CHAPTER 5

IMPLEMENTATION OF BACKPROPAGATION LEARNING ALGORITHMS

5.1 INTRODUCTION

The backpropagation learning algorithm is one of the most widely used algorithms to train artificial neural networks. The backpropagation learning algorithm aims to determine a way to map the inputs rightly to its corresponding outputs from a given set of training data. In this chapter, Levenberg-Marquardt (LM) backpropagation algorithm, Resilient Backpropagation (RPROP) algorithm and Broyden Fletcher Goldfarb Shanno (BFGS) Quasi Newton backpropagation algorithm, which are well-known for their faster convergence are implemented to optimize the error rate obtained using the subtractive clustering algorithm in Chapter 4, to find out the quantity of alloying elements that have to be added at the time of ladle refining process to make S235JRG2 steel grade.

5.2 BACKPROPAGATION LEARNING ALGORITHM

One of the most significant advancements in artificial neural networks (ANNs) is the backpropagation learning algorithm (Arthur Earl Bryson & Yu-Chi Ho 1969; Werbos 1974; Le Cun 1985; Parker 1985; Rumelhart et al. 1986a; Rumelhart et al. 1986b, Sivanandam & Deepa 2011). Backpropagation learning algorithm is an efficient means for training ANNs having multilayer. By using neural networks, this algorithm has revived the
technical and engineering society to design and process many quantitative facts. This algorithm is implemented on multilayer feed-forward neural networks having processing constituents with constantly changing activation functions. The networks related to the backpropagation learning algorithm are also known as Back Propagation Networks (BPN). For a provided set of training input-output pair, backpropagation algorithm offers an effective means for varying the weights in a feed forward network to categorize the input pattern rightly. The gradient descent technique as employed in case of perceptron networks with changeable units is the primary notion of this weight update method. In this method the error is back propagated to the unit that is hidden. The objective of ANN is to train the network to accomplish equilibrium among the networks capability to react and its capability to provide appropriate replies to the input that is alike but not similar to the one that is employed in training.

The BPN network differs from other networks in terms of weight computation procedure during networks learning period. Generally, for perceptrons with multilayer, the computation of hidden layer weights that would result in extremely minimum or zero output error is difficult. Increasing the hidden layers increases the difficulty of network training. The error should be computed in order to update the weights. At the output layer, the error that represents the difference between the actual and required output is calculated. There won’t be any direct information about the error at the hidden layers. So, the main objective is to compute an error at the hidden layer using some other techniques that results in reduction of error at the output.

The training of a backpropagation network involves three stages, namely the feed-forward stage, backpropagation of the errors, and weight updation. The testing of the backpropagation network entails only the calculation of feed-forward stage. The BPN can have more number of hidden
layers. However, one hidden layer is adequate. Although the training consumes much time, once the net is trained it can generate its output more quickly.

A BPN which looks like a feed forward net with multilayer includes an input, a hidden, and an output layer. The neurons in both the output units and the hidden units consist of biases. The biases are the links from units whose activation is one at all times. The bias also acts same as that of weights. The backpropagation network architecture is shown in Figure 5.1. It illustrates only the data flow direction for the feed forward stage. In the backpropagation stage of learning, the signals are passed in the backward direction. The inputs are passed to the backpropagation network and the output acquired from the network might be either bipolar (-1, +1) or binary (0, 1). The use of activation function can be of any type that rises uniformly as well as it could be differentiable.

![Figure 5.1 Backpropagation network architecture](image-url)
The terms used in Figure 5.1 and for training algorithm are as follows:

- \( x \) represents input training vector
- \( t \) represents output target vector
- \( x_i \) is input unit \( i \). As the identity activation function is used at input layer, the signals at the output are similar as the input.
- \( w_{0k} \) is bias on \( k^{th} \) output unit
- \( v_{0j} \) is bias on the \( j^{th} \) hidden unit
- \( \alpha \) is learning rate factor
- \( z_j \) is hidden unit \( j \)
- \( y_k \) is output unit \( k \)
- \( \delta_k \) is error at output unit \( y_k \)
- \( \delta_j \) is error at hidden unit \( z_j \)

5.2.1 Training Algorithm

**Step 0:** Set small random values for learning rate and weights.

**Step 1:** If stopping condition is false, carry out steps from 2-9

**Step 2:** For all training pair, carry out steps from 3-8.

\textit{Phase I - Feed forward phase:}

**Step 3:** Every unit of input gets input signal \( x_i (i = 1 \text{ to } n) \) and forwards it to the hidden unit.

**Step 4:** Every hidden unit \( z_j (j = 1 \text{ to } p) \) adds its weighted input signals to compute the net input:

\[
z_{inj} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij} \quad (5.1)
\]
Compute the hidden unit output by employing its activation functions over $z_{inj}$. The bipolar or binary sigmoidal activation function can be used:

$$z_j = f(z_{inj})$$  \hspace{1cm} (5.2)$$

and pass this output signal to the input of output layer units.

**Step 5:** Compute the net input for all output unit $y_k (k = 1 \text{to} m)$,

$$y_{ink} = w_{ok} + \sum_{j=1}^{p} z_j w_{jk}$$ \hspace{1cm} (5.3)$$

and compute the output signal by applying the activation function

$$y_k = f(y_{ink})$$  \hspace{1cm} (5.4)$$

**Phase II - Back propagation of error:**

**Step 6:** Every output unit $y_k (k = 1 \text{to} m)$ gets a target pattern related to the training pattern of the input and calculates the error adjustment term:

$$\delta_k = (t_k - y_k)f'(y_{ink})$$ \hspace{1cm} (5.5)$$

Based on the use of activation function, the derivative term $f'(y_{ink})$ can be computed. The binary and bipolar sigmoidal activation functions are usually used as it has features like non decreasing uniformity, continuity, and differentiability. Update the variation in bias and weights based on the computed error adjustment term,
\[ \Delta w_{jk} = \alpha \delta_k z_j \]  
\[ \Delta w_{0k} = \alpha \delta_k \]  

Furthermore, send \( \delta_k \) towards the back to hidden layer.

**Step 7:** Every hidden unit \((z_j, j = 1 to p)\) adds its delta inputs from the output units:

\[ \delta_{mj} = \sum_{k=1}^{n} \delta_k w_{jk} \]  

To compute the error term, multiply \( \delta_{mj} \) term with \( f'(z_{mj}) \) derivative,

\[ \delta_j = \delta_{mj} f'(z_{mj}) \]  

The computation of \( f'(z_{mj}) \) derivative can be done based on the use of binary or bipolar sigmoidal function. Update the variation in bias and weights based on the computed \( \delta_j \),

\[ \Delta v_{ij} = \alpha \delta_j x_i \]  
\[ \Delta v_{oj} = \alpha \delta_j \]  

**Phase III – Updation of bias and weight**

**Step 8:** Every output unit \((y_k, k = 1 to m)\) revises its weights and bias:

\[ w_{jk} (new) = w_{jk} (old) + \Delta w_{jk} \]  
\[ w_{0k} (new) = w_{0k} (old) + \Delta w_{0k} \]
Every hidden unit ($z_j, j = 1 \text{ to } p$) revises its weights and bias:

\[ v_{ij}(new) = v_{ij}(old) + \Delta v_{ij} \]  \hspace{1cm} (5.14)

\[ v_{0j}(new) = v_{0j}(old) + \Delta v_{0j} \]  \hspace{1cm} (5.15)

**Step 9:** Stop the process of training when the calculated output is equivalent to the desired output or when the number of epochs reaches the maximum value.

For weight updation, the incremental learning procedure is used by the training algorithm in which the change of weights is done instantly after a pattern of training is given. In another mode of training, namely batch learning approach, the alterations in weights will be done only when the entire set of training patterns are given. The efficacy of both the learning methods is based on the problem. However, to retain the instant weight alterations, batch learning approach necessitates extra local space for each link. The BPN is analogous to optimal Bayesian discriminant function when it is used as a classifier.

The complication in this case is that, whether the backpropagation learning algorithm can converge at all times and determine appropriate weights for the network even after sufficient learning. The convergence takes place as it employs a gradient descent on the surface of error in the weight space, and this will move the surface of error downwards to the nearby smallest error and it will halt. This happens to be true when the surface of error as well as the connection present between the output and the input training sample is deterministic. This won’t be the case in the present world, since the square error surfaces that were produced are arbitrary at all times. This is the random behaviour of the backpropagation algorithm, which relies
entirely on the randomly determined gradient descent technique. The backpropagation network is an exceptional case of stochastic method of approximation.

The backpropagation algorithm might get trapped in local minima when no convergence takes place, and might be incapable to determine the reasonable results. The arbitrariness of the BPN assists it to escape from local minima. The error functions are supposed to have global minima due to variation of weights that maintain the network input and output function unaltered. This makes surfaces of the error to own many troughs.

5.2.2 Learning Factors of Backpropagation Algorithm

The training of backpropagation network relies on the selection of different factors. Moreover, the convergence of backpropagation network relies on some significant aspects of learning like learning rate factor, weight initialization factor, training pattern’s nature and size, rule for updation, number of layers, and neurons per layer.

5.2.2.1 Initial weights

The final result might be disrupted due to initial weights of a feed forward net having multilayer. They have to be set with small arbitrary values. The selection of weight at primary stage finds out how rapidly the convergence of network takes place. Assigning high values for initial weights may saturate the sigmoidal activation functions used here from the starting itself and the method might be trapped in local minima or at an extremely smooth terrain at the initial stage itself. The range of weight \( w_{ij} \) can be chosen as

\[
-3 \frac{3}{\sqrt{O_i}, \sqrt{O_j}}
\]  

(5.16)
where \( o_i \) is the quantity of operating parameter \( j \) that feed forward to operating parameter \( i \). A technique named Nyugen-Widrow initialization can also be used for initialization. By using this initialization technique the network can be made to converge faster. The notion of this initialization technique is that, it relies on the geometric testing of the hidden neuron’s response to one input. This technique is employed to advance the hidden units learning capability. The arbitrary assigning of weights linking neurons of the input to the neurons of hidden unit is attained using the Equation (5.17).

\[
v_{ij}(new) = \gamma \frac{v_{ij}(old)}{\|v_{ij}(old)\|} \tag{5.17}
\]

where \( \bar{v}_j \) denotes the average weight computed for each value of \( i \), and \( \gamma = 0.7(P)^{\frac{1}{n}} \) represents the scale factor where \( P \) is the hidden neurons quantity and \( n \) is the input neurons quantity.

5.2.2.2 Learning rate factor \( \alpha \)

The learning rate factor \( \alpha \) has an effect on the BPN convergence. If the value of \( \alpha \) is large, the convergence speed will be more, but it may cause overrun. If value of \( \alpha \) is small then it leads to vice-versa effect. For numerous BPN experiments, the \( \alpha \) value is chosen from \( 10^{-3} \) to 10. Hence, the learning rate with larger value results in faster learning. However, variation of weights will be there, whereas the learning rate with lesser value results in sluggish learning.

5.2.2.3 Momentum factor

For smaller learning rate \( \alpha \) the gradient descent works in an extremely slow manner and for larger learning rate \( \alpha \) the gradient descent
fluctuates extensively. Adding up a momentum factor to the standard method of gradient descent permits large value for learning rate without fluctuations. The term $\eta \in [0, 1]$ denotes the momentum factor, and the value frequently used for it is 0.9. Moreover, this methodology is very much useful when a few training information differs much from the greater part of the information. A momentum factor may be employed with either updation by batch mode or pattern by pattern. During updation by batch-mode, it has the outcome of total averaging over the samples. The pattern-by-pattern method departs a number of valuable data for updating weights, though the averaging is only partial. The formulas employed here for updating weights are

$$w_{jk}(t + 1) = w_{jk}(t) + \alpha \delta_k z_j + \eta [w_{jk}(t) - w_{jk}(t - 1)] \quad (5.18)$$

and

$$v_{ij}(t + 1) = v_{ij}(t) + \alpha \delta_j x_i + \eta [v_{ij}(t) - v_{ij}(t - 1)] \quad (5.19)$$

The momentum factor furthermore facilitates the convergence rapidly.

5.2.2.4 Generalization

Backpropagation network is the excellent system for generalization. When the network wisely interjects with input networks, which are new to the network, then it is called as a generalized network. If the trainable factors are more for the provided quantity of training data, the net may learn capably but won’t generalize proficiently. This is generally referred as overfitting. The key to solve this problem is to terminate the training when an increase in the error is observed on the test set. If the trainable factors are few, then the net
will not learn the training data and executes so badly on the test data set. In order to improve the network’s capability to generalize from a set of training data to a set of test data, it is enviable to create little modifications in the pattern’s input space without altering the elements of the output. This can be accomplished by establishing alterations in the training pattern’s input space like part of the training set. Nevertheless, this process is computationally pricier. Moreover, a net having more nodes are capable of remembering the training set of data at the price of generalization. Hence, smaller networks are usually chosen than larger ones.

5.2.2.5 Number of training data

There must be enough and appropriate training data. A thumb rule exists, which defines that the training data must include the whole anticipated input space, and at the time of training, the pairs of training-vector must be chosen arbitrarily from the set. Imagine that the input space as being linearly isolatable into “L” separate sections with their limits being a portion of hyper planes. Let the lower bound on the numeral of training patterns be “T”. Then, selecting $T$ in order that $T/L >> 1$ will permit the net to distinguish types of patterns with well piecewise hyper-plane separation. Moreover in some instances, standardization or scaling should be carried out to aid learning.

5.2.2.6 Number of hidden layer nodes

If the hidden layers are more in the backpropagation network, then the computations carried out for one layer has to be done again to each layer and are totaled at the last. The hidden layer size is vital in case of every feed forward networks having multilayer. The required amount of hidden units for an application needs to be found out independently. Generally, the hidden layer size is found out experimentally. The hidden nodes size should be comparatively a little fraction of the input layer for an appropriate sized
network. For instance, the network might require additional hidden nodes when no convergence takes place. Alternatively, if the convergence takes place, the user might attempt an extremely small number of hidden nodes and after that finalize the size according to the performance of the entire system.

5.2.3 Testing Algorithm

**Step 0:** Assign the weights by getting it from the training algorithm.

**Step 1:** Carry out Steps 2-4 for all input vectors.

**Step 2:** For \( x_i (i = 1 to n) \), set the activation of input unit.

**Step 3:** For \( j = 1 to p \), compute the net input to hidden unit \( x \) and its output,

\[
\begin{align*}
    z_{inj} &= v_{0j} + \sum_{i=1}^{n} x_i v_{ij} \\
    z_j &= f(z_{inj})
\end{align*}
\]  

**Step 4:** For \( k = 1 to m \), compute the result of the output layer unit,

\[
\begin{align*}
    y_{nk} &= w_{0k} + \sum_{j=1}^{p} z_j w_{jk} \\
    y_k &= f(y_{nk})
\end{align*}
\]  

For computing the output, employ sigmoidal activation function.

5.2.4 Activation Functions

In order to compute the output response of a neuron, the activation function is employed on the sum of weighted input signal. The data processing of a processing constituent can be perceived as comprising of
input and output parts. The term $f$ representing integration function is related to the processing component input. This function helps to merge data or proof and activation, from an outside origin or other processing components into a net input to the processing component. In order to make sure that a neuron’s reply is enclosed, the nonlinear activation function is employed. This means that the real response of a neuron is trained as an effect of huge or little activating force and is therefore manageable.

To attain the merits of network having multilayer from a net having single layer, some nonlinear functions are employed. The outcome acquired is similar as that might be acquired via a network having a single layer while a signal is fed all the way through a network having multilayer with activation functions that are linear. Because of this cause, nonlinear functions are extensively employed in networks having multilayer.

The backpropagation networks extensively make use of sigmoidal activation functions, as the rapport among the functions value at a point and the derivative value at that point minimizes the calculation difficulty at the time of training. The sigmoidal activation functions are classified into two types:

**Binary Sigmoid Function:** Also known as unipolar or logistic sigmoid function. It can be expressed as

$$f(x) = \frac{1}{1 + e^{-\lambda x}}$$  \hspace{1cm} (5.24)

where $\lambda$ represents the factor of steepness. If this function is differentiated, then

$$f'(x) = \lambda f(x)[1 - f(x)]$$  \hspace{1cm} (5.25)
At this point the sigmoid function ranges from 0 to 1.

**Bipolar Sigmoid Function:** This sigmoid function is expressed as

\[ f(x) = \frac{2}{1 + e^{-\lambda x}} - 1 = \frac{1 - e^{-\lambda x}}{1 + e^{-\lambda x}} \]  \hspace{1cm} (5.26)

where \( \lambda \) denotes the factor of steepness and the range of sigmoid function lies between -1 and +1. If this function is differentiated, then

\[ f'(x) = \frac{\lambda}{2} [1 + f(x)] [1 - f(x)] \]  \hspace{1cm} (5.27)

### 5.3 LEVENBERG-MARQUARDT BACKPROPAGATION ALGORITHM

Levenberg-Marquardt method of backpropagation (Marquardt 1963; Hagan & Menhaj 1994; Hagan et al. 1996; Aldrich 2002; Sapna et al. 2012) is intended explicitly to reduce the sum of the square errors of the form

\[ S = \frac{1}{2} \sum_{k} (r_k)^2 = \frac{1}{2} \| r \|^2 \]  \hspace{1cm} (5.28)

where \( r_k \) specifies the error in the \( k^{th} \) paradigm or sample and \( r \) is a vector with component \( r_k \). If there is only a little difference among the preceding weight vector and the newly obtained weight vector, then by using Taylor series the error vector could be expanded to first order.

\[ r_{(j+1)} - r_{(j)} = \partial f_k / \partial w_i (w_{(j+1)} - w_{(j)}) \]  \hspace{1cm} (5.29)

As a result, the error function can be written as
\[ S = \frac{1}{2} \| r_{(j)} + \partial r_k / \partial w_i (w_{(j+1)} - w_{(j)}) \|^2 \]  

(5.30)

Reducing the error function according to the new weight vector, presents

\[ w_{(j+1)} = w_{(j)} - \left( Z^T Z \right)^{-1} Z^T r_{(j)} \]  

(5.31)

where \( Z_w \equiv \partial r_k / \partial w_i \)

As the sum of square error function has Hessian of the form

\[ (G)_{ij} = \partial^2 S / \partial r_i \partial r_j = \sum \left\{ (\partial r_i / \partial w_i)(\partial r_j / \partial w_j) + r_i \partial^2 r_i / \partial w_i \partial w_j \right\} \]  

(5.32)

Ignoring the second term, the Hessian could be expressed as

\[ G = Z^T Z \]  

(5.33)

Consequently, the weight updation entails an approximation or inverse Hessian thereof for networks that are nonlinear. The Hessian is quite simple to calculate, as it is based on first order derivatives according to the weights of the network that are accommodated effortlessly by backpropagation. Even though the formula for updating weights might be employed iteratively for reducing the error function, this might effect in a huge step size, which would nullify the linear estimation.

So as to assure the strength of the linear estimation the error function is reduced when the step size is kept small. It is achieved through a revised error function of the form

\[ S = \frac{1}{2} \| r_{(j)} + \partial r_k / \partial w_i (w_{(j+1)} - w_{(j)}) \|^2 + \zeta \| w_{(j+1)} - w_{(j)} \|^2 \]  

(5.34)
where $\zeta$ represents the parameter ruling the step size. Reducing the revised error according to $w_{(j+1)}$, presents

$$w_{(j+1)} = w_{(j)} - \left(Z^T Z + \zeta I\right)^{-1} Z^T r_{(j)}$$

(5.35)

When the value of $\zeta$ is extremely large then the Levenberg-Marquardt algorithm switches to gradient descent, and if the value of $\zeta$ is extremely small then the algorithm switches to Newton method.

5.4 RESILIENT BACKPROPAGATION (RPROP) ALGORITHM

Resilient backpropagation algorithm is an effective method of learning for feed-forward networks having multilayer. RPROP carries out weight updation according to the local gradient data (Riedmiller & Braun 1993; Sapna et al. 2015). Compared to the adaptation methods that were developed earlier, the attempt towards weight modification in RPROP is not distorted by performance of the gradient. In order to accomplish this, for every weight its respective update-value $\Delta_{ij}$ is presented, which exclusively finds out the magnitude of the weight update. At the time of learning procedure the adaptive update-value progress according to its confined prospect on the error function $R$, based on the succeeding learning rule:

$$\Delta_{ij}^{(t)} = \begin{cases} 
\eta^+ \times \Delta_{ij}^{(t-1)}, & \text{if } \frac{\partial R^{(t-1)}}{\partial \omega_{ij}} \times \frac{\partial R^{(t)}}{\partial \omega_{ij}} > 0 \\
\eta^- \times \Delta_{ij}^{(t-1)}, & \text{if } \frac{\partial R^{(t-1)}}{\partial \omega_{ij}} \times \frac{\partial R^{(t)}}{\partial \omega_{ij}} < 0 \\
\Delta_{ij}^{(t-1)}, & \text{else}
\end{cases}$$

(5.36)

where $0 < \eta^- < 1 < \eta^+$
The weight updation rule performs as follows: At each instance the partial derivative of the related weight \( \omega_{ij} \) (weight from neuron \( j \) to neuron \( i \)) alters its sign, which denotes that the updation done at the end was excessively large and the algorithm has leaped above the local minimum. The update-value \( \Delta_{ij} \) is reduced by the aspect \( \eta^- \). If there is no change in sign of the partial derivative, then the update-value is increased a little to speed up the convergence in superficial regions.

If the update-value for all weight is modified once, then the weight-update itself pursues an incredibly effortless canon: when the sign of the derivative is positive, then the weight is lessened by its update-value, and when the sign of the derivative is negative, then the weight is increased by its update-value:

\[
\Delta \omega_{ij}^{(r)} = \begin{cases} 
-\Delta_{ij}^{(r)} & \text{, if } \frac{\partial R^{(r)}}{\partial \omega_{ij}} \geq 0 \\
+\Delta_{ij}^{(r)} & \text{, if } \frac{\partial R^{(r)}}{\partial \omega_{ij}} < 0 \\
0 & \text{, else}
\end{cases} 
\]

(5.37)

\[
\omega_{ij}^{(r+1)} = \omega_{ij}^{(r)} + \Delta \omega_{ij}^{(r)}
\]

(5.38)

But, one exemption exists: when the preceding step becomes larger and the minimum was lost, the preceding weight-update is regressed:

\[
\Delta \omega_{ij}^{(r)} = -\Delta \omega_{ij}^{(r-1)} \times \frac{\partial R^{(r-1)}}{\partial \omega_{ij}} \times \frac{\partial R^{(r)}}{\partial \omega_{ij}} \geq 0
\]

(5.39)

Because of that back-off in weight step, once again the sign of the derivative is made to vary in the subsequent step. So as to eliminate a dual chastisement of the update value, the value of the update should not be
modified in the following steps. This can be accomplished by assigning 
\[ \frac{\partial R^{(t-1)}}{\partial \omega_j} := 0 \] in the \( \Delta_j \) modification rule above. The weights and also the
values of the update are modified whenever the entire set of the pattern has
been offered to the network at once.

5.4.1 Algorithm

The subsequent pseudo-code explains the core of the RPROP
learning and alteration procedure. In order to convey the maximum and
minimum of two numbers, the maximum operator and minimum operator is
assumed; the sign operator becomes -1, when the argument is negative, +1, when
the argument is positive, and 0 or else. For every biases and weights \{ 
\[
\text{if} \left( \frac{\partial R}{\partial \omega_j} (t-1) \ast \frac{\partial R}{\partial \omega_j} (t) \gtrless 0 \right) \text{ then } \{
\Delta_j (t) = \text{minimum} \left( \Delta_j (t-1) \ast \eta^+, \Delta_{\text{max}} \right)
\Delta \omega_j (t) = - \text{sign} \left( \frac{\partial R}{\partial \omega_j} (t) \right) \ast \Delta_j (t)
\omega_j (t+1) = \omega_j (t) + \Delta \omega_j (t)
\}\}
\]
\[
\text{else} \text{ if} \left( \frac{\partial R}{\partial \omega_j} (t-1) \ast \frac{\partial R}{\partial \omega_j} (t) \lessgtr 0 \right) \text{ then } \{
\Delta_j (t) = \text{maximum} \left( \Delta_j (t-1) \ast \eta^-, \Delta_{\text{min}} \right)
\omega_j (t+1) = \omega_j (t) - \Delta \omega_j (t-1)
\frac{\partial R}{\partial \omega_j} (t)=0
\}\}
\]
else if \( \frac{\partial R}{\partial \omega_j} (t-1) \times \frac{\partial R}{\partial \omega_j} (t) = 0 \) then {

\[
\Delta \omega_j (t) = - \text{sign} \left( \frac{\partial R}{\partial \omega_j} (t) \right) \times \Delta_j (t)
\]

\[
\omega_j (t+1) = \omega_j (t) + \Delta \omega_j (t)
\]
}

5.4.2 Parameters

Every update-value \( \Delta_{ij} \) is assigned to a primary value \( \Delta_0 \) in the beginning. For \( \Delta_0 \), directly find out the initial weight step size. It is rather selected in a suitable ratio to the primary weight’s size. A fine selection is possibly \( \Delta_0 = 0.1 \). But, the option of this factor is not significant by any means. Even for very high or very less \( \Delta_0 \) values, quick convergence is arrived.

By Spiral Learning Task exclusion, the update-value ranges were limited to a maximum value of \( \Delta_{\text{max}} = 50.0 \) and a minimum value of \( \Delta_{\text{min}} = 1 \times 10^{-6} \) to evade problems of underflow or runoff in floating point variables. It is observed that the error can be reduced by assigning the maximum value of the update to a significantly lesser value \( \Delta_{\text{max}} = 1.0 \).

The value of \( \eta^- \) representing the lessen factor and \( \eta^+ \) signifying the increase factor was assigned based on the succeeding observations: when there exists a jump above minimum, the preceding update-value was excessively high. It is not identified from the information of the gradient that to what extent the minimum was lost. The fine assumption is to set the value
of update to $\eta^- = 0.5$. The $\eta^+$ signifying the increase factor has to be more adequate to permit quick rise of the update-value in superficial areas of the error function. The procedure of learning can be modified substantially, if very large value of increase factor directs to constant variations in the weight step direction. So as to make the selection of the parameter very much easier, the value of lessen/increase factors are set persistently to $\eta^- = 0.5$ and $\eta^+ = 1.2$.

One of the major merits of RPROP is that, for numerous issues no parameters selection is required to attain almost optimal or best possible convergence times.

**5.5 BFGS QUASI NEWTON BACKPROPAGATION ALGORITHM**

BFGS algorithm is named for its creators Broyden Fletcher Goldfarb Shanno. This algorithm uses an iterative procedure to find the solution for unconstrained nonlinear optimization problems (Gill et al. 1981; Dennis & Schnabel 1983; Sapna et al. 2015). The Newton’s method is approximated well by BFGS method. The Newton’s method is a kind of hill-climbing optimization methods which looks for a static point of a function. The gradient must be zero to achieve optimality for such kind of problems. Both BFGS and Newton’s methods are not sure to converge until the function has a quadratic Taylor extension close to the best possible value. These methods make use of both the first order as well as the second order derivatives of the function. Nevertheless, BFGS has proved its superiority in attaining better solution even for the non-smooth optimization problems (Lewis & Overton 2012).
Quasi-Newton techniques do not require computation of the Hessian matrices of second order derivatives. As an alternative, the updates defined by the gradient evaluation are used to approximate the Hessian matrix. The Quasi-Newton techniques are the simplified form of secant technique to compute the root of the first order derivative for problems involving several dimensions. In problems involving several dimensions, the secant equation does not indicate a distinctive result. Moreover, Quasi-Newton techniques vary in how they limit the result. The BFGS algorithm is one of the well-known members of this class (Nocedal & Wright 2006).

The direction of search $p_k$ on phase $k$ is provided by the result of cognate of Newton equation

$$S_k p_k = -\nabla f(x_k) \quad (5.40)$$

where $\nabla f(x_k)$ represents the gradient of the function calculated at $x_k$, and $S_k$ denotes an approximation to the Hessian matrix that is updated in an iterative manner at every step. In order to find the subsequent point $x_{k+1}$, a line search method is used in the direction $p_k$. Updating the Hessian matrix that is approximated at phase $k$ by adding two matrices eliminates the need of entire Hessian matrix at the point $x_{k+1}$ to be calculated as that of $S_{k+1}$,

$$S_{k+1} = S_k + U_k + V_k \quad (5.41)$$

Both $U_k$ and $V_k$ are similar rank-one matrices. However, they have dissimilar bases. On this update the state of Quasi-Newton imposed is

$$S_{k+1}(x_{k+1} - x_k) = \nabla f(x_{k+1}) - \nabla f(x_k) \quad (5.42)$$
5.5.1 Algorithm

From the primary assumption $x_0$ and an approximate Hessian matrix $S_0$, the subsequent steps are reiterated till $x_k$ converges to the result.

**Step 1:** Find the direction of $p_k$ by computing $S_k p_k = -\nabla f(x_k)$.

**Step 2:** Carry out the line search procedure to determine an appropriate step size $\alpha_k$ in the way computed in Step 1, then update $x_{k+1} = x_k + \alpha_k p_k$.

**Step 3:** Assign $r_k = \alpha_k p_k$.

**Step 4:** $b_k = \nabla f(x_{k+1}) - \nabla f(x_k)$.

**Step 5:**

$$S_{k+1} = S_k + \frac{b_k b_k^T}{b_k^T r_k} - \frac{S_k r_k r_k^T S_k}{r_k^T S_k r_k}.$$  

The term $f(x)$ specifies the objective function that has to be reduced. The convergence can be verified by examining the gradient rule $|\nabla f(x_k)|$. Matter-of-fact, $S_0$ may be set with $S_0 = I$, in order that the Step 1 will be similar to a gradient descent method, however other steps are refined even more by the approximation to the Hessian $S_k$.

The matrix $S_k$ is inversed to perform Step 1. The matrix $S_k$ is inversed by employing the Sherman-Morrison formula to the Step 5 of the algorithm, presenting

$$S_{k+1}^{-1} = \left(I - \frac{r_k r_k^T}{b_k^T r_k}\right) S_k^{-1} \left(I - \frac{b_k r_k^T}{b_k^T r_k}\right) + \frac{r_k r_k^T}{b_k^T r_k} \quad (5.43)$$
This can be calculated effectively without utilizing non-permanent matrices, realizing that $S_{k}^{-1}$ is similar, and that $b_{k}^{T}S_{k}^{-1}b_{k}$ and $r_{k}^{T}b_{k}$ are scalar, using an expansion as follows:

$$S_{k+1}^{-1} = S_{k}^{-1} + \frac{(r_{k}^{T}b_{k} + b_{k}^{T}S_{k}^{-1}b_{k})(r_{k}^{T})}{(r_{k}^{T}b_{k})^{2}} - \frac{S_{k}^{-1}b_{k}r_{k}^{T} + r_{k}^{T}b_{k}S_{k}^{-1}}{r_{k}^{T}b_{k}}$$

(5.44)

In problems like statistical estimation, the final Hessian matrix can be inversed to calculate the confidence intervals or credible intervals for the solution.

Nevertheless, these quantities are described technically by the exact Hessian matrix, and the approximation of BFGS might not converge to an exact Hessian matrix.

5.6 PERFORMANCE MEASURE

The performance of backpropagation algorithms are assessed using Mean Square Error (MSE), Root Mean Square Error (RMSE), and Regression (R) analysis. MSE is the average squared difference between the outputs of the network and the outputs of the target, and RMSE is the square root of MSE. Lesser values are better, whereas zero implies no error. Regression R evaluation is carried out to appraise the rapport between the outputs of the network and the outputs of the training data. The R value of 1 represents a close rapport, 0 an arbitrary rapport. The Equation (5.45) is used to calculate the MSE, the Equation (5.46) is used to calculate the RMSE, and the Equation (5.47) is used to calculate the Regression R.

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (n_{i} - t_{i})^{2}$$

(5.45)
\[ RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (n_i - t_i)^2} \]  \hspace{1cm} (5.46)

\[ R = \frac{N \sum_{i=1}^{N} n_i t_i - \sum_{i=1}^{N} n_i \sum_{i=1}^{N} t_i}{\sqrt{N \sum_{i=1}^{N} n_i^2 - \left( \sum_{i=1}^{N} n_i \right)^2} \sqrt{N \sum_{i=1}^{N} t_i^2 - \left( \sum_{i=1}^{N} t_i \right)^2}} \]  \hspace{1cm} (5.47)

where \( n_i \) denotes the network outputs, \( t_i \) denotes the target outputs and \( N \) represents the number of training data. The target outputs \( t_i \) represents the output parameters of optimal cluster centers, which are given in Table A 2.1.

### 5.7 IMPLEMENTATION AND RESULTS

The optimal cluster centers acquired using subtractive clustering algorithm, that has been given in Table A 2.1, are fed as input to the Levenberg-Marquardt backpropagation algorithm, Resilient backpropagation (RPROP) algorithm and BFGS Quasi Newton backpropagation algorithm to validate their performance with respect to optimization of RMSE value obtained using the subtractive clustering algorithm in Chapter 4. The simulation is carried out using MATLAB R2014a. The performances of the proposed backpropagation algorithms are shown in Figures 5.2-5.7. The performance comparison of Levenberg-Marquardt, RPROP, and BFGS Quasi Newton backpropagation algorithms are given in Table 5.1.
Figure 5.2 Performance of Levenberg-Marquardt backpropagation algorithm

Figure 5.3 Regression plot obtained using Levenberg-Marquardt backpropagation algorithm
Figure 5.4 Performance of RPROP algorithm

Figure 5.5 Regression plot obtained using RPROP algorithm
Figure 5.6 Performance of BFGS Quasi Newton backpropagation algorithm

Figure 5.7 Regression plot obtained using BFGS Quasi Newton backpropagation algorithm
Table 5.1 Performance comparison of Levenberg-Marquardt, RPROP, and BFGS Quasi Newton backpropagation algorithms

<table>
<thead>
<tr>
<th>Backpropagation Algorithm</th>
<th>MSE</th>
<th>RMSE</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Levenberg-Marquardt</td>
<td>0.080122</td>
<td>0.2830</td>
<td>0.98633</td>
</tr>
<tr>
<td>RPROP</td>
<td>0.034397</td>
<td>0.1854</td>
<td>0.99303</td>
</tr>
<tr>
<td>BFGS Quasi Newton</td>
<td>0.00064952</td>
<td>0.0254</td>
<td>0.99977</td>
</tr>
</tbody>
</table>

From Figures 5.2-5.7 and Table 5.1, it is evident that the performance of BFGS Quasi Newton is better when compared to the Levenberg-Marquardt and RPROP backpropagation algorithms in terms of MSE, RMSE, and Regression R.

5.8 SUMMARY

This chapter aimed to validate the performance of Levenberg-Marquardt backpropagation algorithm, RPROP algorithm, and BFGS Quasi Newton backpropagation algorithm in optimizing the error rate obtained using subtractive clustering algorithm to determine the amount of alloying elements that have to be added during the process of ladle refining to produce S235JRG2 steel grade. The consistency of the proposed backpropagation algorithm depends on the dataset that has to be trained. Based on the performance analysis, BFGS Quasi Newton backpropagation algorithm has provided the better RMSE value of 0.0254 and Regression R value of 0.99977, when compared to the Levenberg-Marquardt backpropagation algorithm and RPROP algorithm.