ACCELERATION STRATEGIES FOR THE BACKPROPAGATION NEURAL NETWORK LEARNING ALGORITHM

by

ZARITA ZAINUDDIN

Thesis submitted in fulfilment of the requirements

for the degree of Doctor of Philosophy

June 2001

ACKNOWLEDGEMENTS

My greatest and ultimate debt and gratitude is due to Allah, the Creator of the heavens and the earth, for giving me the utmost strength, perseverance, determination and courage to complete this thesis.

This thesis would not have been made possible without the assistance of many individuals to whom I am greatly indebted. In particular, I wish to acknowledge the following:

- Prof. D.J. Evans, who has shown his undiminished dedication and outstanding supervision despite being only a field supervisor without whom this thesis is almost impossible to be completed.

- Assoc. Prof. Dr. Fadhil Mohd. Hani for taking the time out of his busy schedule to advise and suggest the work on image recognition and continue supervising in spite of his heavy commitments.

- Dr. Mohd. Yusoff Mashor for bearing a huge responsibility of being the supervisor and proof reading the thesis in the eleventh hour.

- Prof. Rosihan Md. Ali, former Dean of the School of Mathematics, for his accommodating attitude, hence enabling me to find time completing this thesis.

- Ms. Catherine Saw-Paik Lee for her patience, diligence and dedication in preparing the final version of the thesis for printing.

- My brothers and sisters, colleagues in the School of Mathematics and the School of Computer Sciences, and friends who have helped me in one way or another throughout the research study.

- My dearly adored children, Huzaimi, Hafiz, Saarah, Huzaifah, Haseenah and Hanzalah for their love, patience and affection and having endured superbly to the limitations that accompany a working mother cum part-time PhD student.

- And to my beloved husband, Wan Rosli, who has the dual role as a husband and companion, my deepest and heartfelt thanks from the bottom of my heart for the special blend of understanding, encouragement and support and most of all, the continuing love and never ending patience he has always given me.

Finally, this work is dedicated *in loving memory to my late parents, Latifah Hj. Abdullah and Zainuddin Md. Shariff* who have instilled in me the importance of seeking knowledge. May the rewards of this work be extended to both of them.

May ALLAH, the Most Generous bless them all.

Zarita Zainuddin

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

STRATEGI-STRATEGI PEMECUTAN UNTUK ALGORITMA PEMBELAJARAN RANGKAIAN NEURAL PERAMBATAN BALIK

ABSTRAK

Algoritma perambatan balik telah terbukti sebagai salah satu algoritma rangkaian neural yang paling berjaya. Namun demikian, seperti kebanyakan kaedah pengoptimuman yang berasaskan kecerunan, ianya menumpu dengan lambat dan keupayaannya berkurangan bagi tugas-tugas yang lebih besar dan kompleks.

Dalam tesis ini, faktor-faktor yang menguasai kepantasan pembelajaran algoritma perambatan balik diselidik dan dianalisa secara matematik untuk membangunkan strategi-strategi bagi memperbaiki prestasi algoritma pembelajaran rangkaian neural ini. Faktor-faktor ini meliputi pilihan pemberat awal, pilihan fungsi pengaktifan dan nilai sasaran serta dua parameter perambatan, iaitu kadar pembelajaran dan faktor momentum.

Bagi pilihan pemberat awal, satu prosedur pengawalan pemberat telah dibangunkan untuk menentukan suatu titik awal yang mungkin. Daripada titik ini,, satu arah carian dapat dikira. Analisis secara teori faktor momentum telah membawa kepada pembangunan suatu kaedah baru, Faktor Momentum Dinamik, yang menyelaraskan faktor momentum untuk menyesuaikan sendiri secara setempat kepada keadaan fungsi kos. Begitu juga, analisis teori parameter kadar pembelajaran telah memberi pemahaman penting dalam pembangunan dua kaedah yang berhubung dengan parameter ini, iaitu Kadar Pembelajaran Dinamik Kaedah 1 dan 2.

Simulasi komputer yang meluas dan perbandingan prestasi dengan algoritma perambatan tradisional dan dua lagi kaedah berasaskan kecerunan, iaitu kaedah *Conjugate Gradient* dan *Steepest Descent*, telah menunjukkan penumpuan pantas kaedah-kaedah yang dimajukan di dalam tesis ini. Kaedah-kaedah ini telah dilaksanakan dan diuji pada beberapa masalah dan keputusan yang diperolehi menunjukkan bahawa kaedah-kaedah ini berkemampuan untuk menambahkan penumpuan dan seterusnya memecutkan latihan. Pengiraan eksplisit untuk menentukan nilai optimum faktor momentum dan kadar pembelajaran adalah tidak diperlukan serta pengiraan dan beban storan yang berat tidak diperlukan.

Masalah pengecaman muka manusia dipilih untuk membuktikan keberkesanan kaedah-kaedah yang dimajukan pada satu masalah penggunaan dunia sebenar. Selain daripada itu, keupayaan rangkaian yang terlatih terhadap pengumuman dan penolakan diselidiki. Kesan mengubah bilangan nod terlindung terhadap prestasi rangkaian dan juga prestasi rangkaian terhadap imej hingar juga dikaji. Bukti berangka menunjukkan bahawa kaedah pengawalan pemberat dan kaedah-kaedah pecutan yang telah dimajukan adalah kukuh dan mempunyai prestasi purata yang baik dari segi penumpuan, keupayaan pengumuman dan penolakan, serta pengecaman imej hingar.

ABSTRACT

The backpropagation algorithm has proven to be one of the most successful neural network learning algorithms. However, as with many gradient based optimization methods, it converges slowly and it scales up poorly as tasks become larger and more complex.

In this thesis, factors that govern the learning speed of the backpropagation algorithm are investigated and mathematically analyzed in order to develop strategies to improve the performance of this neural network learning algorithm. These factors include the choice of initial weights, the choice of activation function and target values, and the two backpropagation parameters, the learning rate and the momentum factor.

For the choice of initial weights, a weight initialization procedure is developed to determine a feasible initial point from which a search direction can be computed. Theoretical analysis of the momentum factor leads to the development of a new method, the Dynamic Momentum Factor, which dynamically adjusts the momentum factor to adapt itself locally to the cost function landscape. Similarly, theoretical analysis of the learning rate parameter provides important insights into the development of two learning rate related methods, namely, Dynamic Learning Rate Methods 1 and 2.

Extensive computer simulations and performance comparisons with the conventional backpropagation algorithm and two other gradient based methods, the Conjugate Gradient and Steepest descent methods have demonstrated the fast convergence of the proposed methods. The proposed methods have been implemented and tested on

several benchmark problems and the results indicate that these methods are able to provide enhanced convergence and accelerated training. Explicit computations to determine the optimal momentum factor and learning rate values are not needed and no heavy computational and storage burden is necessary.

The human face recognition problem is chosen to demonstrate the effectiveness of the proposed methods on a real world application problem. In addition, the capabilities of the trained networks on generalization and rejection are investigated. The effect of varying the number of hidden nodes on the network's performance and the performance of the network on noisy images is also examined. Numerical evidence shows that the proposed weight initialization and acceleration methods are robust with good average performance in terms of convergence, generalization and rejection capabilities and recognition of noisy images.

CHAPTER 1

INTRODUCTION

The majority of information processing today is carried out by digital computers. This has led to the widely held misperception that information processing is dependent on digital computers. However, if we look at cybernetics and the other disciplines that form the basis of information science, we see that information processing originates with living creatures in their struggle to survive in their environments, and that the information being processed by computers today accounts for only a small part - the automated portion of this. Viewed in this light, we can begin to consider the possibility of information processing devices that differ from conventional computers. In fact, research aimed at realizing a variety of different types of information processing devices is already being carried out, albeit in the shadows of the major successes achieved in the realm of digital computers. One direction that this research is taking is toward the development of an information processing device that mimics the structures and operating principles found in the information processing systems possessed by humans and other living creatures.

Digital computers developed rapidly in and after the late 1940's, and after originally being applied to the field of mathematical computations, have found expanded applications in a variety of areas, to include text (word), symbol, image and voice processing, i.e., pattern information processing, robot control and artificial intelligence. However, the fundamental structure of digital computers is based on the principle of

sequential (serial) processing, which has little if anything in common with the human nervous system.

The human nervous system, it is now known, consists of an extremely large number of nerve cells, or neurons, which operate in parallel to process various types of information. By taking a hint from the structure of the human nervous system, we should be able to build a new type of advanced parallel information processing device.

In addition to the increasingly large volumes of data that we must process as a result of recent developments in sensor technology and the progress of information technology, there is also a growing requirement to simultaneously gather and process huge amounts of data from multiple sensors and other sources. This situation is creating a need in various fields to switch from conventional computers that process information sequentially, to parallel computers equipped with multiple processing elements aligned to operate in parallel to process information.

Besides the social requirements just cited, a number of other factors have been at work during the 1980's to prompt research on new forms of information processing devices. For instance, recent neurophysiological experiments have shed considerable light on the structure of the brain, and even in fields such as cognitive science, which study human information processing processes at the macro level, we are beginning to see proposals for models that call for multiple processing elements aligned to operate in parallel. Research in the fields of mathematical science and physics is also concentrating more on the mathematical analysis of systems comprising multiple elements that interact in complex ways. These factors gave birth to a major research trend aimed at clarifying

the structures and operating principles inherent in the information processing systems of human beings and other animals, and constructing an information processing device based on these structures and operating principles. The term "neurocomputing" is the name used to refer to the information engineering aspects of this research.

1.1 WHAT IS AN ARTIFICIAL NEURAL NETWORK?

Neurocomputing involves processing information by means of changing the states of networks formed by interconnecting extremely large numbers of simple processing elements, which interact with one another by exchanging signals. Networks such as the one just described are called artificial neural networks (ANNs), in the sense that they represent simplified models of natural nerve or neural networks. The basic processing element in the nervous system is the neuron (Figure 1.1). The human brain is composed of about 10 billion of over 100 types of neurons. Tree-like networks of nerve fiber called dendrites are connected to the cell body or soma, where the cell nucleus is located. Extending from the cell body is a single long fiber called the axon, which eventually branches into strands and substrands, and are connected to other neurons through synaptic junctions, or synapses.

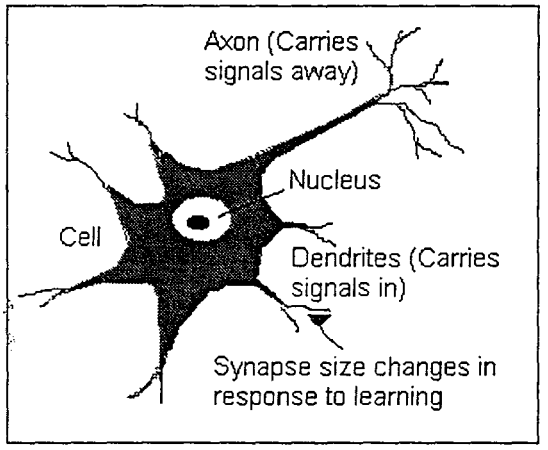

Figure 1.1. A simple neuron cell

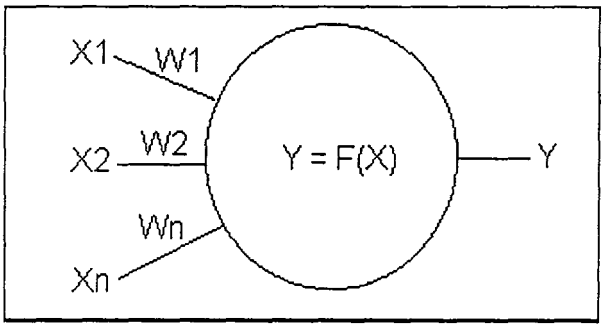

Figure 1.2. A schematic diagram of a neuron

The transmission of signals from one neuron to another at a synapse is a complex chemical process in which specific transmitter substances are released from the sending end of the junction. The effect is to raise or to lower the electrical potential inside the body of the receiving cell. If the potential reaches a threshold, a pulse is sent down the axon - we then say the cell has "fired".

In a simplified mathematical model of the neuron (Figure 1.2), the effects of the synapses are represented by "weights" which modulates the effect of the associated input signals, and the nonlinear characteristics exhibited by neurons is represented by a transfer function which is usually the sigmoid function. The neuron impulse is then computed as the weighted sum of the input signals, transformed by the transfer function. The learning capability of an artificial neuron is achieved by adjusting the weights in accordance to the chosen learning algorithm, usually by a small amount.

Since psychologist Frank Rosenblatt (1958) proposed the "Perceptron", a pattern recognition device with learning capabilities, the hierarchical neural network has been the most widely studied form of network structure. A hierarchical neural network is one that links multiple neurons together hierarchically, as shown in Figure 1.3. The special

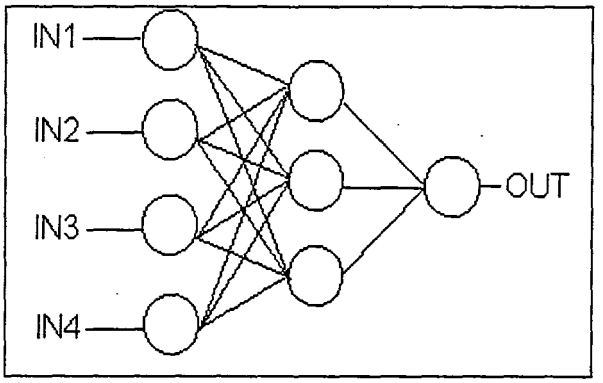

Figure 1.3. A feed forward neural network

characteristic of this type of network is its simple dynamics. That is, when a signal is input into the input layer, it is propagated to the next layer by the interconnections between the neurons. Simple processing is performed on this signal by the neurons of the receiving layer prior to it being propagated on to the next layer. This process is repeated until the signal reaches the output layer completing the processing process for that signal.

The manner in which the various neurons in the intermediary (hidden) layers process the input signal will determine the kind of output signal it becomes (how it is transformed). It is clear then, hierarchical network dynamics are determined by the weight and threshold parameters of each of their units. If input signals can be transformed to the proper output signals by adjusting these values (parameters), then hierarchical networks can be used effectively to perform information processing.

Since it is difficult to accurately determine multiple parameter values, a learning method is employed. This involves creating a network that randomly determines parameter values. This network is then used to carry out input-to-output transformations for actual problems. The correct final parameters are obtained by properly modifying the

parameters in accordance with the errors that the network makes in the process. There are many learning methods that have been proposed. Probably the most representative of these is the error back-propagation learning method proposed by Rumelhart, *et al.* (1986). This learning method has played a major role in the recent neurocomputing boom.

There are multitudes of different types of ANNs. Some of the more popular include the multilayer perceptron which is generally trained with the backpropagation of error algorithm (Rumelhart, *et al.*, 1986), learning vector quantization (Kohonen, 1986; Kohonen, 1990), radial basis function (Broomhead & Lowe; 1988, Moody & Darken, 1989; Renals, 1989; Poggio & Girosi, 1990), Hopfield (Hopfield, 1982), and Kohonen (Kohonen, 1982a; Kohonen, 1982b) to name a few. Some ANNs are classified as feedforward while others are recurrent (i.e., implement feedback) depending on how data is processed through the network. Another way of classifying ANN types is by their method of learning (or training), as some ANNs employ supervised training while others are referred to as unsupervised or self-organizing. Supervised training is analogous to a student guided by an instructor. Unsupervised algorithms essentially perform clustering of the data into similar groups based on the measured attributes or features serving as inputs to the algorithms. This is analogous to a student who derives the lesson totally on his or her own. ANNs can be implemented in software or in specialized hardware.

Although ANNs have been around since the late 1950's, it wasn't until the mid-1980's that algorithms became sophisticated enough for general applications. Today ANNs are being applied to an increasing number of real-world problems of considerable

complexity. They are good pattern recognition engines and robust classifiers, with the ability to generalize in making decisions about imprecise input data. They offer ideal solutions to a variety of classification problems such as speech (Morgan & Bourlard, 1990), character (Le Cun, *et al.*, 1990b) and image recognition (Lawrence, *et al.*, 1997), as well as functional prediction as in economic forecasting (Utan, *et al.*, 1991; Moody, 1995) and system modeling (Uhrig & Guo, 1989) where the physical processes are not understood or are highly complex. ANNs may also be applied to control problems (Talebi, *et al.*, 1998), where the input variables are measurements used to drive an output actuator, and the network learns the control function. The advantage of ANNs lies in their resilience against distortions in the input data and their capability of learning. They are often good at solving problems that are too complex for conventional technologies (eg. problems that do not have an algorithmic solution or for which an algorithmic solution is too complex to be found) and are often well suited to problems that people are good at solving, but for which traditional methods are not.

As indicated above, there are many different ANN architectures designed for different purposes. We will concentrate on the most popular, the multi layer perceptron (MLP). The details of the MLP will be discussed in the next chapter. The most common way of training the MLP is the Back Propagation (BP) algorithm. This involves repeated presentation of inputs in the training set to the network and subsequent adjustment of the weights and biases until the network produces the desired outputs. The popularity of the BP algorithm can be attributed to its simplicity and genericity. The MLP is capable of approximating arbitrary nonlinear mappings, and given a set of examples, the BP algorithm can be employed to learn the mapping at the example points. However, there are a number of practical concerns. The first is the matter of choosing the network

size. The second is the ability of the network to generalize; that is, its ability to produce accurate results on new samples outside the training set. Finally, the time complexity of learning; that is, its ability to learn the desired mapping in a reasonable amount of time. These issues have stirred considerable ongoing research and numerous theories and algorithms have been proposed to this end. One of the principal concerns in BP learning is associated to the last issue above, that is, the complexity of learning. BP learning is too slow for many applications, and it scales up poorly as tasks become larger and more complex. Its slow speed at which the current algorithm learns has been the greatest single obstacle to the widespread use of connectionist learning networks in real-world applications. Viewed in this light, the approach and emphasis taken in this thesis is to investigate factors governing the convergence rate of the BP learning algorithm. The goal is twofold: to develop faster learning algorithms and to contribute to the development of a methodology that will be of value in future studies of this kind.

The problem of finding a set of correct weights for a fixed size network is an inherently difficult problem. This problem has been formulated by Judd (1990) as the loading problem and recently it has been shown that it is NP-complete (Blum & Rivest, 1992; Sima, 1996). This implies that if we have a very large problem, for example, if the dimension of the input space is very large, then it is unlikely that we will be able to determine if a weight solution exists in a reasonable amount of time.

Learning algorithms like BP are based on a gradient search, which is a greedy algorithm that seeks out a local minimum and they may not yield the exact mapping. As with other gradient based methods, BP is very time consuming and learning bigger networks can be intractable (Tesauro, 1987; Tesauro & Janssens, 1988). One way to explain this

sluggishness is to characterize the error surface which is being searched. In the case of a single perceptron with linear activation function the error surface is a quadratic bowl (with a single (global) minimum), making it a relatively agreeable surface to search. For MLPs however, the error surface turns out to be quite harsh (Hush, *et al.*, 1992). These surfaces tend to have a large amount of flatness as well as extreme steepness, but not much in between. It is difficult to determine if the search has even terminate with this type of surface since the transient flat spots "look" much the same as minima, that is, the gradient is very small. Moreover, with this type of surface a gradient search moves very slowly along the flat parts.

Increasing the learning rate to compensate for the sluggishness in these areas can be dangerous because the algorithm may then exhibit instabilities when it reaches the steep parts of the surface. Attempts to speed learning include variations on simple gradient search (Jacobs, 1988; Plaut, *et al.*, 1986; Rumelhart, *et al.*, 1986) line search methods (Hush & Salas, 1988), and second-order methods (Becker & Le Cun, 1988; Watrous, 1987). Although most of these have been somewhat successful, they usually introduce additional parameters which are difficult to determine, must be varied from one problem to the next, and if not chosen properly can actually slow the rate of convergence.

Recently there has been a focus of training feed-forward NNs with optimization techniques that use higher-order information such as the conjugate gradient method (Van Der Smagt, 1994) and Newton's method (Becker & Le Cun, 1989). Both of these methods use the gradient vector (first-order partial derivatives) and the Hessian matrix (second-order partial derivatives) of the cost function to perform the optimization, albeit in different ways. Despite the fact that these techniques possess good convergence

properties, they are computationally more complex. Furthermore, the application of second order methods like the Newton's method to the training of the MLP is hindered by this requirement of having to calculate the Hessian matrix and its inverse which can be computationally expensive.

The main contribution of this thesis consists of efficient BP learning using dynamically optimal momentum factor (MF) and learning rate (LR). These methods exploit the first-order derivative information of the objective function with respect to the MF and LR respectively. Since these approaches do not involve explicit calculation of derivatives in weight space, but rather uses information gathered from the forward and backward propagation, the computational and storage burden scales with the network size exactly like the conventional BP.

1.2 GUIDE TO THESIS

The aim of this thesis is to investigate the factors that govern the learning speed of the BP algorithm in order to develop strategies to improve the performance of this neural network learning algorithm. The investigation includes analyzing the choice of initial weights, the choice of activation function and target values, and the back propagation parameters, in particular, the momentum factor and the learning rate. Three acceleration algorithms to accelerate steepest descent involving the momentum factor and the learning rate respectively are derived and implemented. A comparison is then made between these methods and the conventional BP algorithm. In addition, two classical gradient based training methods – steepest descent gradient search and conjugate gradient are compared with the above algorithms. All programs for the simulations

were performed on the Sequent Balance. In order to evaluate the performance of these algorithms, three benchmark problems are chosen to give a good insight into how these various algorithms will perform on the real-world tasks we eventually want to tackle. The benefit of these algorithms is demonstrated on a real-world application problem, the human face recognition problem using continuous-valued training data that contain random noise.

Basic mathematical and neural network concepts needed to understand multilayer perceptrons trained with the back propagation algorithm are discussed in Chapter 2. Particular emphasis is given to the formulation and derivation of the back propagation algorithm. In Chapter 3, the MLP issues and limitations are discussed, eventually leading to a comprehensive survey of current methods in improving the convergence of the backpropagation algorithm. This includes first and second order optimization techniques applied to supervised learning.

Chapter 4 considers the investigation of the choice of initial weights, choice of target values and activation function. An initialization of weights rule is also proposed here.

In Chapter 5, the derivation of a dynamic momentum factor (DMF) BP algorithm is presented and a momentum factor update rule is proposed. This method is tested and compared with the conventional BP algorithm and two popular deterministic steepest descent based training methods on several benchmark problems. The results presented include both the on-line and batch modes of training.

A similar derivation of a dynamic learning factor (DLR) BP is presented in Chapter 6 leading to a proposal of two dynamic LR update rules. Similarly, these methods are tested and compared with the conventional BP algorithm and deterministic methods on the same benchmark problems as above. Again, the results of both modes of training are considered.

This thesis would not be complete without a real-world application. Therefore, in Chapter 7, in order to evaluate the performance of the BP algorithm using the above proposed algorithms, the human face recognition problem is considered as an example of practical application. A brief survey of the literature on human face recognition is first presented followed by a description of the data set and image database. The numerical results employing the DLR and DMF algorithms against the conventional BP, steepest descent gradient search and conjugate gradient on convergence and generalization capabilities are given. Factors governing the performance of the NN, in particular, network size and training set are also considered and results presented. The performance of the network on noisy images is also investigated. Lastly, a discussion and suggestions for future work pertaining to this research will be given in Chapter 8.

CHAPTER 2

MULTILAYER PERCEPTRONS AND BACKPROPAGATION ALGORITHM

In this chapter, an important architecture of neural networks, namely, multilayer feedforward networks are discussed. Typically, the network consists of a set of sensory units (source nodes) that constitute the 'input layer', one or more hidden layers of computational nodes, and an output layer of computational nodes. The input signal propagates through the network in a forward direction, on a layer-by-layer basis. These ANNs are commonly referred to as multilayer perceptrons (MLPs), which represent a generalization of the single-layer perceptron.

Multilayer perceptrons have been applied successfully to solve some difficult and diverse problems by training them in a supervised manner with a highly popular algorithm known as the error backpropagation (BP) algorithm. This algorithm is based on the error-correction learning rule and hence, it may be viewed as a generalization of the least-mean-square algorithm for the special case of a single linear neuron model. A multilayer perceptron has three distinctive characteristics:

1. The model of each neuron in the network includes a nonlinearity at the output end. In contrast to the hard-limiting used in Rosenblatt's perceptron, the nonlinearity here is smooth that is, differentiable everywhere. A commonly used form of nonlinearity is the sigmoidal nonlinearity defined by the logistic function

$$y_j = \frac{1}{1+\exp(-v_j)}$$

where v_j is the net internal activity level of neuron j and y_j is the output of the

neuron. It is these nonlinearities that make the input-output relation of the network different from a single-layer perceptron.

2. The network comprises of one or more layers of hidden neurons that are not part of the input or output of the network. These hidden neurons provide the network the ability to learn complex tasks by successively extracting meaningful features from the input patterns.

3. The network displays a high degree of connectivity which is determined by the synapses of the network. A change in the connectivity of the network requires a change in the population of synaptic connections or their weights.

These characteristics combined with the ability to learn from experience endows the multilayer perceptron its computing power. However, these same characteristics are also responsible for the deficiencies in the behavior of the network. First, the presence of a distributed form of nonlinearity and the high connectivity of the network make the theoretical analysis of a multilayer perceptron difficult to undertake. Second, the use of hidden neurons makes the learning process harder to visualize. The learning process must decide which features of the input pattern should be represented by the hidden neurons. The learning process is made difficult because the search has to be conducted in a much larger space of possible functions, and a choice has to be made between alternative representations of the input pattern (Hinton & Nowlan, 1987).

Research interest in multilayer feedforward networks dates back to the pioneering work of Rosenblatt (1962) on perceptrons and that of Widrow and his students on Madalines (Widrow, 1962). The development of the back-propagation algorithm represents a

'landmark' in neural networks in that it provides a computationally efficient method for the training of multilayer perceptrons. Although it cannot be claimed that the back-propagation algorithm can provide a solution for all solvable problems, it is fair to say that it has put to rest the pessimism about learning in multilayer machines that may have been inferred from the book by Minsky and Papert (1969).

This chapter begins with a detailed derivation of the backpropagation algorithm in Section 2.1. Using the chain rule of calculus, the essential calculations in focus are those that will be employed in the derivation of the three acceleration methods presented in this thesis. Then, Section 2.2 addresses some basic issues relating to backpropagation learning covering the rate of learning, pattern and batch modes of learning and the issue of initialization which plays a very important role in successful applications of the backpropagation algorithm. A summary of the backpropagation algorithm outlining the principal steps is presented in Section 2.3.

2.1 DERIVATION OF THE BACK-PROPAGATION LEARNING ALGORITHM

Figure 2.1 shows the architectural graph of a multilayer perceptron with two hidden layers. A neuron in a layer of the network is connected to all the nodes in the previous layer. Signal flow through the network progresses in a forward direction from left to right and on a layer by layer basis. The input nodes constitute the first layer and the

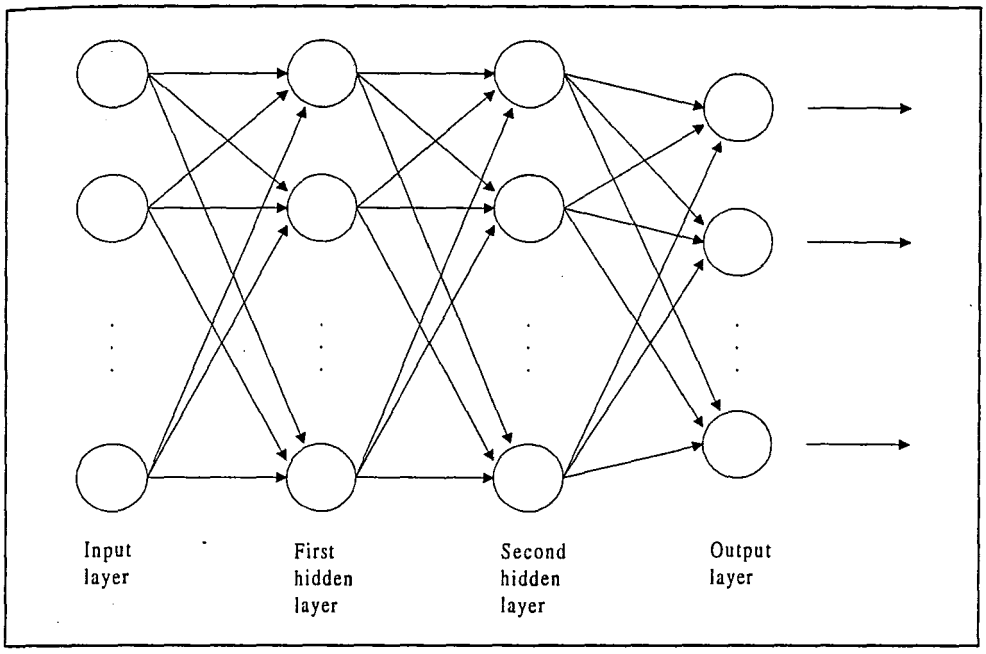

Figure 2.1. Architectural graph of a multilayer perceptron with two hidden layers.

output nodes constitute the output layer while the remaining nodes constitute hidden layers of the network. The input vector is presented to the input layer and the signals are propagated forward to the first hidden layer; the resulting output of the first hidden layer are in turn applied to the next hidden layer and so on for the rest of the network. Each hidden or output neuron of a multilayer perceptron is designed to perform two computations:

1. The computation of the net internal activity level produced at the input of neuron j which is expressed as the summation of input values multiplied by their corresponding weights.

2. The computation of the activation of neuron j which is computed by passing the net internal activity through a continuously differentiable nonlinear activation function. Usually, the sigmoid logistic function is used.

The error signal $e_j(n)$ at the output of neuron j at iteration n (i.e., presentation of the nth training pattern) is defined by

$$e_j(n) = d_j(n) - y_j(n) \tag{2.1}$$

where $d_j(n)$ and $y_j(n)$ is the desired and the actual response of neuron j at iteration n respectively.

Hence, the instantaneous value $\xi(n)$ of the sum of squared errors over all neurons is written as

$$\xi(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n) \tag{2.2}$$

where C includes all the neurons in the output layer of the network. The average squared error over the total number of patterns N is given by

$$\xi_{av} = \frac{1}{N}\sum_{n=1}^{N} \xi(n). \tag{2.3}$$

The objective of the learning process is to adjust the free parameters (i.e., synaptic weights and thresholds) of the network so as to minimize ξ_{av}. The gradient descent method is used to perform the minimization where the weights are updated (adjusted) in accordance with the respective errors computed for each pattern to the network. Taking the arithmetic average of these independent weight changes over the training set would therefore give an estimate of the true change that would result from modifying the weights based on minimizing the cost function ξ_{av} over the training set. Figure 2.2 depicts neuron j being fed by a set of function signals produced by neurons in the previous layer.

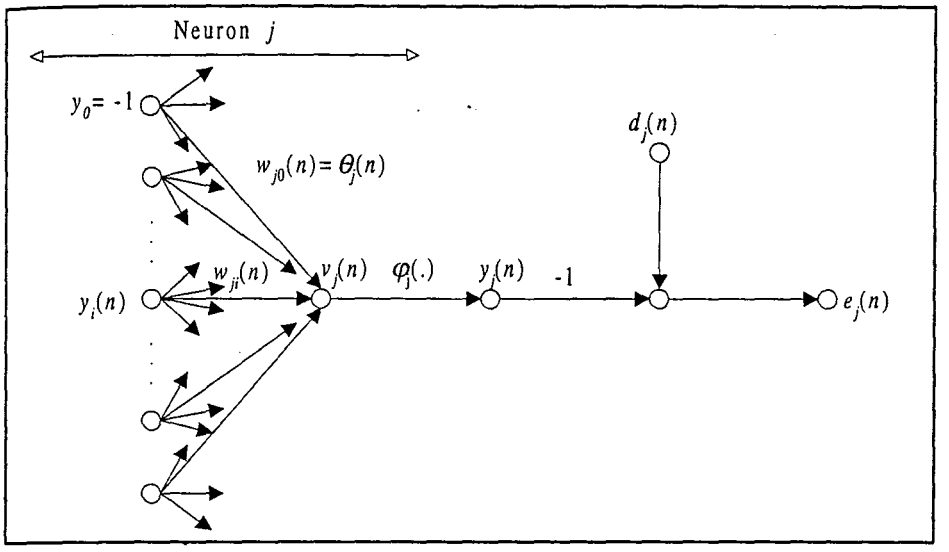

Figure 2.2. Signal-flow graph highlighting the details of output neuron j.

The net internal actively level $v_j(n)$ produced at the input of neuron j is

$$v_j(n) = \sum_{i=0}^{p} w_{ji}(n) y_i(n) \tag{2.4}$$

where p is the total number of inputs (excluding the threshold) applied to neuron j and $w_{ji}(n)$ denotes the synaptic weight connecting the output of neuron i to the input of neuron j at iteration n. The synaptic weight w_{j0} (corresponding to the fixed input $y_0 = -1$) equals the threshold or bias θ_j applied to neuron j.

Hence, the function signal $y_j(n)$ appearing at the output of neuron j at iteration n is

$$y_j(n) = \varphi_j(v_j(n)) \tag{2.5}$$

where $\varphi_j(.)$ is the activation function describing the input-output functional relationship of the nonlinearity associated with neuron j.

The correction $\Delta w_{ji}(n)$ applied to the synaptic weight $w_{ji}(n)$ is proportional to the instantaneous gradient $\partial\xi(n)/\partial w_{ji}(n)$. Employing the chain rule, the gradient can be expressed as follows:

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = \frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}. \tag{2.6}$$

The gradient $\partial\xi(n)/\partial w_{ji}(n)$ determines the direction of search in weight space for the synaptic weight w_{ji}.

Differentiating both sides of Eq .(2.2) with respect to $e_j(n)$, we get

$$\frac{\partial \xi(n)}{\partial e_j(n)} = e_j(n) \tag{2.7}$$

and differentiating both sides of Eq. (2.1) with respect to $y_j(n)$, we get

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1. \tag{2.8}$$

Differentiating Eq. (2.5) with respect to $v_j(n)$, we get

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)). \tag{2.9}$$

Finally, differentiating Eq. (2.4) with respect to $w_{ji}(n)$ yields

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n). \tag{2.10}$$

Hence applying Eqs. (2.7) to (2.10) in (2.6) yields

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi_j'(v_j(n))y_i(n). \tag{2.11}$$

According to the delta rule, the correction $\Delta w_{ji}(n)$ applied to $w_{ji}(n)$ is defined by

$$\Delta w_{ji}(n) = -\eta \frac{\partial \xi(n)}{\partial w_{ji}(n)} \tag{2.12}$$

where η is the learning rate parameter.

Using Eq. (2.11) in (2.12) yields

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n) \tag{2.13}$$

where the local gradient $\delta_j(n)$ is itself defined by

$$\begin{aligned} \delta_j(n) &= -\frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \\ &= e_j(n) \varphi_j'(v_j(n)). \end{aligned} \tag{2.14}$$

The local gradient points to required changes in synaptic weights.

There are two distinct cases associated to the error signal $e_j(n)$ at the output of neuron j:

Case I. Neuron j is an Output Node.

When neuron j is located in the output layer of the network, it would be supplied with a desired response of its own. Hence,

$$e_j(n) = d_j(n) - y_j(n). \tag{2.15}$$

Case II. Neuron j is a Hidden Node.

When neuron j is located in a hidden layer, there is no specified desired response for that neuron. Accordingly, the error signal for a hidden neuron is determined recursively in terms of the error signals of all the neurons to which that hidden neuron is directly connected.

Figure 2.3 depicts neuron j as a hidden node of the network. From Eq. (2.14), the local gradient $\delta_j(n)$ for hidden neuron j can also be written as

$$\begin{aligned} \delta_j(n) &= -\frac{\partial \xi(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \\ &= -\frac{\partial \xi(n)}{\partial y_j(n)} \varphi_j'(v_j(n)). \end{aligned} \tag{2.16}$$

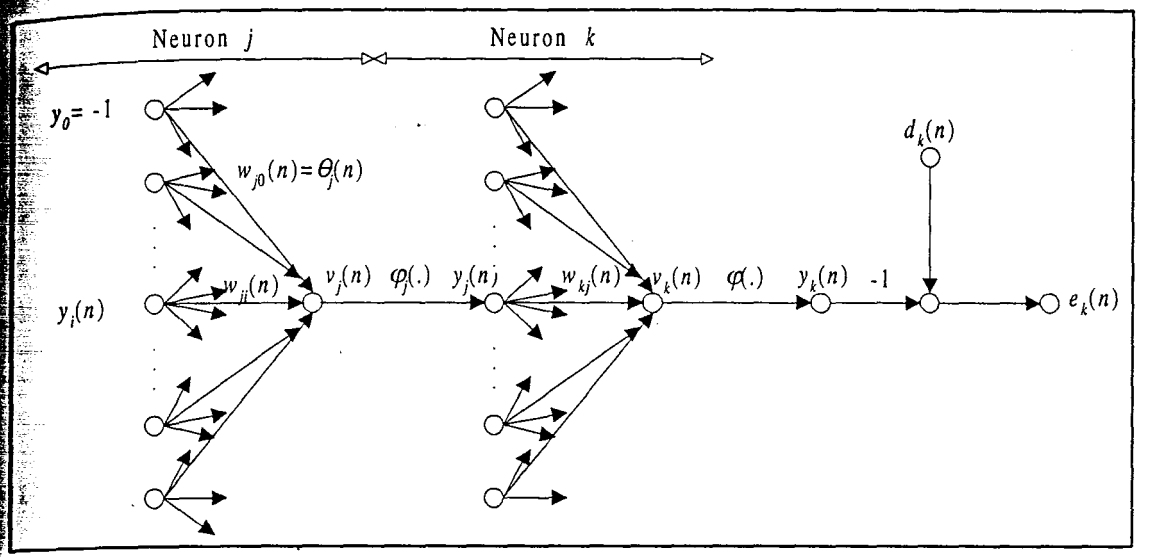

Figure 2.3. Signal-flow graph highlighting the details of output neuron k connected to hidden neuron j.

The partial derivative $\frac{\partial \xi(n)}{\partial y_j(n)}$ can be calculated as follows. From Fig. 2.3, we see that

$$\xi(n) = \frac{1}{2}\sum_{k \in C} e_k^2(n), \qquad \text{neuron } k \text{ is an output node.} \tag{2.17}$$

Differentiating Eq. (2.17) with respect to the function signal $y_j(n)$, we get

$$\frac{\partial \xi(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)} \tag{2.18}$$

Using the chain rule for the partial derivative $\frac{\partial e_k(n)}{\partial y_j(n)}$, we can rewrite Eq.(2.18) in the equivalent form

$$\frac{\partial \xi(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} . \tag{2.19}$$

From Fig. 2.3,

$$\begin{aligned} e_k(n) &= d_k(n) - y_k(n) \\ &= d_k(n) - \varphi_k(v_k(n)) \qquad \text{neuron } k \text{ is an output node.} \end{aligned} \tag{2.20}$$

Hence,

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi_k'(v_k(n)). \tag{2.21}$$

Again, from Fig. 2.3, the net internal activity level for neuron k is given by

$$v_k(n) = \sum_{j=0}^{q} w_{kj}(n) y_j(n) \tag{2.22}$$

where q is the total number of inputs (excluding the threshold) applied to neuron k. Differentiating Eq. (2.22) with respect to $y_j(n)$ yields

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n). \tag{2.23}$$

Thus, using Eqs. (2.21) and (2.23) in (2.19), we get the desired partial derivative:

$$\begin{aligned} \frac{\partial \xi(n)}{\partial y_j(n)} &= -\sum_k e_k(n)\varphi_k'(v_k(n))w_{kj}(n) \\ &= -\sum_k \delta_k(n)w_{kj}(n). \end{aligned} \tag{2.24}$$

Finally, using Eq.(2.24) in (2.16), we get the local gradient $\delta_j(n)$ for hidden neuron j, as follows:

$$\delta_j(n) = \varphi_j'(v_j(n))\sum_k \delta_k(n)w_{kj}(n), \qquad \text{neuron } j \text{ is hidden.} \tag{2.25}$$

2.1.1 The Two Passes of Computation

The back propagation algorithm consists of two distinct passes of computation, the forward pass and backward pass. In the forward pass the synaptic weights remain unaltered throughout the network and the function signals are computed on a neuron-by-neuron basis. The output of neuron j, $y_j(n)$, is computed as

$$y_j(n) = \varphi(v_j(n)) \tag{2.26}$$

where $v_j(n)$ is the net internal activity level of neuron j defined by

$$v_j(n) = \sum_{i=0}^{p} w_{ji}(n) y_i(n). \tag{2.27}$$

If neuron j is the first hidden layer, then

$$y_i(n) = x_i(n) \tag{2.28}$$

where $x_i(n)$ is the ith element of the input vector. On the other hand, if neuron j is in the output layer,

$$y_j(n)=o_j(n) \tag{2.29}$$

where $o_j(n)$ is the jth element of the output vector. This output is compared to the desired response $d_j(n)$, obtaining the error signal $e_j(n)$ for the jth output neuron. Thus the forward phase of computation begins at the first hidden layer by presenting it with the input vector, and terminates at the output layer by computing the error signal for each neuron of this layer.

On the other hand, the backward pass, starts at the output layer by passing the error signals leftward through the network, layer by layer, and recursively computing the δ for each neuron. T he synaptic weights are changed according to the delta rule:

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n). \tag{2.30}$$

For a neuron in the output layer, $\delta_j(n)$ is given by

$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n)). \tag{2.31}$$

Given the delta values (δ's) for the neurons of the output layer, the delta values for all he neurons in the penultimate layer is computed using

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \tag{2.32}$$

nd changes to the weights of all connections feeding into it are made.

This recursive computation is continued on a layer by layer basis, by propagating the changes to all the synaptic weights.

2.1.2 Sigmoidal Nonlinearity

In computing the δ for each neuron, the derivative of the activation $\varphi(.)$ is required. Hence this function has to be continuous. A continuously differentiable nonlinear activation function commonly used in multilayer perceptrons is the sigmoidal nonlinearity given by

$$\begin{aligned} y_j(n) &= \varphi_j(v_j(n)) \\ &= \frac{1}{1+\exp(-v_j(n))}, \qquad -\infty < v_j(n) < \infty \end{aligned} \tag{2.33}$$

where $v_j(n)$ is the net internal activity of neuron j. The range of this nonlinearity lies inside the interval $0 \le y_j \le 1$. Another type of sigmoidal nonlinearity is the *hyperbolic tangent*, which is antisymmetric with respect to the origin and for which the range lies inside the interval $-1 \le y_j \le +1$. This form of nonlinearity will be discussed further in Chapter 4 to pave the way for the discussion on the employment of an alternative activation function in the training of MLPs.

Differentiating both sides of Eq. (2.33) with respect to $v_j(n)$, we get

$$\begin{aligned} \frac{\partial y_j(n)}{\partial v_j(n)} &= \varphi_j'(v_j(n)) \\ &= \frac{\exp(-v_j(n))}{[1+\exp(-v_j(n))]^2}. \end{aligned} \tag{2.34}$$