CHAPTER 2

LITERATURE SURVEY

"The real art of discovery consists not in finding new lands but in seeing with new eyes."
- French novelist Marcel Proust

2.1 INTRODUCTION

Improving the training speed of NN on large dataset is still a focus of research attention. Many research works have been explored on the improvement of learning efficiency of BPN algorithms in different enhancement by estimating optimal initial weight, adaptive learning rate and momentum, and using second order algorithm to achieve faster learning and maintaining generalization. This chapter provides background material for the remainder of the thesis.

2.2 NEURAL NETWORK

NN is a simplified learning model derived from the observed structure of the biological nervous system. The aim of NN is to mimic the human ability to adapt to changing circumstances and the current environment. This depends heavily on being able to learn from events that have happened in the past and to be able to apply this to future situations. It is composed of a large number of highly interconnected simple processing elements (neurons) working in unison to solve a specific problem that is further explained below.

2.2.1 Basic Concepts

The Neuron

The formal definition to the first model of an artificial neuron called McCulloch-Pitts model was formulated by Warren McCulloch and
Walter Pitts in 1943 (McCulloch and Pitts 1943). This model does not exhibit any learning but just serves as a basic building block of every artificial neural network. The neuron is also called as ‘nodes’ or ‘units’. These neurons are connected together to form a network which mimics a biological neural network. Extracting from the human brain model, an artificial neuron is made up of the activation function $\varphi(x)$ (like cell body) and transfer function (like synapses).

The neuron has two modes of operation: the training mode and the testing mode (Kevin 1997). In the training mode, the neuron can be trained to fire (or not) for particular input patterns. In the testing mode, when a taught input pattern is detected at the input, its associated output becomes the current output.

The basic model of a neuron that forms the basis of NNs is illustrated in Figure 2.1.

**Figure 2.1 Simple Artificial Neuron Model**

The basic artificial neuron has a set of $n$ inputs. Each input to the neuron is labeled as $x_1, x_2, \ldots, x_n$ collectively referred as the input vector $X$. 
Each input $x_i$ is weighted by the corresponding weight factor $W_{ij}, W_{2j}, \ldots, W_{nj}$ which represents synaptic strength. The two fundamental operations in a single neuron can be described as:

- **Calculate the Net Input, $net_j$**
  
  Each input $x_1, x_2, \ldots, x_n$ is multiplied by the corresponding weight factor $W_{ij}, W_{2j}, \ldots, W_{nj}$. Weighted inputs are applied to the summation block, labeled as $\sum$. This summation function sums all the weighted inputs and produces an output, $net_j$.

  \[
  net_j = W_{0j}x_0 + W_{1j}x_1 + \cdots + W_{nj}x_n \tag{2.1}
  \]

  \[
  net_j = \sum_{i=1}^{n} W_{ij}x_i + W_{0j} \tag{2.2}
  \]

- **Calculate the final output, $y_j$**

  The activation value is generated by passing the Net Input, $net_j$, to a nonlinear filter $\varphi$ called Activation function, or Transfer function, or Squash function. Except output units, the activation value is fed to one or more other units.

  \[
  y_j = \varphi (net_j) \tag{2.3}
  \]

**Activation Function**

Activation function is a nonlinear function that calculates the output from the received net input signal and bias of the neuron (Engelbrecht 2007). As seen from the artificial neuron model given in Figure 2.1 and its equation
the major unknown variable of the model is its Activation function, $\varphi$. Activation function is chosen based on the problem to be solved. The most frequently used activation functions are enumerated below:

1. **Step Function**

   It is also known as *threshold function* as given in Figure 2.2. It is a binary function that has only two possible output values either zero or one. That means, if input value meets specific threshold ($\theta_i$), the output value results in one value and if specific threshold is not met, that results in different output value. This kind of function is often used in single layer networks.

   The function $\varphi$ is defined as

   $$
   \varphi(x) = \begin{cases} 
   1, & \text{if } x \geq \theta_i \\
   0, & \text{if } x < \theta_i 
   \end{cases}
   $$

   \text{(2.4)}

   ![Figure 2.2 Step Function](image-url)
2. **Identity Function**

Identity function (given in Figure 2.3) is a function that always returns the same value that is used as its argument. It is probably the least commonly used activation function.

The function $\varphi$ is defined as

$$\varphi(x) = x \quad (2.5)$$

![Figure 2.3 Identity Function](image)

3. **Sigmoidal Function**

The sigmoid activation function (given in Figure 2.4) is a common activation function for neural networks. The sigmoid function will only produce positive numbers between 0 and 1. The sigmoid activation function is the most useful for training data that is also between 0 and 1.

The function $\varphi$ is defined as

$$\varphi(x) = \frac{1}{1 + e^{-\beta x}} \quad (2.6)$$

where $\beta$ is a gain parameter.
4. Hyperbolic Tangent Function

The Hyperbolic Tangent function (given in Figure 2.5) is the trigonometric function which will produce numbers between -1 and 1.

The function \( \varphi \) is defined as

\[
\varphi(x) = \tanh(x)
\]  

(2.7)
Activation function not only determines the value of the output, but also manifests the total firing strength of the node (Engelbrecht 2007). Therefore, the activation function should not be arbitrarily selected because it has a huge impact on the NN performance.

In classification problems, activation functions for the hidden units are needed to introduce differentiable non-linearity into the networks. The reason is that a composition of linear functions is again a linear function and also to ensure that the output values stay within the range of the different classes. However, it is the non-linearity that makes multi-layer networks so powerful. The sigmoid function is the most common choice (Ben and Patrick 1996).

2.2.2 Types of Architecture

The NN architecture refers to the way nodes are arranged in the network. The various type of NN architectures are Feedforward Neural Network (FNN) and Recurrent Neural Network (RNN) (Haykin 2009).

Feedforward Neural Network

A feed-forward network is a non-recurrent network which consists of input, output and hidden layers usually interconnected in a feed-forward way. Each layer comprises of one or more nodes. Typically, the layers in this network are said to be fully interconnected if every node in each layer of the network is connected to every other node in the adjacent forward layer. Feedforward means that the data flows in only one direction i.e. from input to output layer (forward).

Single Layer Feedforward Neural Network

Single Layer Feedforward Network is the simplest neural network which consists of an input layer and an output layer. In this network, the input
layer just presents a given input pattern to the network as there is no computational function. So it is not counted for number of layers, i.e., single layer refers to the output layer of computation nodes as shown in Figure 2.6. It is used for the classification of patterns that are linearly separable.

![Figure 2.6: Single Layer Feedforward Neural Network](image)

Multi Layer Feedforward Neural Network

Multi Layer Feedforward Neural Network (given in Figure 2.7) consists of an input layer of source nodes, one or more hidden layers of computation nodes and an output layer of nodes. The number of layers and the number of hidden nodes in each hidden layer are user design parameters. The most popular class of Multilayer Feedforward Neural Network is Multilayer Perceptron (MLP) (Werbos 1974; Rumelhart et al 1986).
The McCulloch-Pitts Neural Model is a quite simple model which only generates a binary output whilst the weight and threshold values are fixed with no learning or adaptation (Minsky and Papert 1969). There is no existing mechanism to compare the desired output and the actual output and therefore no weight adjustment or learning can take place. The need for feedback was quickly recognized, particularly at a time when feedback control theory was moving in the direction of adaptive control. Hence, it made Rosenblatt to propose the new perceptron model, with learning and adaptation using perceptron learning law (Rosenblatt 1958). He proved that his learning law will always converge to the correct network weights, if weights exist that can solve the linearly separable classification problems. In this model, learning was simple. The perceptron could even learn when initialized with random values for its weights and biases.
Today the perceptron is still viewed as an important network. It remains a fast and reliable network for the class of problems that it is able to solve. In 1969, Minsky and Papert showed mathematically that single layer perceptrons could not do some basic pattern recognition operations, i.e., single layer perceptron could handle only linearly separable classes and it was shown that a MLP could be used to perform any nonlinear pattern classification task. Thus, MLP is the most popular and most widely used model for pattern classification that are not linearly separable (Minsky and Papert 1969).

Recurrent Neural Network

Recurrent Neural Network (RNN) (given in Figure 2.8) is a network whose nodes contain at least one feedback loop connection so that activation can flow round the loop. The feedback may be a self feedback, i.e., the output of nodes is fed back to its own input. Sometimes, feedback loops create an internal state of the network, which results in nonlinear dynamic behaviour.

Figure 2.8: Recurrent Neural Network
2.3 LEARNING METHODS

Since the ability to learn is a fundamental characteristic of intelligence, training the NN architecture is essential. Once an NN has been structured for a particular application, that network is ready for training. To start this process, the initial weights are chosen randomly. Then the training or learning begins.

At its most basic level, training can be described as an iterative process by which it acquires and represents the knowledge of the environment autonomously with the help of learning methods. All learning methods used for training the NN can be classified into three major categories:

i. Supervised Learning
ii. Unsupervised Learning
iii. Reinforced Learning

2.3.1 Supervised Learning

In Supervised Learning, the objective is to learn from examples, i.e., generalize from training samples by observing a set of inputs and the desired output (Tsoukalas and Uhrig 1997; Russell and Norvig 2003). Here, a supervisor is assumed to be present during the training process. If the difference between the desired output and the network's actual output is not less than the threshold value, an error is produced, that is used for weight updation - so that the desired output moves towards the actual output.

2.3.2 Unsupervised Learning

In Unsupervised Learning, the desired output is not presented to the network, i.e., there is no ‘supervisor’ to present the desired patterns (Tsoukalas and Uhrig 1997; Russell and Norvig 2003). The core objective is
to self-organize data into categories; hence it is referred as self-organized learning. When inputs are fed into the network, the output should produce the right category after training.

### 2.3.3 Reinforcement Learning

In Reinforcement Learning, the learner does not explicitly know the input-output instances, but it receives some form of feedback from its environment to indicate whether the computed output is correct or incorrect (Tsoukalas and Uhrig 1997; Russell and Norvig 2003). The learner thus adapts its parameters based on the feedback. Hence a reward is given for the correct answer computed and a penalty for the wrong answer. But, it is not a popular learning method.

Among these methods, Supervised Learning is a well-known method, typically used for solving pattern classification problems, using MLP networks.

### 2.4 THE MULTILAYER PERCEPTRON

The MLP is one of the most popular and most frequently used types of NN models due to its clear architecture and comparably simple algorithm (Minsky and Papert 1969). Moreover, MLP with a single hidden layer has been explored as the best neural network architecture which is suitable for a large variety of nonlinear classification application due to its capability to approximate any nonlinear function mapping (Hornik et al 1989; Mehra and Wah 1992).

The MLP consists of a set of nodes organized in a hierarchical structure forming the input layer of nodes, one or more intermediary or hidden layers of computational nodes and an output layer of nodes that
calculate the output of the network as shown in Figure 2.9. The consecutive layers are fully connected in a feedforward way. The connections between the nodes of adjacent layers propagate the signal through the network layer-by-layer. The MLP is said to be fully connected if every node in a given layer is connected to every node in the subsequent layer.

Figure 2.9 Structure of the Multilayer Perceptron

Assume that the network shown in Figure 2.9 contains \( n \) input nodes in the input layer, \( p \) hidden nodes in the hidden layer and \( m \) output nodes in the output layer. This NN would be known as \( n-p-m \) network. Since the above network is highly interconnected, the nodes in each layer are connected with all the nodes in the next layer.

Let \( p \) represent the number of input patterns in the training dataset. The input matrix \( X \), of size \( p \times n \) is presented to the network. The number of nodes in the input layer is equivalent to the number of columns in the input
matrix $X$. Each row in $X$ is considered to be a real-valued vector $x_i \in \mathbb{R}^{a+1}$ where $1 \leq i \leq n$. The summed real-valued vector generated from the hidden layer is represented as $z_i \in \mathbb{R}^{p+1}$ where $1 \leq i \leq p$. The estimated output real-valued vector generated from the network is denoted as $y_i \in \mathbb{R}^m$ where $1 \leq i \leq m$. Let $v_{ij}$ be the $n \times p$ weight matrix which contains input-to-hidden weight coefficient for the link from the input node $i$ to the hidden node $j$ and $v_{oj}$ be the bias weight to the hidden node $j$. Let $w_{jk}$ be the $p \times m$ weight matrix which contains hidden-to-output weight coefficient for the link from the hidden node $j$ to the output node $k$ and $w_{ok}$ be the bias weight to the output node $k$.

Typically MLP networks with single hidden layers trained with the backpropagation training algorithm are the most popular neural networks applied for any supervised pattern classification tasks (Rumelhart et al 1986).

2.5 THE BPN ALGORITHM

The BPN algorithm is a systematic method used for supervised training of MLP. It works by approximating the non-linear relationship between the input and the output by adjusting the weight values internally (Rumelhart et al 1986; Hornik et al 1989). It can further be generalized for the input pattern that is not included in the training dataset.

2.5.1 Working Principle

BPN algorithm is an iterative Gradient Descent (GD) training algorithm designed to estimate the coefficients of weight matrices that minimizes the total Root Mean Squared Error (RMSE) or Mean Squared Error (MSE) of the network by moving down the gradient of the error curve, i.e., in direction of the steepest decrease of the error (Haykin, 2009; Alsmadi et al., 2009). The RMSE is defined between the desired output and the actual output summed over all the training pattern input to the network.
\[
\text{RMSE} = \frac{1}{P} \sum_{p=1}^{P} E^p
\]  \hspace{1cm} (2.8)

\(E^p\) is calculated using the following formula

\[
E^p = \frac{1}{2} \sum_{k=1}^{m} (t_k - y_k)^2
\]  \hspace{1cm} (2.9)

where \(P\) is the total number of training sample patterns, \(m\) is the number of nodes in the output layer, \(t_k^p\) is the target output of the \(k^{th}\) node for the \(p^{th}\) sample pattern and \(y_k^p\) is the actual output of the \(k^{th}\) node estimated by the network for the \(p^{th}\) sample pattern.

The BPN algorithm comprises two passes through the network: forward pass (feedforward mode) and backward pass (backpropagation mode).

**Forward Pass**

During the forward pass, the network reads an input training vector which is fed into the input layer. The input layer does not do any computation and simply disseminate the input training vector from the input layer towards the subsequent layers.

**Hidden Layer Activation net value**

Each hidden node \(z_j\) \((j=1,2,\ldots,p)\) input is aggregated by multiplying input values with the corresponding weights

\[
z_{inj} = v_{oj} + \sum_{i=1}^{n} x_i \cdot v_{ij}
\]  \hspace{1cm} (2.10)
Apply non-linear logistic sigmoid activation function to estimate the actual output for each hidden node $j, 1 \leq j \leq p$.

$$z_j = \frac{1}{1 + e^{-z_{inj}}}$$  \hspace{1cm} (2.11)

Obtaining the differential for the aforementioned activation function,

$$\frac{\partial (z_j)}{\partial x} = z_j \times (1 - z_j)$$  \hspace{1cm} (2.12)

**Output Layer Activation net value**

Each output node $y_k, (k=1, 2, \ldots, m)$ input is aggregated by multiplying input values with the corresponding weights

$$y_{ink} = w_{ok} + \sum_{j=1}^{p} z_j \cdot w_{jk}$$  \hspace{1cm} (2.13)

Apply non-linear logistic sigmoid activation function to estimate the actual output $y_k$ for each output node $k, 1 \leq k \leq m$.

$$y_k = \frac{1}{1 + e^{-y_{ink}}}$$  \hspace{1cm} (2.14)

Obtaining the differential for the aforementioned activation function,

$$\frac{\partial (y_k)}{\partial x} = y_k \times (1 - y_k)$$  \hspace{1cm} (2.15)

**Backward Pass**

In the backward pass, this actual output $y_k$, is then compared to the desired output and the error signal is computed for each output unit. The weight adjustment propagates backward from the output layer to each
unit in the transitional layer that contributes directly to the output during the learning process, thereby reducing the error along a descent direction. The error function at the output neuron is defined as in equation 2.8.

**Hidden to Output Layer Weight Updation**

For each output unit $k$ ($1 \leq k \leq m$), the error gradient calculation for the output layer is formulated as

$$
\delta_k = y_k \cdot [1 - y_k] \cdot [t_k - y_k]
$$

(2.16)

The weight updation for each output unit is computed using the following updating rule

$$
W_{jk}^{(it + 1)} = W_{jk}^{(it)} + \Delta W_{jk}^{(it)}
$$

(2.17)

where

$$
\Delta W_{jk}^{(it)} = \eta \cdot \delta_k \cdot z_j
$$

(2.18)

**Input to Output Layer Weight Updation**

For each hidden unit $j$ ($1 \leq j \leq p$), the calculation of error gradient for the hidden layer is formulated as

$$
\delta_j = \left[ \sum_{k=1}^{m} \delta_k \cdot W_{jk} \right] \cdot z_j \cdot [1 - z_j]
$$

(2.19)

The weight updation for each hidden unit is computed by the following updating rule

$$
V_{ij}^{(it + 1)} = V_{ij}^{(it)} + \Delta V_{ij}^{(it)}
$$

(2.20)
where
\[ \Delta V_{ij}(it) = \eta \cdot \delta_j \cdot x_i \] (2.21)

**Note:**
One way to increase the rate of learning is by adding a momentum term \( \alpha \), to the weight adjustment that is proportional to the amount of the previous weight change as follows (Rumelhart et al 1986):

For Output Layer,
\[ \Delta W_{jk}(it) = \eta \cdot \delta_k \cdot z_j + \alpha \cdot \Delta W_{jk}(it - 1) \] (2.22)

For Hidden Layer,
\[ \Delta V_{ij}(it) = \eta \cdot \delta_j \cdot x_i + \alpha \cdot \Delta V_{jk}(it - 1) \] (2.23)

As shown in Figure 2.10, the error is propagated backwards to modify weights in order to minimize the error.

**Figure 2.10 Gradient Descent Performance Surface**

The equations 2.22 and 2.23 reveal that the BPN algorithm utilizes two terms (or parameters) for training such as \( \eta \) (learning rate) and \( \alpha \)
(momentum coefficient). The reason for employing these two parameters is stated in the next subsection.

2.5.2 Network Parameters

Even though the BPN algorithm is one of the most studied and used NN training algorithms, there are some aspects of its functioning that are not defined in a satisfactory way which limit its application. Among these, the most important parameters that can influence the BPN training performance are the learning rate and the momentum parameter. Since the BPN algorithm utilizes these two parameters for training, it is also known as two-term BPN algorithm.

Learning Rate

Learning Rate is the training parameter that affects the speed at which the NN arrives at the minimum solution by controlling the weight and bias adjustments during the training process. In fact, the training speed of the NN is highly dependent upon the value of the learning rate. The value of the learning rate usually lies in the interval (0,1). If the learning rate is too high, the system will either oscillate around the true solution or it will diverge completely as shown in Fig 2.11. If the learning rate is too low, the system will take a long time to converge on the final solution.

![Figure 2.11 Schematic comparison between Learning rates](image)
**Momentum Coefficient**

Another training parameter to reduce training time is the use of momentum because it enhances the stability of the training process. Momentum is used to keep the training process going in the same general direction analogous to the way that momentum of a moving object behaves. The value of the momentum coefficient usually lies in the interval (0,1). This involves adding a term to the weight adjustment that is proportional to the amount of the previous weight change. In effect, the previous weight adjustment is ‘remembered’ and used to modify the next weight change by which the gradient will move uphill escaping the oscillation along the valley.

Thus the incorporation of these two parameters in the weight adjustment calculation produces a great impact on the convergence of the algorithm and problem of local minima if they are tuned to the correct value. Although the BPN algorithm is used extensively to estimate weights combination for NN, it still has some limitations which will be pointed out in the next section.

**2.5.3 Limitations**

Despite the apparent success of the BPN algorithm in training NN, there are some aspects which make the algorithm not guaranteed to be universally useful. The most important troublesome aspect of the BPN learning algorithm is the long training process because it uses GD method to update weights (Rumelhart et al 1986; Wang et al 2004; Bi et al 2005). Even on simple benchmark problems, a BPN network may require thousands of epochs to learn the desired behaviour from examples. Furthermore, the convergence behaviour of the BPN algorithm also depends on the selection of
network architecture, initial weights and biases, learning rate, momentum coefficient, activation function, value of the gain in the activation function and also size of the training set.

Nevertheless, for these limitations of BPN algorithm, many researchers have devised improvements and extensions to the basic BPN algorithm described above in order to speed up the training process.

2.6 LEARNING DIFFICULTIES WITH THE EXISTING TRAINING ALGORITHMS

In the recent years, with the progress of researches and applications, the NN technology dealing with classification has been enhanced and sophisticated. In order to speed up the NN training process, many researchers have investigated the above mentioned limitations and devoted much of their research in various formation.

2.6.1 Modifications to the standard BPN algorithms

After the discovery of the BPN algorithm (Rumelhart et al 1986), many research works have been accomplished to modify this algorithm in order to improve the efficiency and performance of NN training. Much works have been devoted to improve the generalization ability of the networks. These implicated the development of heuristic techniques based on properties of the conventional BPN algorithm. Those studies showed that the BPN performance was affected by many factors like learning structure, error function, initial weight, learning rate, momentum coefficient and activation function. Various acceleration techniques have been proposed.

This subsection presents a survey of the modifications to the standard BPN algorithm that have been suggested to speed up the training of NNs.
These modified methods can be combined to obtain an algorithm that not only provides faster training but also helps to automate the selection of certain problem-sensitive parameters.

**Improvement of the Error Function**

Since smoothing is produced by a sigmoid in steps of the error function, it is also used as the error function in NN because of its nonlinearity and the computational simplicity of its derivative (Han and Morag 1995). As the sigmoid always has a positive derivative, the slope of the error function provides a greater or lesser descent direction which can be followed and also it sometimes causes slow learning convergence. In order to overcome this problem, the Optical Back Propagation (OBP) (Otair & Salameh 2005) algorithm is designed to improve the convergence speed of the training process by adjusting the error. This algorithm is applied on the output units and is used for training with a very small learning rate only, especially when using a large training set size. Conversely, it does not guarantee to converge at global minima because if the error is close to maximum, the OBP error grows increasingly.

**Weight Initialization Technique**

During the training process, proper initialization of NN initial weights in the starting point of the training algorithm reduces the iteration number in the training process thereby increasing the training speed. It has been shown that the BPN method is sensitive to initial weights (Kolen and Pollack, 1991). Generally, small random values are picked as initial weights of the NN. Nguyen and Widrow (1990) assigned each hidden node an approximate portion of the range of the desired response, and Drago and Ridella (1992) used the statistically controlled activation weight initialization (SCAWI), which aims to prevent neurons from saturation during the adaptation process.
by estimating the maximum value that the weights should take initially. Some researchers suggested to initialize and train NN many times with different sets of small initial weights by using a probability distribution of the mean squared error (Schmidt et al 1993) and DPT (Delta Pre-Training) method (Li et al 1993) which was a very good idea if the weight space is well-conditioned. Generalized simulated annealing is employed to initialize the neural network (Sutter et al 1995). If the best of this group does not meet specifications, the process is started all over again. This method has many advocates; But it is really a trial-and-error method without any real mathematical basis. For example, initial weight values which are too large can cause ‘Premature Saturation’. So, ASCE task committee recommends that weights and thresholds be assigned initial small random values between -0.30 and +0.30 (ASCE Task Committee on Application of Artificial Neural Networks in Hydrology 2000).

In 2004, a new fast learning algorithm called Extreme Learning Machine (ELM) was proposed (Huang et al 2004) for Single Hidden Layer Feedforward Neural Networks (SLFNs) which randomly chooses the input weights and analytically determines the output weights of FNNs. In the new initialization method for neural networks, sensitivity analysis was used (Berdingas and Romeo 2006). First, random values are assigned to the outputs of the first layer. Then initial values are updated based on sensitivity formula and finally the weights are calculated using a linear system of equations. This method has the advantage of achieving a good solution in just one epoch using few computational times. Koppen et al (2009) demonstrated that a complete analysis of the MLP weight space is possible. This approach is based on clustering of the weight vectors after having trained a MLP with the BPN algorithm. Hyder et al (2009) presented a new algorithm known as Initial Weight Selection (IWS) to determine initial weights for NN. However,
starting with incorrect weight values will cause the network to be trapped in local minima or may lead to slow learning progress. The initial weights are carefully selected so that it will speed up the learning process.

**Gradient Descent with Adaptive Momentum**

Erstwhile, the momentum-coefficient is typically preferred as a constant within the range \((0, 1)\). But the experimental results revealed that the fixed momentum coefficient value seems to speed up learning only when the recent downhill gradient of the error function and the last change in weight have a parallel direction. When the recent negative gradient is in a crossing direction to the previous update, the momentum coefficient may cause the weight to be adjusted up the slope of the error surface as opposed to down the slope as preferred (Shao and Zheng 2009). In order to make learning more effective, it is necessary to adjust the momentum coefficient value adaptively instead of keeping as constant throughout the training process. Zhang et al (2006) proved that the result of the BPN method is converged with constant learning rate and adaptive momentum, whereas the error function was not assumed to be quadratic. Both the strong and weak convergence results are proved. This method can escape at local minima and speed up the network training. However, when the training enters smooth area, error gradient is closed to zero which makes the network to converge slowly.

**Gradient Descent with Adaptive Learning Rate**

In the conventional BPN algorithm, the learning rate is fixed and is uniform for all the weights in a layer (Haykin 2009; Alsmadi et al 2009). When gradient descent nears minima in the performance surface, the parameter values will oscillate back and forth around the minima. With the constant value of the learning rate, the change in the network’s parameter is fixed which results in slow convergence to the target error. One method to
prevent this, is to slow down the parameter updates by allowing the learning rate to change adaptively which will allow the network to make better responses after each weight update. The basic idea of adaptive learning rate is that at each epoch, if performance decreases towards the error goal, then the learning rate is increased by the constant parameter. If performance increases, the learning rate is decreased by another constant parameter. Based on the factor inclined to investigate, several dynamic methods for assigning the learning rate adaptively have been codified. Lyapunov stability theory-based learning algorithms were proposed for NNs (Behera et al 2006). Lyapunov Function-based learning algorithm (LF I) and its modified version (LF II) have the same structure as that of popular BPN algorithm with the difference that the fixed learning rate in BPN algorithm is replaced by an adaptive learning rate in the proposed algorithms. When the training has fallen into smooth area, error gradient is closed to zero. Then, the adaptive learning rate is large and the adjustment of weights will be slow, which could cause slow convergence to the target error.

**Gradient Descent with Adaptive Momentum and Learning Rate**

Usually the value of $\eta$ is decreased as the value of $\alpha$ is increased, so that the total step size do not get too large. Dai and MacBeth (1997) showed that the learning rate and the momentum coefficient affect the convergence of training and determined that the combinations of $\eta=0.7$ and $\alpha \in [0.8, 0.9]$ and $\eta=0.6$ and $\alpha=0.9$ exhibit the best convergence. Such combinations are similar to $\eta=0.7$ and $\alpha=0.9$ as suggested in Pao (1989) and Demuth (2000) whereas, an efficient nonlinear acceleration training algorithm, the BPN with Adaptive Learning rate and Momentum term (BPALM) (Yu and Liu 2002) was proposed based on the conventional BPN algorithm by employing an adaptive learning rate and momentum factor, where the learning rate and momentum rate are adjusted at each iteration to reduce the training time.
Gradient Descent with Adaptive Gain

An activation function is used for limiting the amplitude of the output neuron. One of the main reasons for the slow convergence of conventional BPN algorithm is that the derivative of the activation function leads to the occurrence of premature saturation of the neurons (Hollis and Paulos 1989). The value of the gain parameter $\beta$ directly influences the slope of the activation function. For large gain values ($\beta \geq 1$), the activation function approaches a step function, whereas for small gain values ($0 < \beta \leq 1$), the output values change from zero to unity over a large range of the weighted sum of the input values and the sigmoid function approximates a linear function. Wang et al. (2004) proposed an improved BPN algorithm caused by neuron saturation in the hidden layer. Each training pattern has its own activation function of hidden nodes in order to prevent neuron saturation when the network output has not acquired the desired signals. The activation functions were adjusted by the adaptation of gain parameters during the learning process (Nazri et al. 2008). It has been shown that BPN algorithm using gain variation term in an activation function converges faster than BPN algorithm. Unfortunately, higher values of gain parameter may cause instability.

Gradient Descent with Adaptive Momentum, Gain and Learning Rate

Introduction of adaptive gain together with adaptive momentum and adaptive learning rate into weight update process can prevent the network from trapping into local minima that is caused by the neuron saturation in the hidden layer. Hamid et al. (2004) proposed a new modified BPGD with adaptive gain, adaptive momentum and adaptive learning rate (BPGD-AGAMAL).
2.6.2 Second order algorithm

Next, the algorithm that derives the second order differential equation from the cost functions for updating the weight during the training process has been listed. The most popular second order training algorithms are quasi-Newton methods or Levenberg–Marquardt (LM) (Yu and Wilamowski 2010; Yu and Wilamowski 2012) and Conjugate Gradient (CG) methods (Ampazis and Perantonis 2002). Eventhough, the above second order approaches achieve good results, they are computationally very expensive. Ampazis and Perantonis (2002) extracted the importance of the LM and CG methods and derived two different approaches Levenberg–Marquardt with adaptive momentum (LMAM) and optimized Levenberg–Marquardt with adaptive momentum (OLMAM) second order algorithm. Yu and Wilamowski (2010 and 2012) applied vector multiplication for determining the gradient vector and Hessian matrix instead of matrix multiplication which significantly reduces the cost of memory for training and thereby improves the training speed. Unfortunately, the Hessian matrix is usually too large to compute and to store. Inverting a Hessian matrix can be extremely time consuming. Furthermore, there is no guarantee that the Hessian matrix is nonsingular and of full rank. Unless these problems are properly solved, there is no real advantage of using the second-order information.

2.7 INCREMENTAL LEARNING METHOD

In the recent years, incremental learning method has attracted much attention as a result of the increasing demand for developing machine vision systems. It is the process of learning new information from the new training dataset without losing prior knowledge in order to improve its classification. Most of the aforementioned classification methods lack the incremental learning ability. Various algorithms have been suggested for incremental learning.
Zhelong et al (2012) have designed an incremental learning method based on Probabilistic Neural Network (PNN) and an Adjustable Fuzzy Clustering (AFC) algorithm. The Zhelong’s method has the ability to easily learn additional information from new training data to improve the recognition accuracy as well as freely add new activities to be detected and remove existing activities. In the updating process, only the new training samples are used, and the previously used training samples are not involved.

2.8 SUMMARY

This chapter has reviewed the NNs and their learning processes extensively which are beneficial for understanding their basic elements, major parameters, learning or training processes and latest learning algorithms and applications. It has also investigated the main classifications of linear, nonlinear and hybrid latest training algorithms for NNs. It is seen that the slow convergence has been a major problem for all BPN learning algorithms. In order to improve the training speed of BPN, a new algorithm need to be developed which forms the focus of this thesis.