2.0 Literature Review

In 1986, the modern era of neural networks was ushered in by the derivation of back propagation. In the short ten years since the rewriting of parallel distributed processing (Rumelhart and McClelland, 1986), an enormous amount of literature has been written on the topic of neural networks. Because neural networks are applied to such a wide variety of subjects, it is very difficult to absorb the wealth of available material. A brief history of neural networks has been written to give an understanding of where the evolution of neural networks started. A detailed review has also been written for this study of the feed-forward neural network and the back propagation algorithm. Papers on various topics related to this study are detailed to establish the need for the proposed work in this study. However, ten years is not a very long time for research, so no one book has distinguished itself as the leading authority in the area of neural networks.

2.1 History Of Neural Networks

The history of neural networks can be traced back to the work of trying to model the neuron. The first model of a neuron was by physiologists, McCulloch and Pitts (1943). The model they created had two inputs and a single output. McCulloch and Pitts noted that a neuron would not activate if only one of the inputs was active. The weights for each input were equal, and the output was binary. Until the inputs summed up to a certain threshold level, the output would remain zero. The McCulloch and Pitts' neuron has become known today as a *logic circuit*.

The *perceptron* was developed as the next model of the neuron by Rosenblatt (1958), as seen in Figure 2.1. Rosenblatt, who was a physiologist, randomly interconnected the perceptrons and used trial and error to randomly change the weights in order to achieve "learning." Ironically, McCulloch and Pitts' neuron is a much better model for the electrochemical process that goes on inside the neuron than the perceptron, which is the basis for the modern day field of neural networks (Anderson and Rosenfeld, 1987).

The electrochemical process of a neuron works like a voltage-to-frequency translator (Anderson and Rosenfeld, 1987). The inputs to the neuron cause a chemical reaction such that, when the chemicals build to a certain threshold, the neuron discharges. As higher inputs come into the neuron, the neuron then fires at a higher frequency, but the magnitude of the output from the neuron is the same. Figure 2.2 is a model of a neuron. A visual comparison of Figures 2.1 and 2.2 shows the origins of the idea of the perceptron can be traced back to the neuron. Externally, a perceptron seems to resemble the neuron with multiple inputs and a single output. However, this similarity does not really begin to model the complex electrochemical processes that actually go on inside a neuron. The perceptron is a very simple mathematical representation of the neuron.

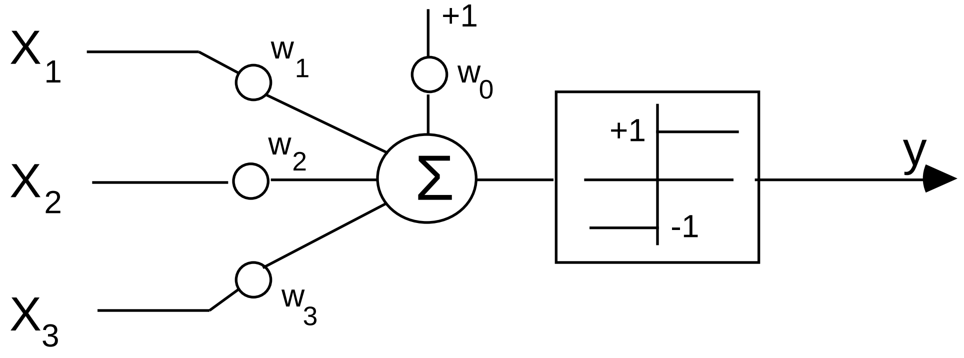

Figure 2.1 The Perceptron

Selfridge (1958) brought the idea of the weight space to the perceptron. Rosenblatt adjusted the weights in a trial-and-error method. Selfridge adjusted the weights by randomly choosing a direction vector. If the performance did not improve, the weights were returned to their previous values, and a new random direction vector was chosen. Selfridge referred to this process as *climbing the* *mountain,* as seen in Figure 2.3. Today, it is referred to as *descending on the gradient* because, generally, error squared, or the energy, is being minimized.

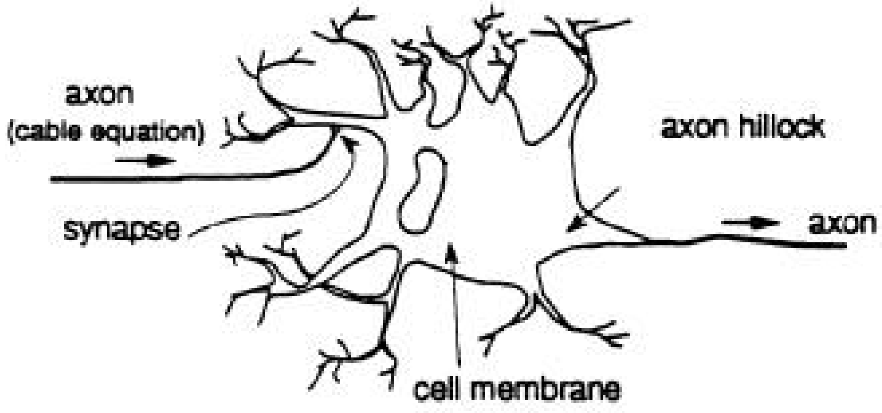

Figure 2.2 The Neuron

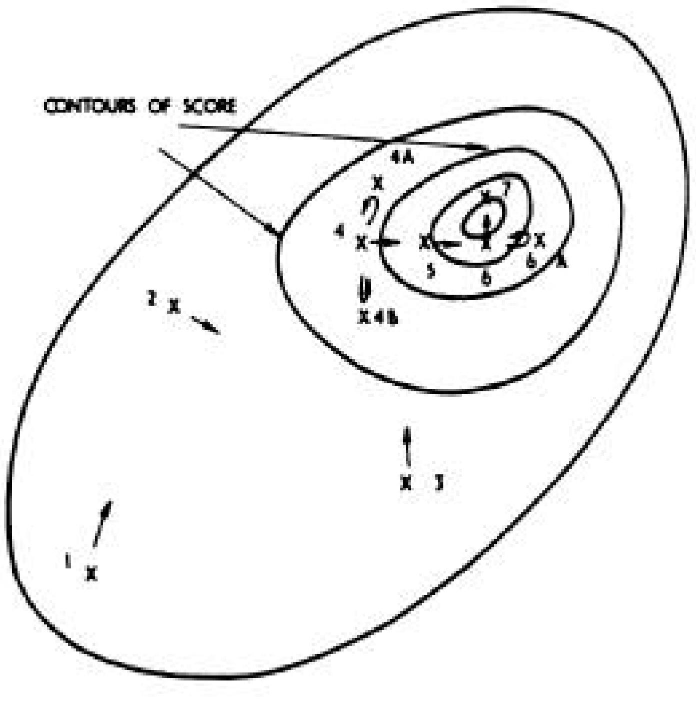

Figure 2.3 Climbing the Mountain

Widrow and Hoff (1960) developed a mathematical method for adapting the weights. Assuming that a desired response existed, a gradient search method was implemented, which was based on minimizing the error squared. This algorithm would later become known as LMS, or Least Mean Squares. LMS, and its variations, has been used extensively in a variety of applications, especially in the last few years. This gradient search method provided a mathematical method for finding an answer that minimized the error. The learning process was not a trial-and-error process. Although the computational time decreased with Selfridge's work, the LMS method decreased the amount of computational time even more, which made use of perceptrons feasible.

At the height of neural network or perceptron research in the 1960's, the newspapers were full of articles promising robots that could think. It seemed that perceptrons could solve any problem. One book, Perceptrons (Minsky and Papert, 1969), brought the research to an abrupt halt. The book points out that perceptrons could only solve linearly separable problems. A perceptron is a single node. Perceptrons shows that in order to solve an n-separable problem, n-1 nodes are needed. A perceptron could then only solve a 2-separable problem, or a linearly separable problem.

After Perceptrons was published, research into neural networks went unfunded, and would remain so, until a method was developed to solve n-separable problems. Werbos (1974) was first to develop the back propagation algorithm. It was then independently rediscovered by Parker (1985) and by Rumelhart and McClelland (1986), simultaneously. Back propagation is a generalization of the Widrow-Hoff LMS algorithm and allowed perceptrons to be trained in a multi-layer configuration, thus a n-1 node neural network could be constructed and trained. The weights are adjusted based on the error between the output and some known desired output. As the name suggests, the weights are adjusted backwards through the neural network, starting with the output layer and working through each hidden layer until the input layer is reached. The back propagation algorithm changes the schematic of the perceptron by using a sigmoidal function as the squashing function. Earlier versions of the perceptron used a signum function. The advantage of the sigmoidal function over the signum function is that the sigmoidal function is differentiable. This permits the back propagation algorithm to transfer the gradient information through the nonlinear squashing function, allowing the neural network to converge to a local minimum. Neurocomputing: Foundations of Research (Anderson and Rosenfeld, 1987) is an excellent source of the work that was done before 1986. It is a collection of papers and gives an interesting overview of the events in the field of neural networks before 1986.

Although the golden age of neural network research ended 25 years ago, the discovery of back propagation has reenergized the research being done in this area. The feed-forward neural network is the interconnection of perceptrons and is used by the vast majority of the papers reviewed. A detailed explanation is given in Section 2.2 for the feed-forward neural network and the back propagation algorithm because the feed-forward neural network will be the cornerstone of the work done in this study.

2.2 Feed-Forward Neural Network

The feed-forward neural network is a network of perceptrons with a differentiable squashing function, usually the sigmiodal function. The back propagation algorithm adjusts the weights based on the idea of minimizing the error squared. The differentiable squashing function allows the back propagation algorithm to adjust the weights across multiple hidden layers.

By having multiple nodes on each layer, n-separable problems can be solved, like the Exclusive-OR, or the XOR problem, which could not be solved with only the perceptron. Figure 2.4 shows a fully connected feed-forward neural network; from input to output, each node is connected to every node on the adjacent layers.

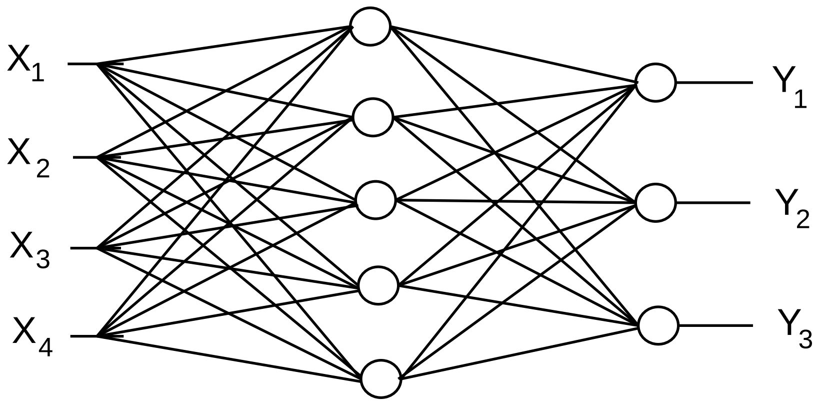

Figure 2.4 Fully-Connected, Feed-Forward Neural Network

In Figure 2.5, the individual nodes, or perceptrons, are representative of the neuron. The input to the node is the input to the neural network or, if the node is on a hidden layer or the output layer, the output from a previous layer. The node is the key to the training of the neural network. The back propagation algorithm propagates the changes to the weights through the neural network by changing the weights of one individual node at a time. With each iteration, the difference between the neural network's output and the desired response is calculated. In the case of a single output, the output of the entire neural network is the output of one individual node whose inputs are the outputs of nodes on the previous layer. By breaking the neural network down to the nodes, the training process becomes manageable. The back propagation algorithm is an LMS-like algorithm for updating the weights. Below is the derivation of the back propagation algorithm, which tries to minimize the square of the error (Rumelhart and McClelland, 1986).

The variables for the derivation of back propagation are defined as follows:

X is the input vector of the node;

W is the vector of weights of the node;

y is the output of the node;

d is the desired response of the node;

e is the difference between output of the node and the desired response;

$\hat{\zeta}$ is the partial differential with respect to the weights;

s is the value inputted into the squashing function;

μ is the learning rate.

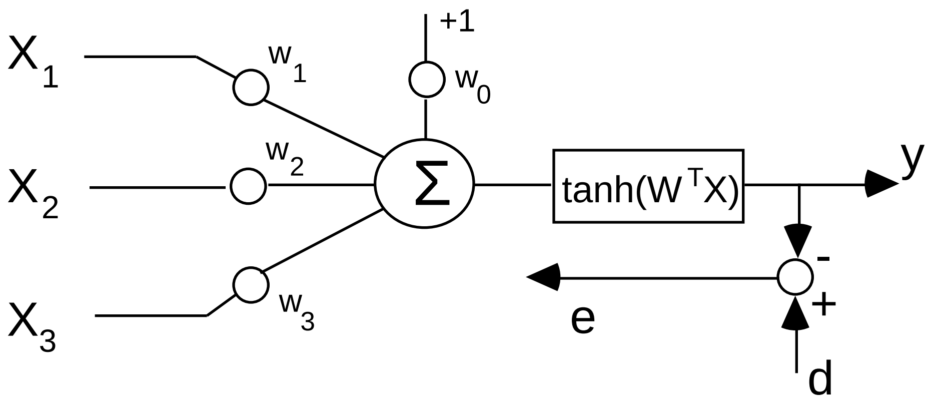

Figure 2.5 A Node

Equation 2.1 is the definition of the error .

$$e = d - y \qquad (2.1)$$

Equations 2.2 and 2.3 are the partial differential of the error with respect the weights.

$$\hat{\zeta} = \frac{\partial e^T e}{\partial W} \qquad (2.2)$$

$$\hat{\zeta} = \frac{\partial}{\partial W}(d - y)^T (d - y) \qquad (2.3)$$

Equation 2.4 is the application of the chain rule to the partial differential.

$$\hat{\zeta} = -2(d-y)\frac{\partial y}{\partial W} = -2(d-y)\frac{\partial y}{\partial s}\cdot\frac{\partial s}{\partial W} \qquad (2.4)$$

Equation 2.5 is the derivative of the squashing function.

$$y = \tanh(s) \qquad \frac{\partial y}{\partial s} = 1 - y^2 \qquad (2.5)$$

Equation 2.6 is the definition of s.

$$s = W^T X \qquad (2.6)$$

Equation 2.7 is the partial differential of s with respect to the weights of the node.

$$\frac{\partial s}{\partial W} = X \qquad (2.7)$$

Equation 2.8 is the substitution of variables into the partial derivative of the error squared.

$$\hat{\zeta} = -2(d-y)(1-y^2)X \qquad (2.8)$$

Equation 2.9 is the change to the weights to be made.

$$\Delta W = -2\mu(d-y)(1-y^2)X \qquad (2.9)$$

The equation for changing the weights is a very simple LMS-like equation that includes a single term not in the LMS equation. The term comes from the hyperbolic tangent function, whose derivative does not require much computational power. The simple weight update equation is applied to each node in the neural network. It is a gradient method that will converge to a local minimum.

During the training process, the inputs enter the neural network and get summed into the first layer of nodes. The outputs from the first layer of nodes get summed into the second layer of nodes. This process continues until the output comes from the neural network. The output is compared to the desired output, and the error is calculated. The error is used to adjust the weights backwards through the neural network. The weight adjustment equation has one shortcoming; the weights on a particular node cannot be the same as another node on the same layer because the weights will be adjusted the same for each node that has identical weights. If all of the neural network's weights were initially set at zero, the weights would be adjusted the same on each layer. Mathematically, it would be equivalent to having a single node per layer. This is why the weights of the neural network need to be randomly initialized when there is a multi-layer neural network configuration. The other reason for randomly initializing the weights is to properly search the weight space, which is not a quadratic function, as is the linear perceptron. The randomly initialized weights make it very difficult to estimate the initial performance of the control system.

2.3 Neural Networks In Control Applications

Controls is only one area in which neural networks have been applied, yet controls has its own unique set of problems to solve when applying any methodology. The main principle behind controls is to change the performance of a system to conform to a set of specifications. This goal can be complicated by uncertainties in the system, including nonlinearities. Control theory has been trying to develop methodologies to handle ever increasing amounts of uncertainties. Neural networks can be applied when no a priori knowledge of the system exists. Unfortunately, there is rarely a complete model of the system, but often, there is a partial model of the system available.

Narendra and Parthasarathy (1990) initiated activity in developing adaptive control schemes for nonlinear plants. Many of Narendra's papers have dealt with the control and identification of a system using neural networks. Nguyen and Widrow (1990) worked on self-learning control systems. Widrow's papers have a long history of control and identification problems using neural networks. His assumptions have included no a priori knowledge and open-loop control only. Widrow's work has never included closed-loop feedback, with the exception of his suggestion to stabilize an unstable system with feedback, and then to use a neural network to achieve the specified performance. Narendra and Widrow have done much ground work in the field of neural network based control; most of the papers reviewed in the following sections reference the work done by them. The following sections review in detail papers on specific topics in the area of neural networks in control applications.

2.3.1 A-Priori Information

Only a limited amount information about any system is going to be known. Hence, it is not good design methodology to throw out any of the a priori system knowledge. Integration of the system's information into neural network control systems has been studied, and the use of a priori information has been suggested in several places. Selinsky and Guez (1989) and Iiguni and Sakai (1989) trained the neural network off-line with the known system dynamics before applying the neural network controller to the actual system. Joerding and Meador (1991) constrained the weights of the neural network using a priori knowledge through the modification of the training algorithm. Nordgren and Meckl (1993) incorporated a priori knowledge through a parallel control path to the neural network. The development of a neural network structure, called CMAC, incorporated knowledge into a topographical weight map (Miller, Sutton, and Werbos, 1990). Pao (1989) developed a technique for enhancing the initial representation of the data to the neural network by replacing the linear inputs with functional links. Brown, Ruchti, and Feng (1993) incorporated a priori knowledge into the system as the output layer of the neural network, called a *gray layer*. The use of a priori knowledge is very important to the design of a fast, effective controller.

Selinsky and Guez (1989) and Iiguni and Sakai (1989) used knowledge of the system to train the neural network off-line, a very common practice in neural network control. Selinsky's and Iiguni's papers are typical examples of the use of a priori knowledge. The basic idea is to create a model of the system with as much detail as available and to use it to train the neural network, as seen in Figure 2.6. The input for the training set is usually colored noise, which would get its frequency content from the expected input to the actual system. Once the neural network is trained, it is connected to the actual system. The problem with this method is that, if the model is not precisely correct, the nonlinearities of the neural network, interacting with the nonlinearities of the actual system, may not perform as expected.

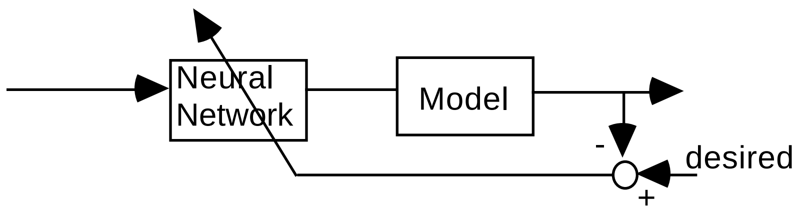

Figure 2.6 Off-line Training

A similar idea is used by many other researchers, such as Narendra and Parthasarathy (1990). They assumed access to the actual system and created a neural network model of the system. The neural network model is used to train the neural network controller. This method works almost as well as using the actual system for training. The first problem is to create a good model of the system using a neural network. This method relies on the neural network finding its own correlations between the inputs and the outputs. The second problem is to create a neural network controller to control the model, which also relies on the neural network to find its correlation. Neither of the two previous methods use a priori information to directly influence the working of the neural network controller.

Joerding and Meador (1991) constrained the weights of the neural network using a priori knowledge in a modified training algorithm. They addressed the problem of incorporating a priori knowledge about an optimal output function into specific constraints. The two general approaches are an Architecture Constraint method and a Weight Constraint method. Both assume the knowledge of the form of the optimal output function, such as monotonic and concavity. A monotonic function is one whose slope does not change sign, and a concave (convex) function has a slope that decreases (increases) as the function arguments increase. The desired output of the neural network is constrained to these function types. The two methods are used to exploit the mathematical nature of the feed-forward neural network with a hyperbolic tangent squashing function. The hyperbolic tangent is monotonic and concave; the sign of the hyperbolic tangent is the same as the sign of its argument. The modified training algorithm consists of the back propagation term plus the derivative of the optimal function. It is an interesting idea to encode the a priori information into the neural network. These methods work well for modeling the system. However, they are not directly translatable into a controller application.

A Cerebellar Model Arithmetic Computer, or Cerebellar Model Articulation Controller (CMAC), neural network is a table look-up technique for representing a complex, nonlinear function, f(s) (Miller, Glanz, and Kraft, 1987). The original work on the CMAC neural network was done by Albus (1975). Each point in the input space, S, maps into C locations in the N-dimensional memory A. The values of function f(s) are then determined by summing the values at each corresponding location in A. The training data, F_O, of the CMAC neural network for the input state, s_O, is used to train the weights inside the look-up table. The correction factor, δ, can be determined from Equation 2.10:

$$\delta = \beta * (F_o - f(s_o)) / C \qquad (2.10)$$

where β is a training factor between 0 to 1. For each element of the training data available, δ can be computed and added to each of the C memory locations. If $\beta = 1$, $f(s_O) = F_O$ as the result of the training step. If $\beta < 1$, $f(s_O)$ is changed in the direction of F_O. The determination of the function $f(s_O)$ is based on the nonlinear system to be controlled. The function is usually a pseudo-inverse of the system. Without knowledge of the nonlinearity, it is very difficult to use a CMAC neural network.

Pao (1989) developed a technique for enhancing the initial representation of the data to the neural network by replacing the linear inputs with functional links. Functional links are an attempt to find simple mathematical correlations between the input and output, such as periodicity or higher-order terms. Functional links are very important in preprocessing the data for the neural network. A functional link is sometimes called *conditioning the input*. There is a parallel between adaptive control and neural networks. Adaptive control has a method called *the MIT rule* in which the input to the adaptive controller is limited to an order of magnitude of zero (Astrom and Wittenmark, 1995). The MIT rule allows the adaptive scheme to adjust to the adaptive coefficients without the magnitude of the input overwhelming the coefficients. A functional link in its simplest form could constrain the input of the neural network. If the input of the neural network is ill-conditioned, the functional link makes the input more usable by the neural network. Functional links can decrease the amount of work done by the neural network by structuring the input such that the correlation of the input to output is easier to see by the neural network. If a priori knowledge of the system contains information such that a function link can be used, the functional link is very useful; if a priori knowledge does not, the functional link is limited.

Brown, Ruchti, and Feng (1993) developed a method called *a gray layer*. A gray layer uses the output of the neural network to incorporate a priori information of the system, as seen in Figure 2.7. Their paper includes a change to the training method to propagate the error through the gray layer to the weights. The error needs to be propagated through the gray layer in order to converge the weights of the neural network. The authors exert that the gray layer has a decided advantage in the identification of uncertain nonlinear systems. The exploitation of such information is usually beneficial, resulting in the selection of more accurate identification models and a faster rate of parameter convergence (Ljung, 1987). The gray layer requires knowledge of the nonlinearities, which is often the most difficult part of a model to obtain.

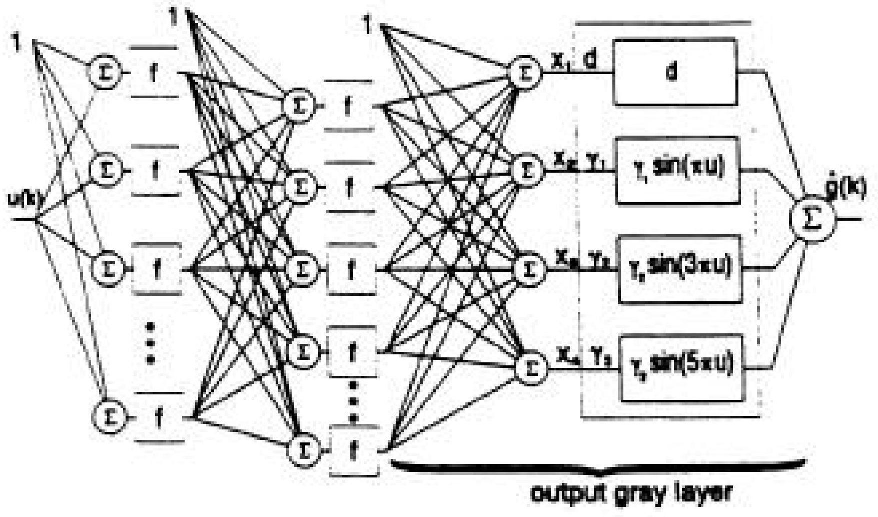

Figure 2.7 Neural Network with Gray Layer

CMAC neural networks, functional links, and gray layers are dependent on knowing the nonlinearities of the system. They are all very useful in the appropriate situations. There are many methods for incorporating a priori knowledge into the neural network. Each method seems to need a specific kind of knowledge, and in many situations, each can be used to a limited degree. If the nonlinearities of a system are known, CMAC, functional links, and gray layers can be used to reduce the problem to a pseudo-linear problem, which can be trained quickly and effectively. Often, it is the linear parameters of a system that are known. Adding a parallel classical controller to the neural network controller is a possibility. However, for all of these methods the neural network controller is an unpredictable factor in the control of the system.

2.3.2 Direct and Indirect Adaptive Control

Direct and Indirect adaptive control are two methods for applying neural networks to control systems. Direct adaptive control can applied when a viable model for the plant exists. Indirect adaptive control is applied when a model must developed by a second neural network. The work for the two methods using neural networks was originally done by Narendra and Parthasarathy (1990). There are several other researchers that have followed up the work. Tanomaru and Omatu (1991) applied to methods to the inverted pendulum problem. Greene and Tan (1991) applied the indirect adaptive control to a two-link robot arm. Both methods make use back propagation to adjust the weights of the neural networks.

Direct adaptive control can be applied when a model of the plant exists. The update algorithm uses the Jacobian to develop a gradient for convergence, as seen in Figure 2.8. The controller adapts to the reference model. Since the plant lies between the adaptive neural network and the output error that is be minimized, the error must be back propagated through the plant's Jacobian matrix. This procedure requires the knowledge of the Jacobian. For SISO plants, the partial derivatives can be used to replaced the Jacobian. An alternative is to assume that the only the signs of the elements of the Jacobian are known and that variable learning rates in the back propagation algorithm compensate for the absolute values of the derivatives.

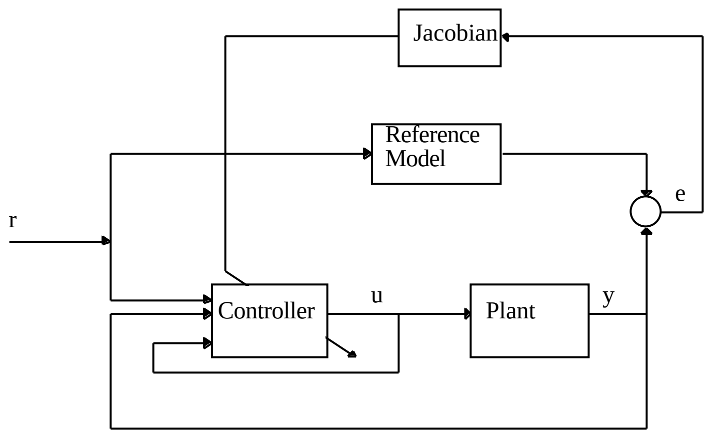

Figure 2.8 Direct Adaptive Control

A serious drawback to direct adaptive control is that it requires some knowledge of the plant. The indirect adaptive control scheme does not require any knowledge of the plant. It does require two neural networks: a plant emulator and a controller. A block diagram of the indirect adaptive control method can be seen in Figure 2.9. The plant emulator is a feed-forward neural network and should be trained off-line with a data set sufficiently large to allow for identification. The emulator provides an efficient way to calculate the derivatives of the plant via back propagation. This allows the parameters of the controller to be adjusted by considering the two networks as parts of a bigger one. The training process of both networks can be performed on-line. Narendra and Parthasarathy (1990) developed the convergence process with static and dynamic back propagation. By using the static back propagation, the derivatives of the output of the plant are calculated for the dynamic back propagation, which updates the weights of the controller. The indirect adaptive control approach is particularly interesting when there is not a model of the plant.

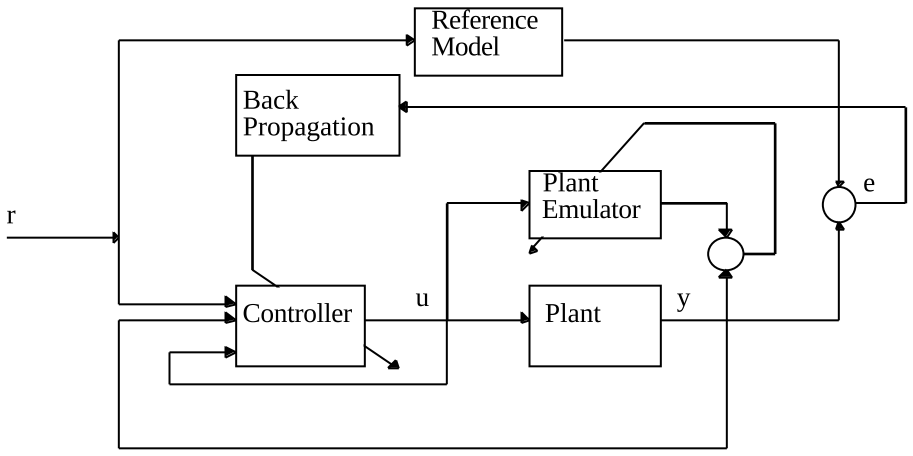

Figure 2.9 Indirect Adaptive Control

The direct and indirect adaptive control methods have been applied to a wide variety of control problems. The direct adaptive control method works very well if the model of the plant and the plant have similar Jacobian matrices. The indirect method works well if there is sufficient data to create a model off-line. Both methods rely on back propagation to converge the weights of the neural networks. The research into the area of direct adaptive on control was carried the next step further with the addition of fixed-gain controller inside a closed loop.

2.3.3 Closed-Loop, Fixed-Gain Controller

Nordgren and Meckl (1993) used a classical PD controller in a parallel path to the neural network controller, as seen in Figure 2.10. The a priori information is used to create a model of the system, and from that model, a classical controller is built to control the actual system. A neural network controller is placed in a parallel path to the classical controller to supplement the classical controller and to increase the performance of the system. The adaptive law is based on the a priori knowledge of the plant and the system. This idea can be seen in several different papers, such as Jin, Pipe, and Winfield (1993) and Chen and Chang (1994). The classical controller is a method of incorporating a priori knowledge into the system. However, the neural network is still randomly initialized, giving it an unknown initial gain. The performance and stability of the system is difficult to predict, and the neural network controller has to find its own correlation in the data.

Psaltis, Sideris, and Yamamura wrote a series of papers about a neural network in the closed-loop including Psaltis, Sideris, and Yamamura (1988) and Yamamura, Sideris, Ji, and Psaltis (1990). The papers use three independent neural networks to control a nonlinear plant. The three neural networks are set up as a prefilter, a feed-forward controller, and a feedback controller. The different learning techniques are used to train the three different neural networks. Indirect Learning and General Learning Architectures use back propagation to teach the prefilter and the feed-forward controller. Specialized Learning Architecture is used to teach the feedback controller. They developed a new algorithm to train the neural network. The new algorithm thinks of the plant as another layer to the neural network, and the partial derivatives of the plant at its operating point are used to train through the plant. This method requires knowledge of the nonlinearities of the plant.

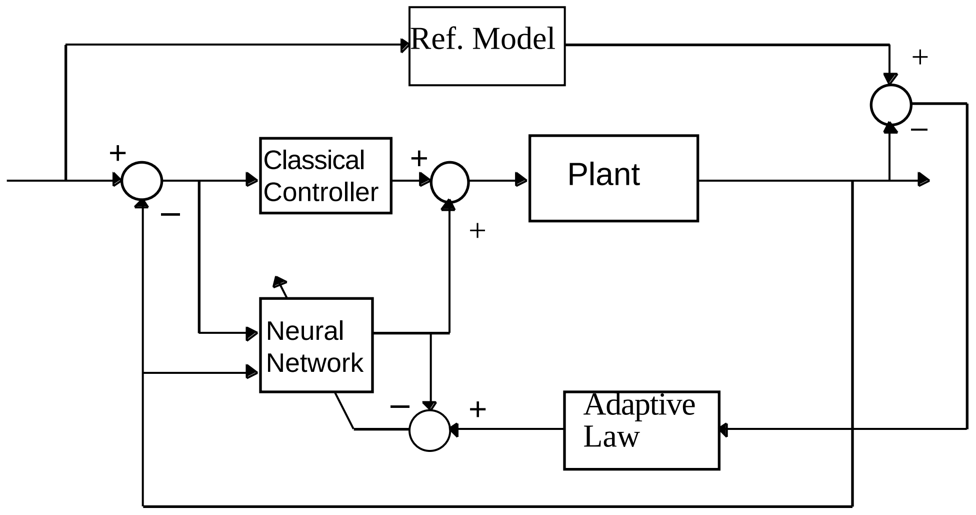

Figure 2.10 Parallel Control Path for Neural Network Controller

Lightbody and Irwin (1995) placed a neural network controller in the closed-loop and parallel to a PID controller. The update algorithm that was developed was a gradient-based training algorithm, which used a Jacobian cost function to determine the gradient. They compared their results to a Lyapunov model reference adaptive controller.

2.3.4 Stability

When working with neural network based control, stability is required for the neural network and the overall system. Stability criteria must be established for both for the controller and the controlled system. Vos, Valavani, and von Flotow (1991) comment that a problem with neural network use is the lack of guaranteed stability for the weight update. They do not propose a stability guarantee but discount neural networks as a plausible controller because of the lack of stability.

Perfetti (1993) developed a proof of asymptotic stability of equilibrium points. To characterize the local dynamic behavior near an isolated equilibrium point, it is sufficient to construct the Jacobian matrix of the linearization around the equilibrium and to check its eigenvalues. If all such eigenvalues have negative real parts, the equilibrium point is asymptotically stable. This approach, called *Lyapunov's first method*, is impractical for neural networks, as their order of complexity is usually very large. Perfetti showed through the use of Gerschgorin's disks that the slopes around an equilibrium point are all greater than zero, thus proving that the neural network at an equilibrium point is stable.

Renders, Saerens, and Bersini (1994) proved the input-output stability of a certain class of nonlinear discrete MIMO systems controlled by a multi-layer neural network with a simple weight adaptation strategy. The stability statement is only valid, however, if the initial weight values are not too far from the optimal values that allow perfect model matching. The proof is based on the Lyapunov formalism. They proposed to initialize the weights with values that solve the linear problem. This research is an extension of Perfetti's paper, showing that, if there are no local minima between the initial weights and the global minimum, the weights will asymptotically converge on the global minimum.

Bass and Lee (1994) developed a method for linearizing nonlinear plants with neural networks, resulting in robustly-stable closed-loop systems. In this method, neural network outputs are treated as parametric uncertainty and combined with other plant uncertainties so that a robust controller can be designed. An algorithm for confining the network's output to be less than a given bound is presented. This method has an inner and an outer loop. The outer feedback loop is designed to robustly stabilize the entire system; the inner feedback loop, composed of fixed linear gains and an on-line adaptable neural network, is used to invert the plant's dynamics. From a priori information of the plant and the linear plant approximation, the saturation limit on the neural network can be calculated. Because the neural network's output uncertainty is bounded, the robust controller can be developed.

A stability proof was developed for the linear perceptron, which limits the learning rate to one over the maximum eigenvalue of the system. No stability proof has been developed for the neural network that limits its learning rate. Perfetti showed that an equilibrium point is asymptotically stable. Renders and Bersini showed that the neural network will asymptotically converge on the global minimum if there are no other local minimums between the initial weights and the global minimum weights. Bass and Lee bounded the output of the neural network, then designed a robust controller to stabilize the entire system. Because no global stability proof exists, the papers have tried to develop stability proofs for various aspects of the neural network spectrum, but none have broad applications.

2.3.5 Performance

The design of a controller is generally based on performance criterion, such as rise time, percent overshoot, and settling time. Performance is defined as how well the overall system meets the performance criterion. A question arises if the system does not meet rise time but does meet all the other criterion: is the performance acceptable? Performance is subjective.

Hao, Tan and Vandewalle (1993) said that the difficulty in applying the supervised learning for actual control problems is that it is not always clear what the targets are for the neural networks to learn. Supervised learning is characterized by the existence of training data consisting of input vectors and corresponding desired output vectors. However, when the data necessary for supervised learning is not directly available, reinforcement learning should be used to optimize some performance function. The paper stopped short of applying reinforcement learning, but it did try to extract rules for the controller based on a human controlling the system.

Miller, Sutton, and Werbos (1990) discussed performance evaluation and performance criteria. The parameter estimation process, which is very straight forward, has many pitfalls when it comes to problem representation, performance criteria, estimation methods, and the use of a priori knowledge. System identification and control are two mutually exclusive ideas. Miller et al (1987) contended that all control applications are really reinforcement learning processes that are more difficult to learn than a supervised learning process. The controller's own dynamics are factored into the overall system performance. The error of the system is not directly related to the real error of the neural network. Instead, the neural network must see the trends and not the actual error generated by the overall system. The learning process should be in the form of a critic rather than a simple mathematical error.

Okafor and Adetona (1995) discussed, in a practical application of neural network based control, the effects of different neural network structures on the performance of a particular system. Performance was based on the amount of training time and how well the results from the neural network matched the desired response. They presented a systematic evaluation of the individual effects of different training parameters: the learning rate, the number of hidden layer nodes, and the squashing function. Increasing the number of hidden nodes had little effect on the prediction error, but the number of training cycles increased dramatically once they passed an optimal number of nodes. Increasing the learning rate had little effect on prediction error, until the learning rate made the neural network unstable and decreased the number of training cycles until the learning rate destabilized the system. Three different types of squashing functions were tried: sigmoid, hyperbolic tangent, and sine. The hyperbolic tangent had the least amount of prediction error and was slightly worse than the sine for the number of training cycles needed. The sigmoid function was not very good overall. The results would be different if the problem that the neural network had been trying to solve was different; however, the trends will stay the same.

The performance of the controller is measured by the performance of the overall system. The papers covered discuss the difficulty in training the neural network with the performance criteria, and the effects different neural network structures have on performance but do not give any clear practices for integrating performance criteria into the neural network controller.

2.3.6 Reinitialization

A commonly-used heuristic is to train a neural network using a large number of different initial weights. The converged neural network weight with the lowest mean squared error is selected as the optimal neural network. Kolen and Pollack (1991) showed that the back propagation algorithm is sensitive to the initial weights, and the training may be regarded to be an unstable system from a traditional signals and systems viewpoint. Nguyen and Widrow (1990) showed that, for flexibility in training the neural network, the initial weights should be based on a piece-wise linear method. Kim and Ra (1991) extended the work of Nguyen and Widrow by proposing a method that suggests the minimum bound of the weights based on dynamics of the decision boundaries. Osowski (1993) developed a piece-wise linear principle resulting in initial neural network's weights that are better able to form a function than randomly initialized weights. Schmidt et al. (1993) developed a method for reinitializing the weights of the neural network by using a probability distribution of the mean squared error. Sutter, Dixon, and Jurs (1995) used generalized simulated annealing to initialize the neural network. A proper initialization method can reduce convergence time and increase stability. Each method is covered in greater detail below.

Nguyen and Widrow (1990) formulated a method for initialization of the weights of neural networks to reduce training time. The idea is to bound the initial weight of the neural network such that there is a piece-wise linear solution. When using a hyperbolic tangent squashing function, the output from a node is bound from -1 to +1. It is very easy to saturate the hyperbolic tangent. During training, the neural network learns to implement the desired function by building a piece-wise linear approximation to the function. The pieces are summed together to form the complete approximation. This method expects each node to contribute approximately the same amount. The initial weights are selected from a uniformly-random distribution bounded on the positive and negative sides by the number of nodes on the layer and the range of the inputs, such that the nodes are not saturated. It is a general method that is used to this day and has its own matlab function, nwtan(A,R), where A is the number of nodes on the layer, and R is a matrix with the number of rows equal to the number of inputs and two columns that give the range for each input.

Kim and Ra (1991) proposed a method to use the minimum bound of the weights based on the dynamics of decision boundaries, which are derived from the generalized delta rule. The combination of the incoming weights of a node with its internal threshold forms a decision plane in the output space. During the learning process by the back propagation algorithm, the weight values of all the nodes are updated at each iteration step, and all the decision planes converge to the locations of minimum LMS error. By watching the dynamics of the decision planes, a useful trend of the weight values can be conceived for the stable and fast convergence. As the weights change during convergence, a trend by the weights can be seen in a new reference plane based first on the normalized weights, then the back propagation algorithm gets caught in a local minimum, and finally, the weights can be reinitialized based on the trends seen in the reference plane. The two key assumptions are (1) that the back propagation tends towards the global minimum and (2) that the weight space of the neural network is conditioned enough to yield the near global weight solution.

Osowski (1993) developed a method based on the piece-wise linear principle result. This method is based on the magnitude of the inputs and tries to have the middle or linear portion of the squashing function activated. It requires that the solution also be known. The number of nodes on the hidden layer is to be equal to the number piece-wise sections of the curve to be approximated. Each node is to approximate a section of the desired curve, and the weights are chosen appropriately. By assigning a section of the desired curve to a node, the weights can be initialized based on the expected inputs for that section of the curve. This method is not very applicable to control applications because the solution must be known.

Schmidt et al. (1993) developed an idea to reinitialize the network based on a probability distribution of mean squared error. They advocate the idea that training a neural network using the back propagation method is a stochastic process. Important for the expected performance is the joint probability distribution of the mean squared error that is optimized and the probability of the error for the entire population. Once the back propagation algorithm converges, the neural network is reinitialized with weights based on the weights just converged upon, but adjusted according to a probability distribution of the mean squared error. The larger the mean squared error, the further the weights are adjusted from their current values. This is a very good idea if the weight space is well-conditioned and the back propagation algorithm is always tending towards a global solution.

Sutter, Dixon and Jurs (1995) used a neural network to solve a chemical engineering problem. In this paper, generalized simulated annealing is employed to model molecular structures and to predict their toxicity. Generalized simulated annealing is an alternative to the gradient search method of back propagation. It is included in this section because it uses a kind of global search pattern of trial and error and can be thought of as a whole series of reinitializations. It is called *annealing* because there are parallels between annealing and this method. Annealing is the process of heating and cooling a metal until it achieves the strength that is desired. This method is similar because several different sets of weights are tried (heating) until a set of weights has good characteristics, then several more sets of weights are tried in the region around the good set of weights (cooled). If the best of this group does not meet specifications, the process is started all over again (reheated). This method has many advocates, but it is really a trial-and-error method without any real mathematical basis.

The methods developed for initializing the weights of the neural network are based in the mathematics of the neural network and its nonlinearities. They can be employed for this study, but none are directly applicable to control application and the incorporation of a priori information into the neural network. However, the methods have some productive ideas that can be implemented for control applications.

2.3.7 Time To Convergence

The amount of time needed for convergence is determined by several factors. The back propagation algorithm, the learning rate, and the squashing function are some of the factors that influence the rate of convergence but are outside the scope of the study. The use of a priori information, stability, pre- and post-processing the data, and initialization of the weights are some of the factors this study will examine. Nguyen and Widrow (1990), Kim and Ra (1991), and several other of the papers in section 2.3.4 stated that the primary reason for developing method for initial weight estimation was to reduce time to convergence. Pao (1989) stated that using functional links should reduce time to convergence. Brown, Ruchti, and Feng (1993) commented on reducing time to convergence by using gray layers. Reducing the time needed to converge is a very important goal when working with neural networks.

2.3.8 Examples - Inverted Pendulum

The inverted pendulum problem is a widely-used benchmark for comparing different types of controllers, especially neural network controllers. It is a difficult nonlinear control task to balance an inverted pendulum. A linear controller can be implemented for the inverted pendulum, but it has limited range for the initial conditions and is sensitive to parameter changes. Widrow and Smith (1963) used a single ADALINE logic circuit to control a "broom-balancer," or inverted pendulum. Barto, Sutton, and Anderson (1983) has become the standard for many of the papers, such as Geva and Sitte (1993) and Hung and Fernandez (1993) when it comes to the basic experimental test bed for the cartpole or inverted pendulum problem. Many others have applied various types and methods of neural networks to the inverted pendulum problem.

Widrow and Smith (1963) introduced neural network control to the cart and pendulum problem. The use of a single ADALINE (adaptive linear neuron) to control an inverted pendulum was the first practical application of the LMS algorithm. The LMS algorithm was not developed until 1960, so the inverted pendulum experiment was one of the first dynamical applications of LMS. The neural network was not trained to control the cart and pendulum problem. Rather, the control function was learned by the neural network off-line. This requires the knowledge of solution before the controller is placed on-line. In the state space, the desired switching surface is represented and learned by the neural network, as shown in Figure 2.11. In order to apply this method, the switching surface needs to be known.

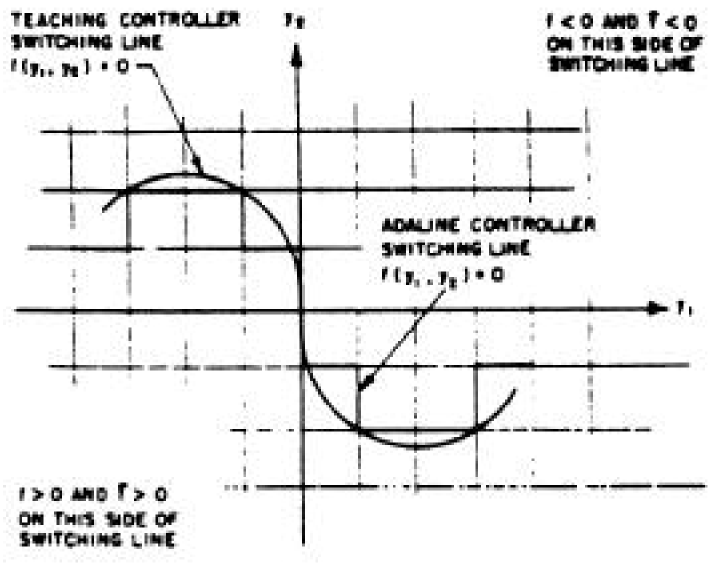
TEACHING CONTROLLER SWITCHING LINE
$f(y_1, y_2) = 0$
y_2
$f < 0$ AND $\dot{f} < 0$ ON THIS SIDE OF SWITCHING LINE
ADALINE CONTROLLER SWITCHING LINE
$f(y_1, y_2) = 0$
y_1
$f > 0$ AND $\dot{f} > 0$ ON THIS SIDE OF SWITCHING LINE

Figure 2.11 The Desired Switching Surface in State Space

Barto, Sutton, and Anderson's (1983) work was very similar to Widrow and Smith's (1963) work. They referred to the neuron as the Associate Search Element (ASE) and to the learning algorithm as the Adaptive Critic Element (ACE). The Adaptive Critic Element is different than LMS in that it is not supervised learning but reinforcement learning. Supervised learning is characterized by the existence of training data, consisting of input vectors and corresponding desired output vectors. However, when the data necessary for supervised learning is not directly available, reinforcement learning should be used to optimize some performance function. The authors showed the number of trials it takes to train the neuron, or the number of times the experiment needed to be reset, and how long each attempt runs until failure, or the pendulum can not recover, as seen in Figure 2.12. Barto's work can be compared to Michie and Chamber's (1968) boxes method, which breaks the problem down into a piece-wise problem. The neuron does not have much success in controlling the pendulum until fifty trials, where the performance increases dramatically. Anderson (1989) revisits the problem after the derivation of back propagation in 1986. He applied a two-layer neural network, which was not possible in 1983 because back propagation had not yet been developed. He compared his results to a single layer neural network, as seen in Figure 2.13. The two layer neural network works significantly better than the single layer neural network under the same conditions.

Hung and Fernandez (1993) and Pack, Meng, and Kak (1993) did comparative studies on different types of controllers for an inverted pendulum. The first study included PID, Sliding Mode, Expert System, Fuzzy Logic and Neural Network controllers. All five controllers were implemented experimentally and subjected to plant parameter changes. The results of this study can be seen in Table 2.1. The Fuzzy Logic controller worked the best, and the Neural Network controller seemed to work the worst. The second study included PD, Linear Quadratic, Neural Network, Non-linear, and Fuzzy Logic controllers. All five controllers were also implemented experimentally. Pack, Meng, and Kak devised two indices to measure the results of each type of controller: The first of these, called *the effectiveness coefficient*, measures the ratio of the sizes of the actual to the ideal controllable regions of the space spanned by the designated control variables; the second, *the utilization coefficient*, measures the economy, or the ratio of the theoretical minimum to the actual amount of the total control input required to make the system transition from a designated initial state to the goal state. The results can be seen in Table 2.2. The PD controller in this study did the best, with the neural network controller being third in both criterion. The Fuzzy Logic controller did much worse than the rest of the controllers. It is interesting that two very similar studies came up with such completely different results. Obviously, controllers cannot be just implemented; they must be implemented so that the controller uses the strengths of the control method.

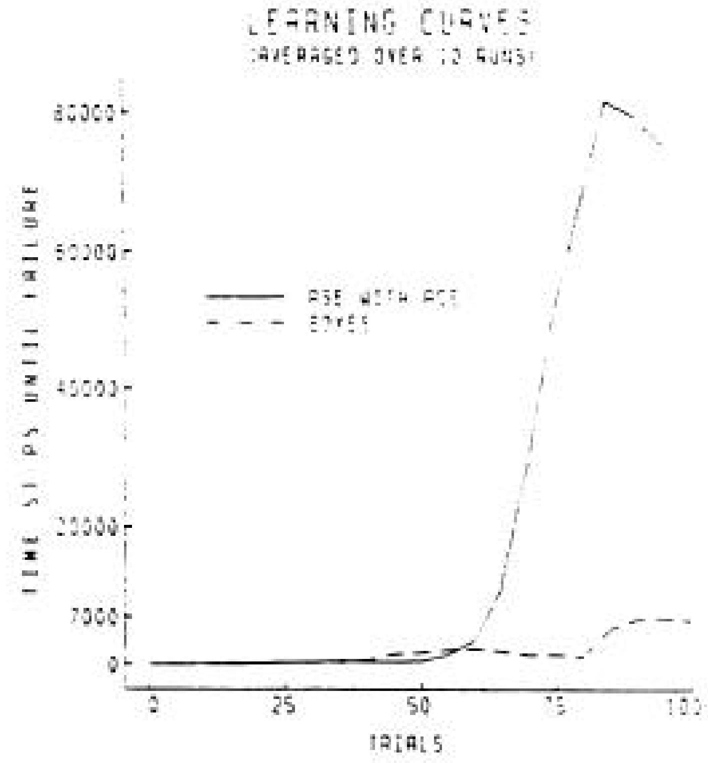

Figure 2.12 ASE & ACE Results Compared to BOXES Method

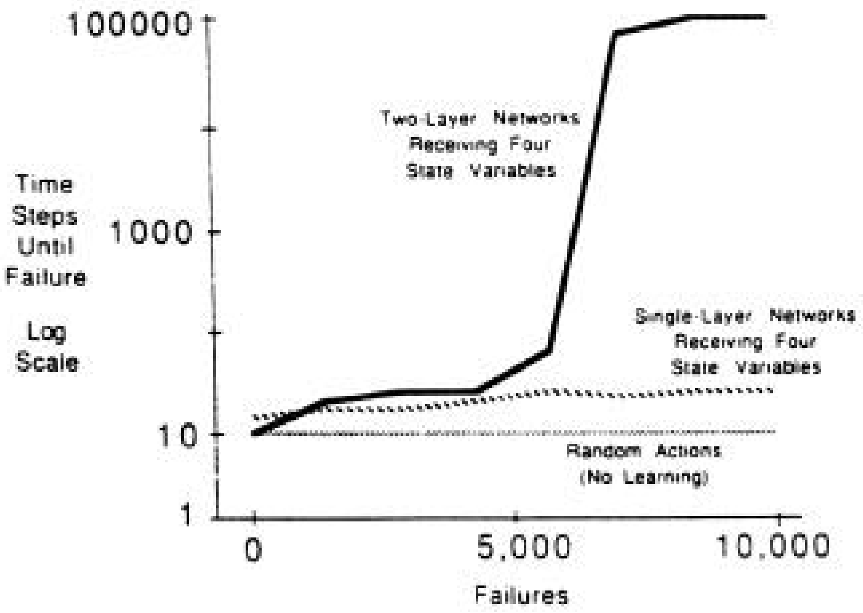

Figure 2.13 Anderson's Results with Multi-Layer Neural Networks

Table 2.1 Hung's Et Al Results

	Before Changes		After Changes	
	Stable Time	Steady Error	Stable Time	Steady Error
PD Controller:	7 seconds	+/- 0.07 rad	3 seconds	+/- 0.02 rad
Fuzzy Logic Controller:	13 seconds	+/- 0.02 rad	13 seconds	+/- 0.02 rad
Sliding Mode Controller:	11 seconds	+/- 0.02 rad	10 seconds	+/- 0.02 rad
Expert System Controller:	9 seconds	+/- 0.02 rad	7 seconds	+/- 0.02 rad
Neural Network Controller:	5 seconds	+/- 0.01 rad	4.5 seconds	+/- 0.01 rad

Table 2.2 Pack's Et Al Results

	Effectiveness Coefficient		Utilization Coefficient	
	Simulation	Experiment	Simulation	Experiment
PD Controller:	0.324	0.022	0.282	0.098
Linear Quadratic Controller:	0.524	0.234	0.564	0.052
Neural Network Controller:	0.785	0.272	0.237	0.122
Nonlinear Controller:	0.248	0.203	1.0	0.129
Fuzzy Logic Controller:	0.349	0.262	0.792	1.0

Tanomaru and Omatu (1991) showed how neural networks can be applied to a control problem in several different manners. They broke up the control configurations into two categories: supervised learning and reinforcement learning. Five different types of supervised learning were tried: supervised control, plant emulation, direct inverse control, indirect inverse control, and direct adaptive control. Supervised control is when the neural network is trained off-line to mimic a controller that works, useful when the existing controller cannot be used in practice (e.g., a human controller.) Plant emulation allows a plant to be identified, and a control scheme can then be designed for the neural network plant model. Direct inverse control is when the neural network is used to cancel out the plant dynamics. This control method works well if the plant's zeros are inside the unit circle. Direct adaptive control is when the neural network is trained on-line using a reference model as the desired output of the plant. Indirect adaptive control is when two neural networks are used to identify and control the plant simultaneously. There was only one scheme using reinforcement learning control that did not determine the controller's output required to produce target plant outputs. The problem is to determine the controller's outputs that improve the performance. This generality is usually obtained at a cost of efficiency when compared with supervised learning methods. All six learning methods can be seen in Figure 2.14. The results of the various control schemes were limited because the methods that use inversing could not be implemented effectively on the inverted pendulum. Overall, no one method stood out above the others.

Phillips and Muller-Dott (1992) used a functional link to enhance the performance of the neural network. A functional link is used to condition the input to the neural network, which makes it easier for the neural network to perform. They compared a single-layer feed-forward neural network and a single-layer feed-forward neural network with a functional link. By limiting the neural network to a single layer, the performance of the normal neural network would be greatly limited to the work done before back propagation. In both cases, two neural networks were employed: one for identification and the other for control. This paper shows that, given equal amounts of computational complexity, the functional link neural network is able to increase performance over the feed-forward neural network.

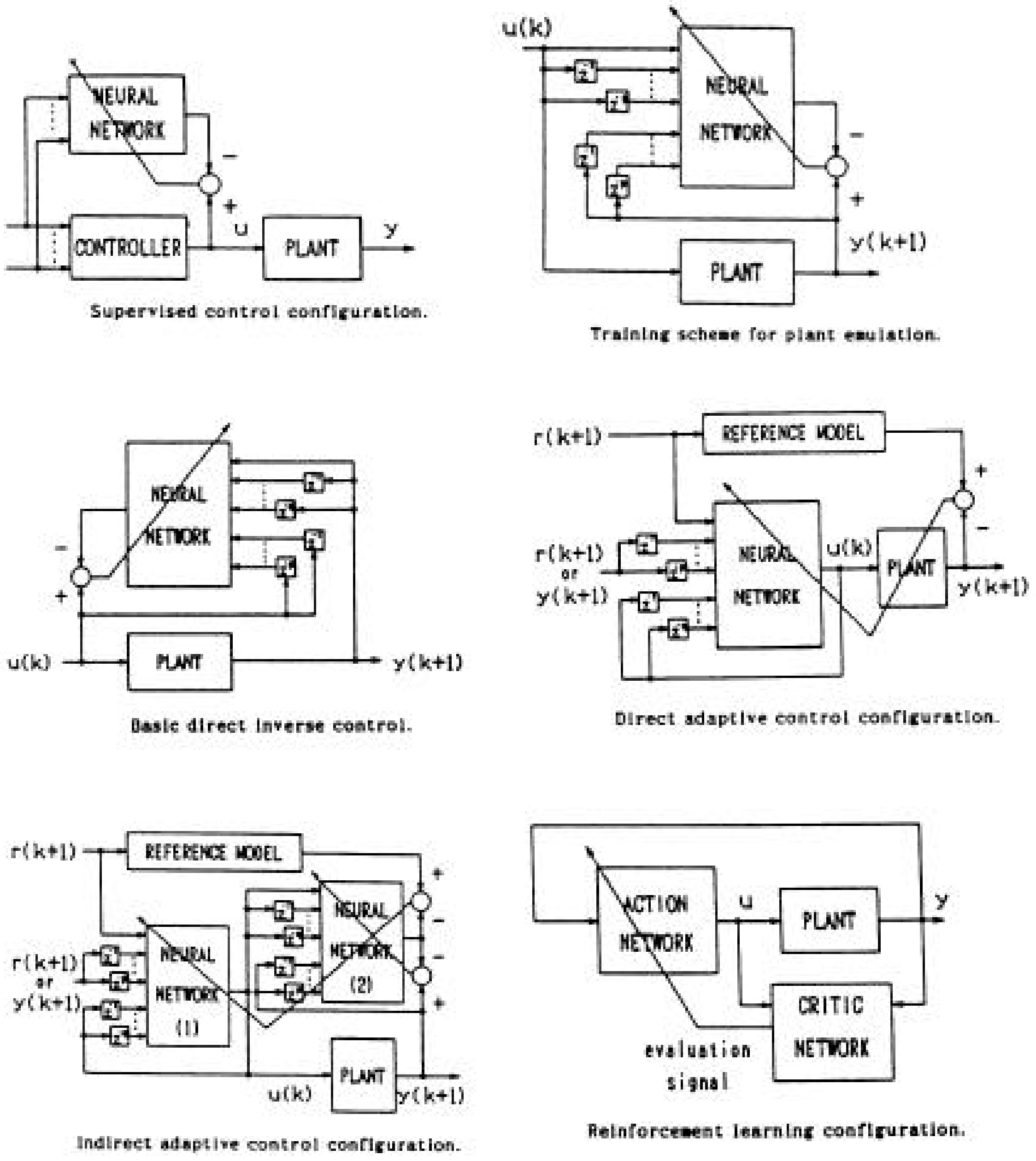

Figure 2.14 Various Learning Schemes from Tanomaru and Omatu (1991)

Huang and Huang (1994) used a gray layer to enhance the performance of the neural network. A gray layer is the output layer to a neural network that conditions the output based on a priori knowledge in order to better control the plant. The difference between a functional link and a gray layer is that the back propagation error must propagate through the gray layer in order to train the weight of the neural network. The impressive thing about their work is that, experimentally, they got a pendulum to swing from the down vertical position at rest to stabilized in the up vertical position. None of the other controllers have shown such a large region of stability.

The inverted pendulum problem has become the benchmark problem with neural network controllers. None of the papers, with the exception of Anderson (1989), showed the amount of time to convergence. Most showed only converged neural network controllers, but only Anderson discussed the number of times that the pendulum had to be reset until the neural network converged. None of the papers discussed the number of times the weights of the neural network were reinitialized. It is very difficult to assess the performance of the neural network controller or any of the other controllers until the whole training process is known.

2.4 FIR Filter In A Closed-Loop System

There are only a few papers that have addressed the need for a new update algorithm for neural networks. A review of work was done on linear plants with FIR filters in the closed-loop. It was done to gain insight into the work being done on linear systems. No work was found to be done in the area of this dissertation. This was very surprising, with the large body of work that has been done in FIR filters. The body of work is mainly discussed in an open-loop configuration. The LMS algorithm, which is used extensively with FIR filters, is not valid inside the closed-loop because, in its derivation, the inputs into the FIR filter are assumed to be independent of its weights.

MacMartin (1994) discussed the differences between feedback and feed-forward techniques. He also gave an alternative representation to the adaptive feed-forward control in a feedback manner. The paper showed the strengths and weaknesses of the two different methods.

Nishimura and Fujita (1994) developed a new active adaptive feedback algorithm based on the Filtered-X configuration. The paper had a simple experiment using a straight duct to increase acoustic damping in a closed sound field. Like the Filtered-X configuration, the error signal was filtered through a model of the plant and feedback to add damping to the system.

Kemal and Bowman (1995) developed an adaptive Filtered-X algorithm strategy to control a combustion chamber. The paper used the Filtered-X LMS update algorithm that has been modified for an IIR controller architecture. This was all applied to acoustically control a combustion chamber.

2.5 Summary

The literature has shown that neural networks have been applied to control nonlinear systems. There has been a large body of paper applying neural networks for controls, but most of the papers do not address initial performance, reinitializing the weights, the randomly-initialized weights, incorporating a priori information, or time to convergence. These issues are very important if an effective controller is to be developed.

Very little work has been done with the neural network in the feedback loop. Even less work has been done with the neural network in series with an existing controller. Initially, the neural network's gain is going to be unknown if the neural network is randomly initialized. The next chapter addresses the problem of randomly initialized neural networks for control with the development of the feed-through neural network, which allows the existing controller to run unencumbered initially.