Recirculation Network.

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.

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.

6.0 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.

6.1 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.

6.2 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.

6.3 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.

6.4 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.

Another field related to defense sensor processing is the recognition of patterns within the sensor data of the medical industry. A neural network is now being used in the scanning of PAP smears. This network is trying to do a better job at reading the smears than can the average lab technician. Missed diagnoses is a too common problem throughout this industry. In many cases, a professional must perceive patterns from noise, such as identifying a fracture from an X-ray or cancer from a X-ray "shadow." Neural networks promise, particularly when faster hardware becomes available, help in many areas of the medical profession where data is hard to read.

6.5 Signal Processing

Neural networks' promise for signal processing has resulted in a number of experiments in various university labs. Neural networks have proven capable of filtering out noise. Widrow's MADALINE was the first network applied to a real-world problem. It eliminates noise from phone lines.

Another application is a system that can detect engine misfire simply from the noise. This system, developed by Odin Corp, works on engines up to 10,000 RPMS. The Odin system satisfies the California Air Resources Board's mandate that by 1994 new automobiles will have to detect misfire in real time. Misfires are suspected of being a leading cause of pollution. The

Odin solution requires 3 kbytes of software running on a Motorola 68030 microprocessor.

6.6 Financial

Neural networks are making big inroads into the financial worlds. Banking, credit card companies, and lending institutions deal with decisions that are not clear cut. They involve learning and statistical trends.

The loan approval process involves filling out forms which hopefully can enable a loan officer to make a decision. The data from these forms is now being used by neural networks which have been trained on the data from past decisions. Indeed, to meet government requirements as to why applications are being denied, these packages are providing information on what input, or combination of inputs, weighed heaviest on the decision.

Credit card companies are also using similar back-propagation networks to aid in establishing credit risks and credit limits.

In the world of direct marketing, neural networks are being applied to data bases so that these phone peddlers can achieve higher ordering rates from those annoying calls that most of us receive at dinner time. (A probably more lucrative business opportunity awaits the person who can devise a system which will tailor all of the data bases in the world so that certain phone numbers are never selected.)

Neural networks are being used in all of the financial markets - stock, bonds, international currency, and commodities. Some users are cackling that these systems just make them "see green," money that is. Indeed, neural networks are reported to be highly successful in the Japanese financial markets. Daiichi Kangyo Bank has reported that for government bond transactions, neural networks have boosted their hit rate from 60% to 75%. Daiwa research Institute has reported a neural net system which has scored 20% better than the Nikkei average. Daiwa Securities' stock prediction system has boosted the companies hit rate from 70% to 80%.

6.7 Servo Control

Controlling complicated systems is one of the more promising areas of neural networks. Most conventional control systems model the operation of all the system's processes with one set of formulas. To customize a system for a specific process, those formulas must be manually tuned. It is an intensive process which involves the tweaking of parameters until a combination is found that produces the desired results. Neural networks offer two advantages. First, the statistical model of neural networks is more complex that a simple set of formulas, enabling it to handle a wider variety of operating conditions without having to be retuned. Second, because neural networks learn on their own, they don't require control system's experts, just simply enough historical data so that they can adequately train themselves.

Within the oil industry a neural network has been applied to the refinery process. The network controls the flow of materials and is touted to do that in a more vigilant fashion than distractible humans.

NASA is working on a system to control the shuttle during in-flight maneuvers. This system is known as Martingale's Parametric Avalanche (a spatio-temporal pattern recognition network as explained in section 5.3.5). Another prototype application is known as ALVINN, for Autonomous Land Vehicle in a Neural Network. This project has mounted a camera and a laser range finder on the roof of a vehicle which is being taught to stay in the middle of a winding road.

British Columbia Hydroelectric funded a prototype network to control operations of a power-distribution substation that was so successful at optimizing four large synchronous condensors that it refused to let its supplier, Neural Systems, take it out.

6.8 How to Determine if an Application is a Neural Network Candidate

As seen by the sections above, neural networks are being successfully applied in a number of areas. Each of these applications can be grouped into two broad categories. These categories offer a test for anyone who is considering using neural networks. Basically, a potential application should be examined for the following two criteria:

- Can a neural network replace existing technologies in an area where small improvements in performance can result in a major economic impact? Examples of applications which meet this criteria are:

    - loan approvals

    - credit card approvals

    - financial market predictions

    - potential customer analysis for the creation of mailing lists

- Can a neural network be used in an area where current technologies have proven inadequate to making a system viable? Examples of applications which meet this criteria are:

    - speech recognition

    - text recognition

- target analysis

(Another example where other technologies failed was in explosive detection at airports. Previous systems could not achieve the FAA mandated level of performance, but by adding a neural network the system not only exceeded the performance, it allowed the replacement of a $200,000 component.)

The most successful applications have been focused on a single problem in a high value, high volume, or a strategically important application.

The easiest implementation of neural networks occur in solutions where they can be made to be "plug compatible" with existing systems. To simply replace an existing element of a system with a neural network eases an installation. It also increases the likelihood of success. These "plug compatible" solutions might be at the front end of many systems where neural networks can recognize patterns and classify data.

7.0 Emerging Technologies

If the 21st Century is to be the age of intelligent machines, then artificial neural networks will become an integral part of life.

In order that software engineers can lead us to this "promised life" they must begin by utilizing the emerging technology of Neural Networks. To do that they must optimize their time by using already implemented hardware and commercial software packages while anticipating what is still to come. To accomplish this understanding, this section is broken into two pieces - what currently exists and what implementors think the next developments will be.

7.1 What Currently Exists

Currently, a number of vendors exist within the marketplace. These vendors are each seeking a share of the neural network business. Some of them do that by hitching their wagon to other packages within the industry. Neural network products exist which are simply add-ons to the popular data bases and spreadsheets. Other products are geared for particular operating systems on particular machines. There are vendors of neural network development tools for most machines. The most popular tools work on either Apple's Macintosh or the IBM PC standard.

Some of these packages are geared toward particular applications such as image processing. Others are general but lack good data routing capabilities. Each of these companies are identifying their weaknesses and are working on them. It is an exciting time for them, with both the rewards and risks high.

In choosing a development tool a software engineer needs to beware of this emerging field. Most products are not evolved into the user friendly routines that draw raves. This is a young field. Its very volatility has created a confusing set of offerings, and features within offerings, which will ultimately be relegated to the trash.

7.1.1 Development Systems.

Good development systems allow an user to prototype a network, train it, tweak it, and use it. These systems run on the standard range of computers. These packages usually don't run on specialized hardware, although some vendors have packaged fast RISC processors into special neural processing boards. Usually, these packages are simply tools which create networks that prove concepts but may be way too slow to run. One of the more complete lists of these vendors is published in the November 1991 issue of *Personal Engineering & Instrumentation News.*

7.1.2 Hardware Accelerators.

The key to the continued evolution of neural networking lies in the hardware. Traditional hardware does not enable the massive parallelism that is required by neural networks. There are several approaches that are being worked on. One is to develop a processor which is specifically tailored to performing the tasks of individual artificial neurons. Another approach is to package fast processors, primarily RISCs, onto a hardware accelerator. These processors can be packed many to a board to facilitate the parallel nature of neural networks. Other accelerator boards simply provide more horsepower for sequential processing.

Accelerator board products are being developed both independently and by the makers of neural network development systems. Each have specific characteristics that lend themselves to particular resolutions.

7.1.3 Dedicated Neural Processors.

Dedicated neural processors are processors with specific capabilities that enable their use in neural networks. Several of the large chip manufacturers have developed neural processors. Some of these processors were created specifically for the development system vendors. Some of these chips package a number of simplistic neurons onto one chip. Others incorporate proprietary concepts, such as creating a specific type of fuzzy neuron. These chips come in many broad technologies - analog, digital, hybrid, and optical. There is no clear winner to date.

7.2 What the Next Developments Will Be

The vendors within the industry predict that migration from tools to applications will continue. In particular, the trend is to move toward hybrid systems. These systems will encompass other types of processes, such as fuzzy logic, expert systems, and kinetic algorithms. Indeed, several manufactures are working on "fuzzy neurons."

The greatest interest is on merging fuzzy logic with neural networks. Fuzzy logic incorporates the inexactness of life into mathematics. In life most pieces of data do not exactly fit into certain categories. For instance, a person is not just short or tall. He can be kinda short, pretty tall, a little above average, or very tall. Fuzzy logic takes these real-world variations into account. In potential application of neural networks, in systems which solve real problems, this fuzziness is a large part of the problem. In automating a car, to stop is not to slam on the brakes, to speed up is not to "burn rubber." To help neural networks accomodate this fuzziness of life, some researchers are developing fuzzy neurons. These neurons do not simply give yes/no answers. They provide a more fuzzy answer.

Systems built with fuzzy neurons may be initialized to what an expert thinks are the rules and the weights for a given application. This merging of expert systems and fuzzy logic with neural networks utilizes the strength of all three disciplines to provide a better system than either can provide themselves. Expert systems have the problem that most experts don't exactly understand all of the nuances of an application and, therefore, are unable to clearly state rules which define the entire problem to someone else. But the neural network doesn't care that the rules are not exact, for neural networks can then learn, and then correct, the expert's rules. It can add nodes for concepts that the expert might not understand. It can tailor the fuzzy logic which defines states like tall, short, fast, or slow. It can tweak itself until it can meet the user identified state of being a workable tool. In short, hybrid systems are the future.

8.0 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.

9.0 References

[Aarts, 89] Aarts, Emile, and Korst, Jan, *Simulated Annealing and Boltzmann Machines, A Stochastic Approach to Combinatorial Optimization and Neural Computing*, Wiley, Tiptree Essex GB, 1989.

[Abu-Mostafa, 85] Abu-Mostafa, Y.S., and St. Jacques, J.M., "Information Capacity of the Hopfield Model", *IEEE Transactions on Information Theory*, Volume IT-31, Number 4, July 1989.

[Ackley, 85] Ackley, D.H., Hinton, G.E., and Sejnowski, T.J., "A Learning Algorithm for Boltzmann Machines", *Cognitive Science*, Volume 9, 1985.

[Ahmad,90] Ahmad, Z., "Improving the Solution of the Inverse Kinematic Problem in Robotics Using Neural Networks", *Journal of Neural Network Computing*, Vol. 1 Num. 4, Spring 1990.

[Anderson, 72] Anderson, James A., "A Simple Neural Network Generating an Interactive Memory", *Mathematical Biosciences*, Volume 14, 1972.

[Anderson, 77] Anderson, James A., Silverstein, Jack W., Ritzx, Stephen A., and Jones, Randall S., "Distinctive Features, Categorical Perception, and Probability Learning: Some Applications of a Neural Model", *Psychological Review*, Volume 84, Number 5, September 1977.

[Anderson, 83] Anderson, James A., "Cognitive and Psychological Computation with Neural Models", *IEEE Transactions on Systems, Man, and Cybernetics*, Volume SMC-13, Number 5, September 9183.

[Anderson, 86a] Anderson, James A., and Murphy, Gregory L., "Psychological Concepts in a Parallel System", *Physical 22D*, 1986.

[Anderson, 86b] Anderson, James A., "Cognitive Capabilities of a Parallel System", Springer-Verlag, 1986.

[Anderson,88] Anderson, J.A. and Rosenfeld,E., eds., "Neurocomputing: Foundations of Research, MIT Press, Boston, MA 1988, page 125.

[Antognetti,91] Antognetti, P. and Milutinovic, V., *Neural Networks: Concepts, Applications, and Implementations* (Eds.) Volumes I-IV, Prentice Hall, Englewood Cliffs, NJ., 1991.

[Baba, 77] Baba, Norio, Shoman, T., and Sawaragi, Y., "A Modified Convergence Theorem for a Random Optimization Method", *Information Sciences*, Volume 13, 1977.

[Baba, 89] Baba, Norio, "A New Approach for Finding the Global Minimum of Error Function of Neural Networks", *Neural Networks,* Volume 2, 1989.

[Barto, 81] Barto, A.G., and Sutton, R.S., "Goal Seeding Components for Adaptive Intelligence: An Initial Assessment", *Air Force Wright Aeronautical Laboratories/Avionics Laboratory Technical Report AFWAL-TR-18-1070,* Ohio, Wright-Patterson AFB, 1981.

[Batchelor, 74] Batchelor, B.G., *Practical Approach To Pattern Recognition*, Plenum Press, New York, 1974.

[Bernstein, 81] Bernstein, J., "Profiles: Marvin Minsky", *The New Yorker*, December 1981.

[Brown, 87] Brown, Robert J., "An Artificial Neural Network Experiment", *Dr. Dobbs Journal,* April 1987.

[Burr, 88] Burr, D.J., "An Improved Elastic Net Method for the Traveling Salesman Problem", *Proc. of the IEEE First International Conference on Neural Networks*, Volume 1, June 1988.

(California,88) California Scientific Software, "Brainmaker User Guide and Reference Manual", California Scientific Software, 1988.

[Carpenter, 87a] Carpenter, Gail A., and Grossberg, Stephen, "A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine", *Computer Vision, Graphics and Image Processing 37*, 1987.

[Carpenter, 87b] Carpenter, Gail A., and Grossberg, Stephen, "ART 2: Self-Organization of Stable Category Recognition Codes for Analog Input Patterns", *Applied Optics,* 1987.

[Carpenter, 87c] Carpenter, Gail, A., and Grossberg, Stephen, "ART 3: Hierarchical Search Using Chemical Transmitters in Self-Organizing Pattern Recognition Architectures", *Neural Networks,* Volume 3, 1987.

[Carpenter, 88] Carpenter, Gail A., and Grossberg, Stephen, "The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network", *Computer*, March, 1988.

[Carpenter, 91] Carpenter, Gail, A., Grossberg, Stephen, and Rosen, D.B., "ART-2A: An Adaptive Resonance Algorithm for Rapid Category Learning And Recognition", *Neural Networks,* Volume 4, 1991.

[Caudill, 90] Caudill, Maureen, and Butler, Charles, *Naturally Intelligent Systems*, The MIT Press, ISBN 0-262-03156-6, 1990.
[Cohen, 83] Cohen, Michael A., and Grossberg, Stephen, "Absolute Stability of Global Pattern Formation and Parallel Memory Storage by Competitive Neural Networks", *IEEE Transactions on Systems, Man and Cybernetics*, Volume SMC-13, 1983.

[Cottrell, 87] Cottrell, G.W., Munro, P., and Zipser D., "Learning Internal Representations form Gray-Scale Images: An Example of Extensional Programming", *In Proc. 9th Annual Conference of the Cognitive Science Society*, 1987.

[DeSieno, 88] DeSieno, D., "Adding a Conscience to Competitive Learning", *Proc. of the Second Annual IEEE International Conference on Neural Networks*, Volume 1, 1988.

[Durbin, 87] Durbin, R., and Willshaw, D., "An Analog Approach to the Traveling Salesman Problem Using an Elastic Net Method", *Nature*, Volume 326, April 1987.

[Eberhart, 90] Eberhart, Russell C., and Dobbins, Roy W., *Neural Network PC Tools: A Practical Guide*, Academic Press, ISBN 0-12-228640-5, 1990.

[Fahlmann, 88] Fahlmann, Scott E. "An Empirical Study of Learning Speed in Back-Propagation Networks", *CMU Technical Report*, CMU-CS-88-162, June 1988.

[Fukushima, 75] Fukushima, Kuniko, "Cognitron: A Self-Organizing Multilayered Neural Network", *Biological Cybernetics*, Volume 20, 1975.

[Fukushima, 80] Fukushima, Kuniko, "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position", *Biological Cybernetics*, Volume 36, 1980.

[Fukushima,83] Fukushima, Kuniko, and Takayuki, I., "Neocognition: A Neural Network Model for a Mechanism of Visual Pattern Recognition", *IEEE Transactions on Systems, Man, and Cybernetics* 13(5), September/October 1983, pp. 826-34.

[Fukushima, 88] Fukushima, Kuniko, "Neocognitron: A Hierarchical Neural Network Capable of Visual Pattern Recognition", *Neural Networks*, Volume 1, 1988.

[Fukushima, 89] Fuksushima, Kuniko, "Analysis of the Process of Visual Pattern Recognition by the Neocognitron", *Neural Networks*, Volume 2, 1989.

[Galland, 88] Galland, C., "Biologically Plausible Learning Procedures in Connectionist Networks", *M.S. Thesis*, Department of Physics, University of Toronto, 1988.

[Gaudiano, 91] Gaudiano, P., and Grossberg, Stephen, "Vector Associative Maps: Unsupervised Real-Time Error-Based Learning and Control of Movement Trajectories", *Neural Networks*, Volume 4, 1991.

[Glover, 88] Glover, David E., "A Hybrid Optical Fourier/Electronic Neurocomputer Machine Vision Inspection System", *Proc. Vision '88 Conference*, SME/MVA, 1988.

[Glover, 89] Glover, David E., "Optical Processing and Neurocomputing in an Automated Inspection System", *Journal of Neural Network Computing*, Fall 1989.

[Golden, 86] Golden, Richard M., "Brain-State-in-a-Box Neural Model is a Gradient Descent Algorithm", *Journal of Mathematical Psychology*, 1986.

[Gorman, 88] Gorman, R.P., and Sejnowski, T.J., "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets", *Neural Networks*, Volume 1, 1988.

[Grossberg, 69a] Grossberg, Stephen, "Embedding Fields: A Theory of Learning with Physiological Implications", *Journal of Mathematical Psychology*, Volume 6, 1969.

[Grossberg, 69b] Grossberg, Stephen, "Some Networks That Can Learn, Remember, and Reproduce any Number of Complicated Space-Time Patterns, I", *Journal of Mathematics and Mechanics*, Volume 19, 1969.

[Grossberg, 70] Grossberg, Stephen, "Some Networks That Can Learn, Remember, and Reproduce any Number of Complicated Space-Time Patterns, II", *Studies in Applied Mathematics*, Volume 49, 1970.

[Grossberg, 71] Grossberg, Stephen, "Embedding Fields: Underlying Philosophy, Mathematics, and Applications to Psychology, Physiology, and Anatomy", *Journal of Cybernetics*, Volume 1, 1971.

[Grossberg, 76] Grossberg, Stephen, "Adaptive Pattern Classification and Universal Recoding: I. Parallel Development and coding of Neural Feature Detectors", *Biological Cybernetics*, Volume 23, 1976.

[Grossberg, 85] Grossberg, Stephen, and Mingolla, E., "Neural Dynamics of Perceptual Grouping: Textures, Boundaries, and Emergent Segmentations", *Perception and Psychophysics*, Volume 38, 1985.
[Grossberg, 89] Grossberg, Stephen, and Rudd, M.E., "A Neural Network for Visual Motion Perception: Group and Element Apparent Motion", *Neural Networks*, Volume 2, 1989.

[Hebb, 49] Hebb, D. O., *The Organization of Behavior*, Wile, New York, New York, 1949.

[Hecht-Nielsen, 86] Hecht-Nielsen, Robert, "Nearest Matched Filter Classification of Spatio-temporal Patterns", special report published by Hecht-Nielsen Neuro-Computer Corporation, San Diego, California, June 1986.

[Hecht-Nielsen, 87] Hecht-Nielsen, Robert, "Counter-Propagation Networks", *IEEE First International Conference on Neural Networks,* Volume II, 1987.

[Hecht-Nielsen,88] Hect-Nielsen, Robert, "Neurocomputing: Picking the Human Brain", *IEEE Spectrum* 25(3), March 1988, pp. 36-41.

[Hecht-Nielsen, 90] Hecht-Nielsen, Robert, *Neurocomputing*, Addison-Wesley, ISBN 0-201-09255-3, 1990.

[Hegde, 88] Hegde, S.V., Sweet, J.L., and Levy, W.B., "Determination of Parameters in a Hopfield/Tank Computational Network", *Proc. of the IEEE First International Conference on Neural Networks*, Volume 2, June 1987.

[Hinton, 87] Hinton, Geoffrey E., and Sejnowski, Terrence J., "Neural Network Architectures for AI", *Tutorial Number MP2, National Conference on Artificial Intelligence (AAAI-87)*, July 1987.

[Hinton, 88] Hinton, G.E., and McClelland, J.L., "Learning Representations by Recirculation", *Proc. of the IEEE Conference on Neural Information Processing Systems*, November 1988.

[Hopfield, 82] Hopfield, John J., "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", *Proceedings of the National Academy of Sciences,* Volume 79, April 1982.

[Hopfield, 83] Hopfield, John J., Feinstein, D.I., and Palmer, R.G., "Unlearning has a Stabilizing Effect in Collective Memories", *Nature*, Volume 304, July 1983.

[Hopfield, 84] Hopfield, John J., and Tank, David W., "Neural Computation of Decisions in Optimization Problems", *Biological Cybernetics,* Volume 52, 1985.

[Hopfield, 86a] Hopfield, John J., "Physics, Biological Computation and Complimentarity", *The Lessons of Quantum Theory*, Elsevier Science Publishers, B.B., 1986.

[Hopfield, 86b] Hopfield, John J., and Tank, David W., "Collective Computation with Continuous Variables", *Disordered Systems and Biological Organization*, Springer-Verlag, 1986.

[Isik, 91] Isik, C. and Uhrig, R. E., "Neural Networks and Power Utility Applications", *EPRI Knowledge Based Technology Applications Seminar*, September 1991.

[Jacobs, 88] Jacobs, R.A., "Increased Rates of Convergence Through Learning Rate Adaptation", *Neural Networks*, Volume 1, 1988.

[Johnson,89] Johnson, R. Colin, "Neural Nose to Sniff Out Explosives at JFK Airport", *Electronic Engineering Times*, May 1, 1989.

[Johnson,91a] Johnson, R. Colin, "Moto Readies Odin Neural Net", *Electrical Engineering Times*, November 4, 1991.

[Johnson,91b] Johnson, R. Colin, "Darpa Continues Neural Funding", *Electrical Engineering Times*, August 5, 1991.

[Johnson,92a] Johnson, R. Colin, "Odin Delivers Neural Pack", *Electrical Engineering Times*, March 9, 1992.

[Johnson,92b] Johnson, R. Colin, "Gap Closing Between Fuzzy, Neural Nets", *Electronic Engineering Times*, April 13, 1992.

[Jorgensen, 86] Jorgensen, C., and Matheus, C., "Catching Knowledge in Neural Nets", *AI Expert*, December 1986.

[Kirpatrick, 83] Kirpatrick, S., Gelatt Jr., C.D., and Vecchi, M.P., "Optimization by Simulated Annealing", *Science,* Volume 220, 1983.

[Klimasauskas, 91] Klimasauskas, Casimir, "Applying Neural Networks: Parts I-VI", *PC AI, January-December 1991.*

[Kohnen, 82] Kohonen, T., "Self-Organization Formation of Topologically Correct Feature Maps", *Biological Cybernetics*, Volume 43, 1982.

[Kohonen, 88a] Kohonen, T., *Self-Organization and Associative Memory*, Second Edition, Springer-Verlag, New York, 1988.

[Kohonen, 88b] Kohonen, T., et al., "Statistical Pattern Recognition with Neural Networks: Benchmark Studies", *Proc. of the Second Annual IEEE International Conference on Neural Networks*, Volume 1, 1988.

[Kosko, 87] Kosko, Bart, "Adaptive Bidirectional Associative Memories", *Applied Optics*, Volume 26, 1987.

[Kosko, 92] Kosko, Bart, *Neural Networks and Fuzzy Systems*, Prentice Hall, Englewood Cliffs, NJ, 1992.

[Lapedes, 87] Lapedes, A., and Farber, R., "Non-Linear Signal Processing Using Neural Networks: Prediction and System Modeling", *Los Alamos National Laboratory Report LA-UR-87-2662*, 1987.

[Lippman, 87] Lippmann, Richard P., "An Introduction to Computing with Neural Nets", *IEEE ASSP Magazine*, April 1987.

[Maren, 90] Maren, Alianna, and Harston, Craig, and Pap, Robert, *Handbook of Neural Computing Applications*, Academic Press, ISBN 0-12-546090-2, 1990.

[Matyas, 65] Matyas, J., "Random Optimization", *Automation and Remote Control*, Volume 26, 1965.

[McClelland, 87] McClelland, J.L., and St. John, M., "Three Short Papers on Language and Connectionism", *Carnegie-Mellon University Technical Report AIP-1*, 1987.

[McCord, 91] McCord-Nelson, Marilyn, and Illingworth, W.T., Addison-Wesley, ISBN-0-201-52376-0, 1991.

[McCulloch, 43] McCulloch, Warren S., and Pitts, Walter H., "A Logical Calculus of the Ideas Immanent in Neural Nets", *Bulletin of Mathematical Biophysics*, Volume 5, 1943.

[McEliece, 86] McEliece, R.J., Posner, E.C., Rodemich, E.R., and Venkatesh, S.S., "Capacity of the Hopfield Associative Memory", *California Institute of Technology Internal Report*, 1986.

[Miller, 91a] Miller, Donald L., and Pekny, Joseph F., "Exact Solution of Large Asymmetric Traveling Salesman Problems", *Science*, Volume 251, February 1991.

[Miller, 91b] Miller, W. Thomas, Sutton, Richard S., and Werbos, Paul J., *Neural Networks for Control*, MIT Press, ISBN 0-262-13261-3, 1991.

[Minia, 90a] Minai, A.A., and Williams, R.D., "Acceleration of Back-Propagation through Learning Rate and Momentum Adaptation", *International Joint Conference on Neural Networks,* Volume 1, January 1990.

[Minsky, 69] Minsky, Marvin L., and Papert, Seymour S., *Perceptrons*: *An Introduction to Computational Geometry,* MIT Press, Cambridge, MA 1969.

[Miyazaki,91] Miyazaki, Noboyuki, "Neural Networks Go to Work in Japan", *Electronic Engineering Times,* January 28, 1991.

[Nelson,91] Nelson, M. McCord and Illingworth, W. T., *A Practical Guide to Neural Nets,* Addison-Wesley, Reading, MA, 1991.

[NeuralWare, 91] *Neural Computing,* authored by NeuralWare, Inc. employees for their NeuralWorks Professional II/Plus ANN Development Software, Pittsburg, PA, 1991.

[North,91] North, Robert, "Are Neural Networks Practical Engineering Tools or Ivory-tower Technology?", *Personal Engineering & Instrumentation News*, October 1991.

[Olson, 89] Olson, Willard W., and Huang, Yee-Wei, "Toward Systemic Neural Network Modeling", *IJCNN-89 Proceedings*, IEEE Cat. #89CH2765-6, June 1989.

[Parker, 85] Parker, David B., "Learning-logic", *Report TR-47,* Cambridge, MA: Massachusetts Institute of Technology, Center for Computational Research in Economics and Management Science, 1985.

[Pao, 89] Pao, Yoh-Han, *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, 1989.

[Parker, 87] Parker, D.B., "Optimal Algorithms for Adaptive Networks: Second Order Back Propagation, Second Order Direct Propagation and Second Order Hebbian Learning", *Proc. of the 1st ICNN,* Volume II, 1987.

[Porter,91] Porter, Michael L., "Neural Nets Offer Real Solutions - Not Smoke and Mirrors", *Personal Engineering & Instrumentation News*, November 1991.

[Reece, 87] Reece, Peter, "Perceptrons and Neural Nets", *AI Expert*, Volume 2, 1987.

[Reed, 89] Reed, F., "USPS Investigated Neural Nets", (p. 10) and "Neural Networks Fall Short of Miraculous But Still Work" (p. 28), *Federal Computer Week*, January 23, 1989.

[Reeke, 87] Reeke, G.N., and Edelman, G.M., "Selective Neural Networks and Their Implications for Recognition Automata", *International Journal of Supercomputer Applications*, Volume 1, 1987.

[Rosenberg, 87] Rosenberg, Charles R., "Revealing the Structure of NETtalk's Internal Representations", *In Proc. 9th Annual Conference of the Cognitive Science Society,* 1987.

[Rosenblatt, 58] Rosenblatt, F., "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain", *Psychological Review*, Volume 65, 1958.

[Rumelhart, 85] Rumelhart, D.E., Hinton, G.E., and Williams, R.J., "Learning Internal Representations by Error Propagation", *Institute for Cognitive Science Report 8506*, San Diego, University of California, 1985.

[Rumelhart, 86] Rumelhart, D.E., and McClelland, J.L., editors, "Parallel Distributed Processing : Explorations in the Microstructure of Cognition", *Foundations,*, Volume 1, MIT Press, 1986.

[Samad, 89] Samad, Tariq, "Back-Propagation Extensions", *Honeywell SSDC Technical Report,* Golden Valley, MN, 1989.

[Saridis, 70] Saridis, G.N., "Learning Applied to Successive Approximation Algorithms", *IEEE Transaction on Systems Science and Cybernetics,* Volume SSC-6, 1970.

[Schalkoff,92] Schalkoff, R. J., *Pattern Recognition: Statistical, Structural, and Neural Approaches*, John Wiley & Sons, New York, NY, 1992.

[Scheff, 90] Scheff, K. and Szu, H. "Gram-Schmidt Orthogonalization Neural Networks for Optical Character Recognition", *Journal of Neural Network Computing,* Winter, 1990.

[Schrack, 76] Schrack, G., and Choit, M., "Optimized Relative Step Size Random Searches", *Mathematical Programming,* Volume 10, 1976.

[Scofield, 88] Scofield, C.L., "Learning Internal Prepresentations in the Coulomb Energy Network", *ICNN-88 Proceedings*, IEEE Cat. #88CH2632-8, July 1988.

[Sejnowski, 87] Sejnowski, T.J., and Rosenberg, C.R., "Parallel Networks that Learn to Pronounce English Text", *Complex Systems,* Volume 1, 1987.

[Solis, 81] Solis, F.J., and Wets, R.J.B., "Minimization by Random Search Techniques", *Mathematics of Operations Research*, Volume 6, 1981.

[Specht, 88] Specht, D.F., "Probabilistic Neural Networks for Classification, Mapping or Associative Memory", *ICNN-88 Conference Proceedings*, 1988.

[Specht, 90] Specht, D.F., "Probabilistic Neural Networks", *Neural Networks*, November 1990.

[Stanley,88] Stanely,Jeannette, "Introduction to Neural Networks", Californian Scientific Software, 1988.

[Stork, 88] Stork, David G., "Counter-Propagation Networks: Adaptive Hierarchical Networks for Near-Optimal Mappings", *Synapse Connection*, Volume 1, Number 2, 1988.

[Szu, 87] Szu, H., and Hartley, R., "Fast Simulated Annealing", *Physics Letters 1222(3,4)*, 1987.

[Szu, 88] Szu, Harold, "Fast TSP Algorithm Based on Binary Neuron Output and Analog Neuron Input Using the Zero-Diagonal Interconnect Matrix and Necessary and Sufficient Constraints of the Permutation Matrix", *Proc. of the IEEE First International Conference on Neural Networks*, Volume 2, June, 1987.

[Tank, 86] Tank, David W., and Hopfield, John J., "Simple Neural Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit", *IEEE Transactions on Circuits and Systems*, Volume CAS-33, Number 5, May 1986.

[Uttley, 66] Uttley, Albert M., "The Transmission of Information and the Effect of Local Feedback in Theoretical and Neural Networks", *Brain Research*, Volume 2, 1966.

[Van den Bout, 88] Van den Bout, D.E., and Miller, T.K., "A Traveling Salesman Objective Function That Works", *Proc. of the IEEE First International Conference on Neural Networks*, Volume 2, June 1987.

[Wassermann, 88] Wassermann, Philip D., "Combined Back-Propagation/Cauchy Machine, Neural Networks", *Abstracts of the First INNS Meeting*, Volume 1, Pergamon Press, 1988.

[Wassermann, 89] Wassermann, Philip D., *Neural Computing, Theory and Practice*, Van Nostrand, NY, 1989.

[Watanabe, 85] Watanabe, S., "Theorem of the Ugly Duckling", *Pattern Recognition: Human and Mechanical*, Wiley, 1985.

[Waltrous, 87] Waltrous, R.L., "Learning Algorithms for Connectionist Networks: Applied Gradient Methods of Nonlinear Optimization", *Proc. of the 1st ICNN*, Volume II, 1987.

[Widrow, 60a] Widrow, Bernard, and Hoff, Marcian, "Adaptive Switching Circuits", *1960 IRE WESCON Convention Record*, Part 4, August 1960.

[Widrow, 60b] Widrow, Bernard, "An Adaptive 'Adaline' Neuron Using Chemical 'Memistors'", *Technical Report Number 1553-2,* Stanford Electronics Laboratories, October 1960.

[Widrow, 63] Widrow, Bernard, and Smith, Fred W., "Pattern-Recognizing Control Systems", *Computer and Informations Sciences Symposium Proceedings,* Spartan Books, Washington, DC, 1963.

[Widrow, 64] Widrow, Bernard, "Pattern Recognition and Adaptive Control", *Applications and Industry,* September 1964.

[Widrow, 73] Widrow, Bernard, Bupta, Narendra K., and Maitra, Sidharha, "Punish/Reward: Learning with a Critic in Adaptive Threshold Systems", *IEEE Transactions on Systems, Man, and Cybernetics,* Volume SMC-3, Number 5, September 1973.

[Widrow, 75] Widrow, Bernard, Glover, John R., McCool, John M., Kaunitz, John, Williams, Charles S., Hearn, Robert H., Zeidler, James R., Dong, Eugene, and Goodlin, Robert C., "Adaptive Noise Canceling: Principles and Applications", *Proceedings of the IEEE,* Volume 63, Number 12, December 1975.

[Widrow, 88] Widrow, Bernard, editor, *DARPA Neural Network Study*, AFCEA International Press, ISBN 0-916159-17-5, 1988.

[Willshaw, 76] Willshaw, D.J., and Von Der Malsberg, C., "How Patterned Neural Connections Can Be Set Up by Self-Organization", *Proc. R. Soc. London*, Brit., Volume 194, 1976.

[Wilson, 88] Wilson, G.V., and Pawley, G.S., "On the Stability of the Traveling Salesman Problem Algorithm of Hopfield and Tank", *Biological Cybernetics*, 1988.

[Wright,91] Wright, Maury, "Neural-network IC Architectures Define Suitable Applications", *EDN*, July 4,1991.

[Yager, 80] Yager, R.R., "A Measurement - Information Discussion of Fuzzy Union and Intersection", *International Journal of Man-Machine Studies,* 1980.

[Zadeh, 65] Zadeh, L.A., "Fuzzy Sets", *Information and Control*, 1965.

[Zornetzer, 90] Zornetzer, Steven, F., Davis, Joel L., and Lau, Clifford, *An Introduction to Neural and Electronic Networks*, Academic Press, ISBN-0-12-781881-2, 1990.