ARTIFICIAL NEURAL NETWORK (ANN)

KANG MIEW HOW

A project report submitted in partial fulfilment of the requirements for the award of the degree of Bachelor (Hons.) of Electronic Engineering

Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

May 2011

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature	:	______________________
Name	:	KANG MIEW HOW
ID No.	:	07 UEB 06154
Date	:	06 MAY 2011

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"ARTIFICIAL NEURAL NETWORK (ANN)"** was prepared by **KANG MIEW HOW** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of degree (Hons.) Electronic Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature	:	_________________
Supervisor	:	Mr. Lau Kean Hong
Date	:	________________

ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Mr Lau Kean Hong for his invaluable advice, guidance and his enormous patience throughout the development of the research. I am very fortunate to be blessed with the guidance from my supervisor who sharing his knowledge and experience with me. He also gave me many idea and opportunity to complete this project.

In addition, I would also like to express my gratitude to my loving parent and friends who had helped and given me encouragement. They share their ideas with me so that I can complete the tasks quickly. Besides, I also would like to thank Mr. How Yan Tar who always helping me on understanding the current knowledge of this project technology.

Finally, I would like to thanks to every single person again and wish all the best to you all. Thank you very much.

ARTIFICIAL NEURAL NETWORK (ANN)

ABSTRACT

The main development in this research is worked on the ordering of the training patterns and proposed a combined algorithm from existing algorithm for a neural network. It found to increase the generalization ability and convergence speed of neural network.

Ordering patterns of training data was found to influence the generalization ability of neural network. The best ordering patterns to learn the data was the sequential ordering plus transitional and rotational order. The results shows that the best ordering patterns helps to improve the network generalization ability at least twice compare to backpropagation algorithm. The optimal ordering patterns learning all the similar patterns of digits before proceeding to another digit. It learned one particular digit until expert and move on to another. It was similar with the human being learning process.

Besides, the combined algorithm where combining the momentum and adaptation learning rate algorithm into the backpropagation algorithm proved to shorten 50% training time compare to backpropagation method. The optimal parameter of the momentum constant, learning rate and amplifying factor H are found to be 0.1, 0.05 and 1.0 respectively. It proved that the new combined algorithm helps in convergence speed of the neural network.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Background

The human brain is probably the most complex and intelligent system in the world. It consists of the basic structural constituents known as neurons. Each neuron has a set of simple operations but in a network they exhibit complex global behavior. Artificial neural network (ANN) is an engineering approach to imitate the brain's activities.

An artificial neural network (ANN) is composed of a large number of processing elements called neurons which are connected by associated weight. The input of a neuron is modulated by a weight and then the particular neuron will add up these weighted inputs to get the net input. Next, the net input will pass through an activation function to determine its output. This behaviour follows closely our understanding of how real neurons work. The figure 1.1 has shown the basic operation of one neuron.

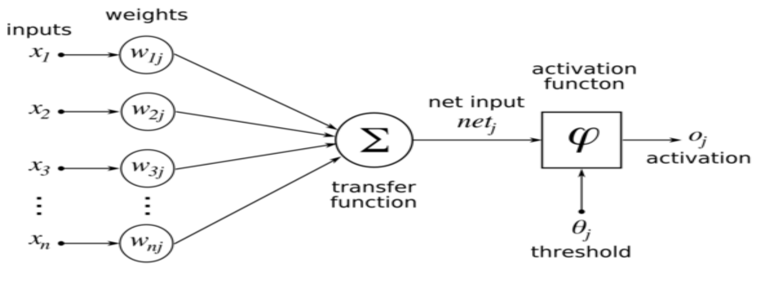


Figure 1.1: Basic operation of neuron

Each associated weight is adjusted through the training/learning process to improve recognition. Neural networks are able to learn from examples and generalize input patterns to output patterns. Neural networks can be applied to a wide variety of problems such as pattern recognition, data classification or constrained optimization problems.

1.1.1 Fundamental Feature of Neural Network

A neural network can be distinguished by its network architecture (the connection pattern between the neurons), weight-setting method (training algorithm) and the activation function used. Different types of network architecture, weight-setting method and activation function are used in different types of problems to obtain an optimization system.

1.1.1.1 Network Architecture

Network architecture can be classified into feed-forward networks and feedback/recurrent networks. Feed-forward networks are the networks in which the signals flow from the input units to the output units, in a forward direction. It can be single-layered or multilayered. The number of layers in the network can be defined to be the number of layers of weighted communication links between the slabs of neurons. For instances, single-layer network has only one layer of connection weights where the input units directly links to output units whereas multilayer network requires at least one hidden layer between input and output units. Feed-forward networks are fully connected between a neuron of one layer to another neuron from another layer but neuron in the same layer are not connected each other.

Besides, feedback/recurrent networks are the networks, in which there are closed-loop signal paths with at least one feedback loop. It has full interconnection between all units in the nets included neuron among same layer. Unlike feed-forward networks, recurrent networks do not response based purely on the inputs alone, they also make use of the internal “memory” kept in the network (resulted from a previous state or processing) to generate the outputs. Figure 1.2 shows the simple example of network architectures.

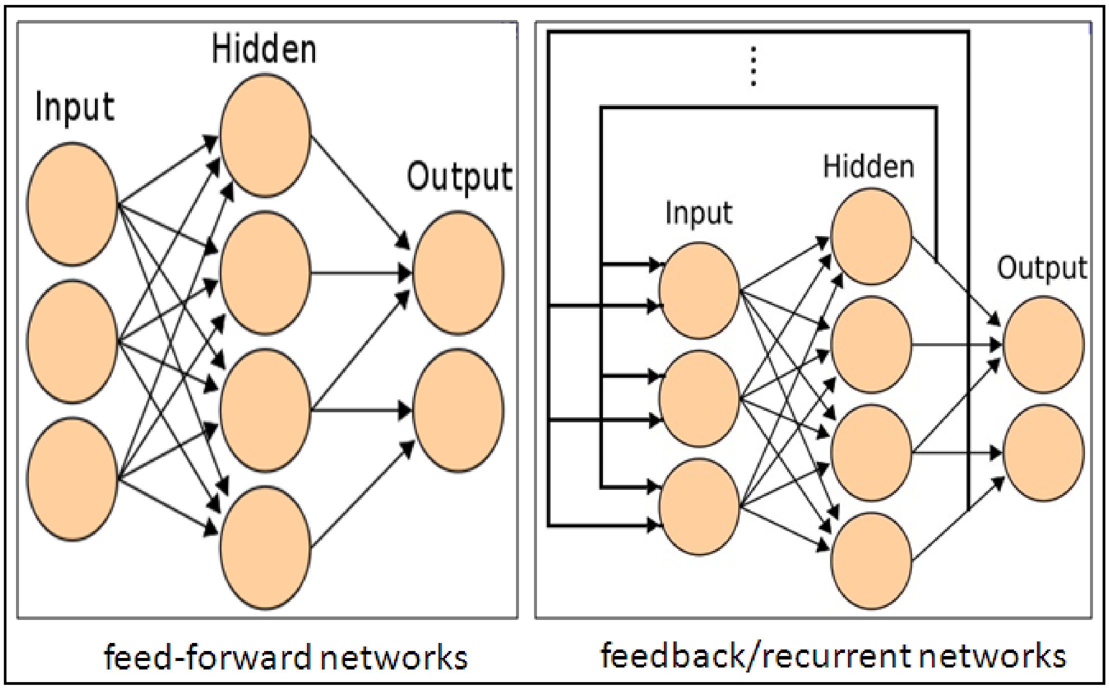


Figure 1.2: Examples of network architecture

1.1.1.2 Weight-Setting Method

Weight-setting method can be distinguished to either fixed or adaptive networks. Fixed networks use fixed-weight which set according to the optimal solution for the problem, so it does not require any iterative training. For adaptive networks, there are 2 different learning methods to change the weight. They are supervised learning and unsupervised learning.

Supervised learning incorporates an external teacher. Training is accomplished by presenting a sequence of training patterns as inputs and desired output is given during training. The weights are then adjusted to allow the network to produce answers close to the known correct answers. Supervised training used for single-layer networks is Hebb rule or delta rule while backpropagation is used to train the multilayer networks. Hebb rule stated that the weight connection between two neurons increases when they are both active simultaneously, whereas Delta rule adjusts the weights to reduce the difference between the net input and the desired output. Backpropagation is a generalized Delta rule where propagating information about errors at the output units back to the hidden units and weights adjusted to reduce these errors.

Unsupervised learning provided a sequence of input vector but no target vectors are specified. The network modifies the weights so that the most similar input vectors are assigned to the same output units. Kohonen self-organizing maps and competitive learning are example of unsupervised training.

1.1.1.3 Activation Function

Activation function used in input, hidden and output units are different depends on their needs in the particular function. Typically, the same activation function will be used for the neurons in the same layer. For input units, identity/ ramp function $f(x) = x$, for all x (net input) is used, so that activation of each unit is equal to an external input signal. For single-layer nets, threshold function is used as activation of output units which $f(x) = 1$ if $x \geq \theta$, and $f(x) = 0$ if $x < \theta$ where θ = threshold.

Sigmoid functions (*S*-shaped curves) are useful activation function for multilayer nets especially for backpropagation neural networks, because the simple relationship between the value of the function and its derivative at that point reduce the computational time during training. Logistic function, also called binary sigmoid function is an S-shaped curve with range 0 to 1. Logistic function, *f(x)* and its derivative, *f'(x)* are shown as formula 1.1. x represents the net input of the neuron.

$$f(x) = \frac{1}{1+\exp(-x)} \quad , \quad f'(x) = f(x)[1 - f\{x\}] \qquad (1.1)$$

Bipolar sigmoid function with range -1 to 1 which scaled from the logistic function also a common function used in backpropagation networks shown as below.

$$f(x) = \frac{2}{1+\exp(-x)} - 1 \quad , \quad f'(x) = \frac{1}{2}[1 + f(x)][1 - f(x)] \qquad (1.2)$$

Figure below show the common activation function used in neural networks.

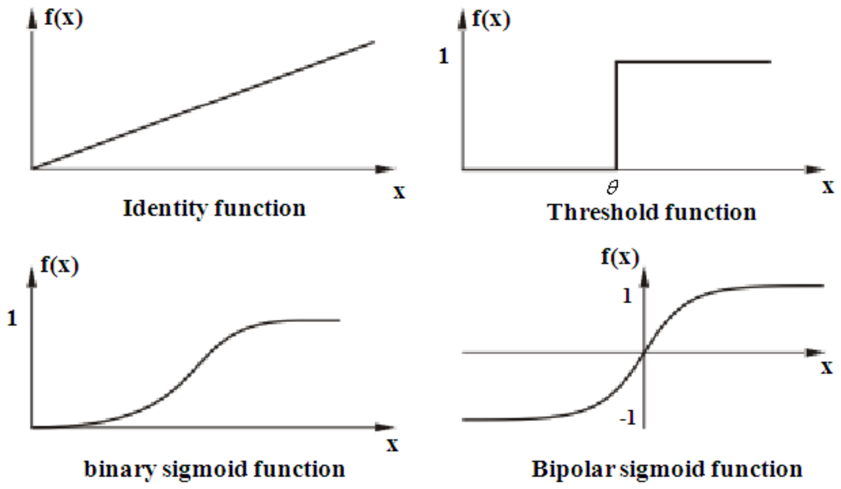


Figure 1.3: Activation Function

Different combination of network architecture, weight-setting method and activation function will result in different effect for specific problems. For example, a feed-forward multilayer network with bipolar sigmoid activation function trained by backpropagation method can used to recognize object (like alphabets). However, there is problem of error convergence for supervised learning, which is the minimization of error between the desired and computed unit values. The global minimum cannot find accurately throughout the training, therefore fail to convergence and identify a wrong alphabet. Besides, to solve more sophisticated problems as object recognition, multilayer is powerful than a single-layer networks but the training process is difficult and requires longer time.

1.2 Aims and Objectives

The aim of the project is to study and understand the basic components of neural networks, and to advance the current knowledge of this technology. The specific area of interest here are the ordering of the training patterns and the learning algorithm of neural network. It is the aim of this project to optimize the generalization ability and convergence speed of neural network.

The objectives of the project are to develop the training patterns ordering and investigate existing learning algorithm for the dynamics of the learning process, and use them to make learning faster and more generalizable.

CHAPTER 2

LITERATURE REVIEW

2.1 Backpropagation algorithm

Minsky and Papert (1988) demonstrated the limitation of the single-layer neural networks. The demonstration made people try to find a best solution in a multilayer network. However, there are not effective methods to train the hidden layers of the multilayer networks. D.E. Rumelhart and J.L. McClelland play a major role to re-emergence of neural network by proposed a forward backpropagation algorithm to train the multilayer neural network. Backpropagation method is also known as the generalized delta rule. It is simply a gradient descent method to minimize the total square error of the output computed by the networks. Nowadays, the backpropagation algorithm is the most commonly used method to train a multilayer feed-forward network.

2.1.1 Backpropagation determination

Training a network by backpropagation involves three stages which are the feed-forward of the input training pattern, the backpropagation of the associated error and the adjustment of the weights (Laurene Fausett, 1994).

For simplicity, assume the multilayer neural network with only one hidden layer. At first, the weights between each layer were initialized to small random values. Then, the feed-forward process is carried on. A training pattern is sent to the input nodes of the neural network with a desired output value given. Each input unit broadcast the signal to the units in the hidden layer. The signal will be modulated by the weight. Hidden node will then sum its weighted input signal as well as its weighted bias value received. The summed signal passes through an activation function to compute its output activation value. Furthermore, the output activation value will sent to the units in output layer. The output node processes the signal as the hidden node and produces an output value.

Next, the backpropagation of the associated error is executed. The actual output value which produced in the feed-forward process was compared to the desired output value. An error function ε is calculated by subtracted the desired output value to the actual output value. Then, the total squared error is found to be $E = \frac{1}{2}\varepsilon^2$. To minimize this total square error of the output, a gradient descent method is proposed. The gradient descent method set the weight change equal to the derivative of E for that particular weight. All the weight changes in the network are then calculated.

After that, the adjustment of the weights is carried on. A learning rate is used to determine the weight changes extent in a training time. The learning rate should be small, if not oscillating may occur and cause the convergence towards the target output failed. The weight changes are modulated by the learning rate and then add to the original weights of the neural network to improve the performance of network. After adjusting the weights in the network, a new training pattern is sent to the input nodes and operations are carried on. The training will complete after pass through all the input patterns with the weight modifications done. Figure below shows the process of the backpropagation algorithm.

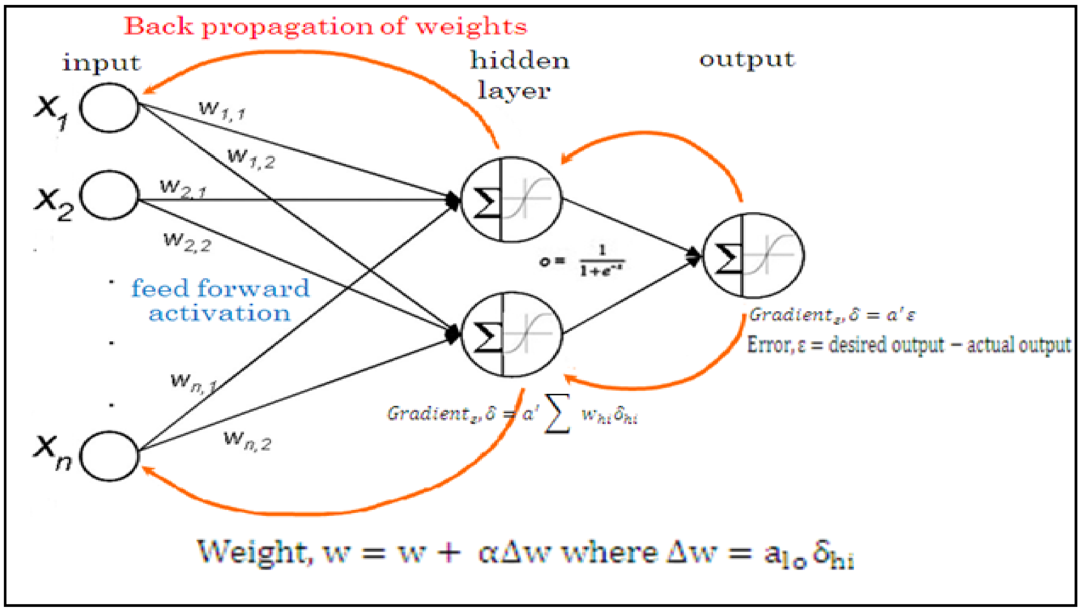

Back propagation of weights
input
hidden layer
output
x_1
x_2
x_n
$w_{1,1}$
$w_{1,2}$
$w_{2,1}$
$w_{2,2}$
$w_{n,1}$
$w_{n,2}$
feed forward activation
$o = \frac{1}{1+e^{-x}}$
$Gradient_{\varepsilon}, \delta = a'\varepsilon$
$Error, \varepsilon = desired\ output - actual\ output$
$Gradient_{\varepsilon}, \delta = a' \sum w_{hi}\delta_{hi}$
$Weight, w = w + \alpha\Delta w \text{ where } \Delta w = a_{lo}\,\delta_{hi}$

Figure 2.1: Backpropagation algorithm

2.1.2 Pros and cons and performance improvements

The backpropagation algorithm is found to train a multilayer neural network and solve for non-linear problem that are impossible in single-layer neural network. The gradient descent method used in this algorithm ensures the convergence of the output value towards the desired target. Besides, there are a number of drawbacks found in the backpropagation such as slow convergence rate, large number of hidden nodes required and global minimum problem.

However, this backpropagation had solved many complex problems such as pattern recognition and data classification which are impossible by using a single-layer neural network. The development of the backpropagation method had successfully attracting many researchers to work on the neural network to perform an intelligent task.

2.2 Layer-By-Layer Optimizing Procedures

The multilayer feed-forward network is the most widely used neural model nowadays. There have been many training algorithms proposed to train this network such as error backpropagation, delta-bar-delta (momentum) algorithm, Scalero's fast new algorithm (S.T. algorithm), Kalman filter, and quasi-Netwon based algorithm. These training algorithms shared a number of drawbacks like the slow convergence rate, large number of nodes required, local minimum problem, and time-consuming in calculating first-order derivations, etc. To conquer these problems, a faster new learning algorithm based on layer-by-layer (LBL) optimizing was proposed by Gou-Jen Wang and Chih-Cheng Chen (1996).

2.2.1 Layer-By-Layer parameter determination

For simplicity, assume that the neural network contains only one hidden layer. In this new training pattern, the optimal solution $\left(W_2^*, Y_P^*\right)$ is found initially, where w_2^* is the optimal weights of an output node and Y_P^* is the optimal output value of the hidden layer. These optimal solutions help to minimize the total sum of squared error between the desired net-input and the actual net-input.

At first, the desired output value is used to deduce the desired net-input value of an output node, which is the value just before the activation function is applied. It can be easily gotten by inverting the activation function for output node. Then this desired net-input value is used together with the output/activation values of the hidden layer to calculate the optimal weights, w_2^* between the hidden and output layer. Next, the weight w_2^* and the desired net-input are used to determine the optimal output, Y_P^* of the hidden layer.

After solving for the weight in the last-layer of the network, we go to the preceding layer and use the Y_P^* as a desired output for the hidden layer and then the desired net-input for hidden layer can be found by inversing the activation function as before. The weight w_1^* between the input and hidden layer can be determined with the same procedure as w_2^*. Here, w_1^* is the only unknown that needs to be solved in order to minimize the sum of squared error since the input layer activation value is fixed by the training pattern.

One extra work has to be done if Y_P^* goes beyond the range of activation function. For instance, bipolar sigmoid function has the output values which limited from -1 to 1. If the output value is greater than 1 or less than -1, which not achievable in this function, then it makes the inversion of the activation function unreachable. To overcome this problem, Gou-Jen Wang and Chih-Cheng Chen (1996) proposed to truncate those Y_P^* which are beyond the limitation to meet the upper or lower bound. With these Y_P^* pinned to the limits, recalculate other Y_P^* to maintain the minimization of sum of squared error.

2.2.2 Pros and cons and performance improvements

The advantage of the LBL algorithm is that the optimal pair $\left(W_2^*, Y_P^*\right)$ found help the network converge in shorten time. Besides, LBL can zoom into the optimal solution in one step per layer without the need of iterative gradient descent method. The algorithm uses a technique similar to the Kalman filter to reduce the complexity in calculating matrix inverse of high dimensions. The author discovered that replacing the sigmoid function in the output layer by the identity function in their simulation example did not result in any degradation of performance.

In contrast to the advantage of this algorithm, the LBL has some disadvantages. First, it does not solve for the global minimum solution, but just accelerating convergence. Generally, the solution is incremental, but not global as it relies on front layer random weights as the starting point to determine w_2^*, therefore it will face the local minimum problem and the optimal solution depends on the starting points. The initial parameters setting are important in this algorithm. Second, the computation time per iteration increase exponentially as the total number of node increases. If the number of node increases continuously for more complicated problem, LBL method may require longer computational time than other techniques.

The LBL algorithm makes a great improvement to the backpropagation method by performing one step zooming to get the optimal solution. The simulation results in the paper (Gou-Jen Wang & Chih-Cheng Chen, 1996) showed that the LBL produce a fastest convergence speed compared to other algorithm in the carried experiments. LBL may lead to a very small sum of squared error in the shorten time due to optimal solution is achieved in only one step.

2.3 Improved Algorithm and Architecture Networks

2.3.1 Improved backpropagation determination

In pattern recognition, the backpropagation algorithm showed that there is a state in which neural networks can learn no more patterns, in spite of there being large errors. This neural network state is called a learning standstill. In this state, the connection weights cannot be corrected, even though the output layer has a large error value. For instances, consider that a binary sigmoid function is used. A learning standstill occurs when all value on one layer are nearly 1 or 0. The binary sigmoid function curve is nearly flat or horizontally at value close to 0 or 1, therefore the changes of connection weights are invisible or close to zero. Thus, the neural network learns no more patterns.

In order to evade the learning standstill, Yamada K., Kami H., Tsukumo J. and Temma, T. (1989) proposed an improved learning algorithm where the slant parameter of the activation function is controlled. Initially, a serious error is judged. If the error greater than a threshold value (0.5 is used in this paper), then the slant parameter is enlarged double to make the curve more diagonal, so that output values on any layer are far from 0 and 1. Then all output values for every layer are recalculated. Process is repeated until the error judged is less than the threshold value. After solving this learning standstill problem, the slant parameter is set to initial value again.

2.3.2 Three distinct architecture neural networks

In order to investigate the neural network ability to extract feature for characters, to discriminate between features and to classify them into correct classes, three distinct architecture neural network are used to observe these conditions in the paper (Yamada K., Kami H., Tsukumo J. & Temma, T., 1989).

First one is globally-connected neural network, into which a grey level character images as input. It is a basic multilayer neural network, in which each hidden unit is connected to all inputs units. Second is locally-connected neural network, into which a grey level character images as input, but each hidden unit is connected to input units included in a local area of a character. Each local area in the network should not define too closely since the feature extraction is self-organized, so it may cause oversensitive to positions for the local features. Last one is feature input neural network, into which contour feature vectors to be extracted from character are input. Figure 2.2 shows the architecture of these 3 neural networks.

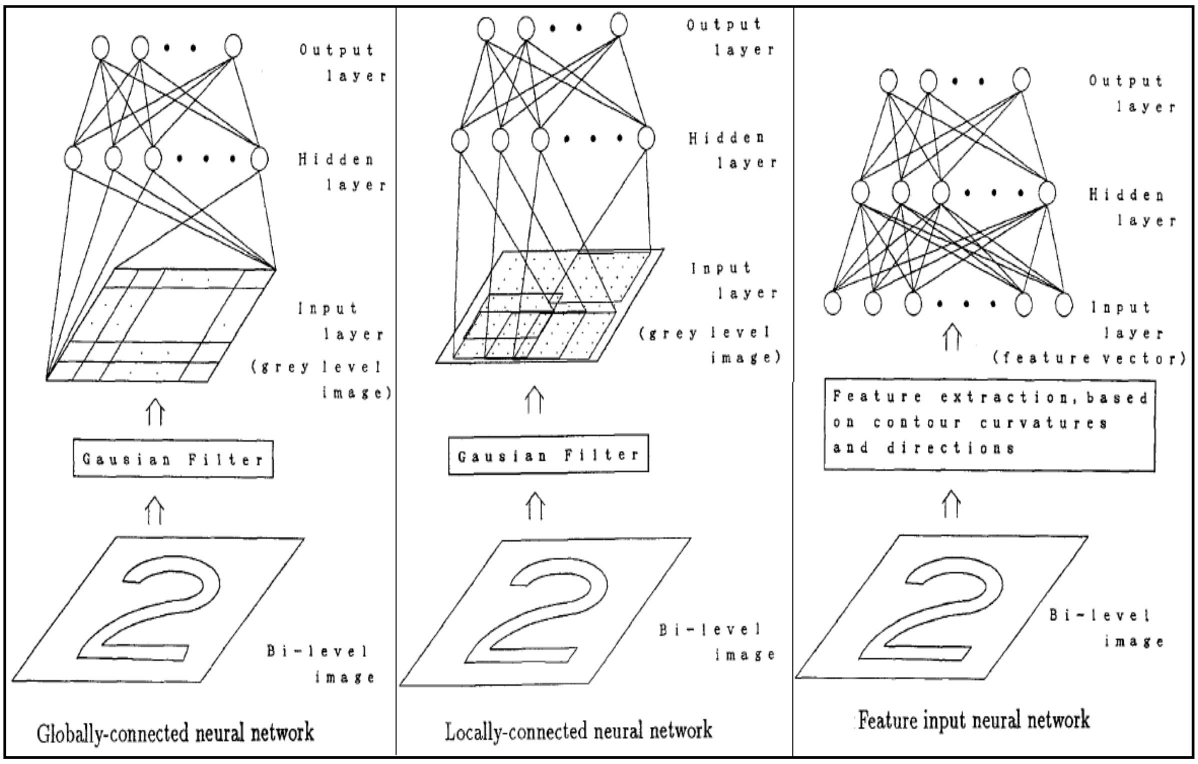

Output layer
Hidden layer
Input layer (grey level image)
Gausian Filter
Bi-level image
Globally-connected neural network
Output layer
Hidden layer
Input layer (grey level image)
Gausian Filter
Bi-level image
Locally-connected neural network
Output layer
Hidden layer
Input layer (feature vector)
Feature extraction, based on contour curvatures and directions
Bi-level image
Feature input neural network

Figure 2.2: Three distinct architecture neural networks

In the globally-connected neural network, each hidden unit can response to a global feature of a character, so they extract feature in the entire area of a character. However, locally-connected extract feature of a character locally and recognize a character by combining local features. From these different extraction methods, locally-connected network was found that can perform extraction of the global feature more flexibly than the globally-connected network, because it can find a global feature, even if the local feature positions are relatively changed.

Unlike the grey level input images of globally-connected and locally-connected structure, the feature vectors extracted from a character is inputted to the feature input neural network. A contour feature is calculated based on the direction and curvature for a character contour. This feature is a kind of conventional pattern recognition method used as inputs to a neural network, and whose classification process is carried out by a neural network.

2.3.3 Pros and cons and performance improvements

The improvement performance in this paper is that the improved backpropagation algorithm is evaded a learning standstill. This improved algorithm is effective for letting a neural network learn all the trained patterns, which cannot be learned by backpropagation, and to hasten learning speed by three times. By using this improved algorithm, the simulation graphs show in the paper (Yamada K et al., 1989) proved that the greater the number of hidden units, the higher the recognition rate can achieve.

The next improvement performance is the locally-connected structure had the higher recognition rate compared to globally-connected structure. The reason is that if parts of character change slightly, other hidden units in the locally-connected networks respond to the changed parts and their combination often enables the character to be recognized correctly, but the globally-connected neural network may fail to extract the same feature. These considerations prove that it is important to construct an innate structure in a neural network, in order to improve the feature extraction ability.

On the other hand, there are some disadvantages of the improved algorithm. When the number of hidden node is small, the recognition rate of the improved algorithm is lower than the standard backpropagation algorithm. It showed that this improved algorithm is not so suitable for the applications where a small number of hidden nodes required. Then, for the feature-input structure, where it is impossible to recognize a character correctly, the feature vectors may be confused with other categories. Furthermore, the feature extracted is not always optimal for a recognition target, so it will affect the discrimination of the neural network and produce the mis-recognized results.

2.4 Adaptation Learning Rate Algorithm

Backpropagation algorithm solved the problem of training the weight of a hidden neuron by using gradient descent method. This gradient method uses a fixed learning rate. Learning rate denotes the weights changing extent. In case, if the learning rate set to be larger, the learning speed is faster but it may cause the oscillating problems. In contrast, if the learning rate is smaller, the learning process is more stable but the learning speed will be slower. A fixed learning rate sets to gain a balance between the learning rate and the learning progress stability, which is normally small. It causes a slow convergence rate. Chao Yang and Ruzhi Xu (2009) had proposed a new adaptation learning rate algorithm in this paper to increase the convergence speed with no oscillating in the learning progress.

2.4.1 Adaptation learning rate determination

Adaptation learning rate algorithm (Chao Yang & Ruzhi Xu, 2009) make the learning rate changes as the error value calculated changes, through the whole learning progress of a neural network. In this new algorithm, the training method is same as the backpropagation algorithm, and the only improvement in this algorithm is replaced the fixed learning rate by an adaptation learning rate. This new adaptation learning rate algorithm can be expressed as equation 2.1.

$$\eta = \eta_o \exp(h \cdot E) \tag{2.1}$$

where
E = mean square error when one iteration is finished
η = learning rate for the next iteration
η_o = initial learning rate, $\eta_o > 0$
h = amplifying factor, $h > 0$

Initially, a learning rate is selected as a very small value. Therefore, when the error value close to zero, the learning rate turns to its initial rate due to $\exp(0) = 1$ and only if the initial rate is very small to guarantee no oscillating in the later period of the learning progress. In the first stage of the learning progress, the error value is larger so the learning rate is enlarged by the adaptive algorithm equation to accelerate the convergence speed. As the learning proceeds, the error is getting smaller and the amplified rate is reduced exponentially. Then, at the end of the training process, where the error value is close to zero, the learning rate turns to the initial rate. This expression was accelerated the learning speed in the starting of training progress and performed a small step improvement towards the target at the end of training, as the fixed learning rate. It had descended the error value faster in the condition of the stability of learning progress.

Another important parameter in this algorithm is *h*, the amplifying factor. The parameter *h* can control the amplified extent, therefore if the value is not properly set, the learning rate turns to be too large to cause the oscillating happens. The error value can be calculated from the backpropagation method.

2.4.2 Pros and cons and performance improvements

The advantage of the adaptation learning rate algorithm is that it can be easily expressed by getting the fixed learning rate multiply with an exponential function of the error, so it can easily plugged into the backpropagation algorithm to get widely used (Chao Yang & Ruzhi Xu, 2009). This new algorithm helps to improve the convergence speed. It minimizes the error value faster, and at the same time maintaining the stability of the learning progress. Besides, this algorithm is able to zoom towards the desired target at the beginning of training and moving a small steps to approach the target at the end of training.

The disadvantage of this new algorithm is the parameter setting of *h*, the amplifying factor is not taught clearly. From the formula 2.1, we know that setting *h>0* but there did not state any proper range for this parameter. The large value of *h* will turn the amplified rate to be too large and cause an oscillating problem.

The improvement of this paper is used a new equation for the learning rate rather than just using a fixed learning rate. The expression $\eta = \eta_o \exp(h \cdot E)$ used will increase the convergence speed of the backpropagation algorithm. The simulation experiment showed that the adaptation learning rate converge faster than the fixed learning rate at the beginning of the training process, and when the error is reduced to a very small value, which smaller than 0.1, then the convergence speed is slowed down and close to the fixed learning rate.

2.5 Synthesis Algorithm by Growing Layers Using Training Results

The nature of the neural network architecture is a very important consideration when it is concerned about the optimal trainability and generalization. However, backpropagation algorithm has no prior knowledge about the number of required hidden layers and neurons. An additional investigation is required to carry on before starting a backpropagation algorithm.

At present, there are two approaches to determine the optimal network architecture. The first approach is starting with a large number of hidden nodes and then cut down the network whenever it is possible. Second is generated the required neural network architecture dynamically as part of the learning process. It starts with a small network and grows the additional neurons or layers only when it is necessary. Mezard’s tiling algorithm is an algorithm which adds the hidden layers and neurons in each layer only when necessary, at will until convergence and system error reduced to zero after the network is fully trained. However, this tiling algorithm only suitable used for one output neuron problems. In this paper, Yu X., Loh N.K. and Miller W.C. (1993) proposed a synthesis algorithm to generalize the tiling algorithm to several output neurons and improve the method of selection for the target output values of the ancillary neurons.

2.5.1 Synthesis algorithm determination

At first, the new synthesis algorithm is started with a simple architecture where only have X input nodes and Y output nodes. A pocket algorithm is used to train this network until the convergence reached. It stores the best weight vector so far in a “pocket” while continuing to learn. In this new algorithm, the activation function of each neuron can be hard limiting such as step function instead of sigmoid function. Once the convergence is reached, the training patterns are classified into groups according to the values of actual output patterns. If each group corresponds to only one target value where actual the output pattern equals to the corresponding target output pattern, the resultant trained neural network represents the final required architecture. Otherwise, the number of different target output values in each group is counted and then the maximum number of different target output, M is set. Then, number of ancillary neurons, N= $\log_2 M$ are added to the layer.

Next, the target values of these added ancillary neurons are defined such that those input patterns in each group corresponding to different final target values must have different output values. Yu X. et al. (1993) stated that the ancillary neurons had to define correctly in order to ensure any two patterns in a layer with distinct target outputs have distinct internal representations. After defining the target value of ancillary neurons, the pocket algorithm is repeated again. The verification process is repeated until each group corresponds to only one target value, and then this layer is considered completed and proceeds to the next layer. This next layer used the previous layer with X+N nodes as input and Y output nodes as the original target output patterns. Then the algorithm continues with the operation defined previously until the correct target output pattern produced. This synthesis algorithm is guaranteed to converge since the subsequent layers decrease in size.

2.5.2 Pros and cons and performance improvements

The advantage of the synthesis algorithm is the network architecture is generated dynamically during the learning process, so there is unnecessary to estimate the number of layers and hidden neurons before training. From the simulation results carried by Yu X. et al. (1993) showed that the learning speed is faster than backpropagation algorithms and the system error is equal to zero when the neural network is fully trained. Besides, this algorithm can classify both linear separable and linear non-separable families, whereas the backpropagation algorithm will fail sometimes. In advance, it uses a hard limit activation function instead of continuous sigmoid function. Hence, it is suited for a VLSI implementation.

Disadvantage of this synthesis algorithm is that it does not generate architecture with the fewest number of neurons possible. As the number of output nodes increases, the selections of the target output values for the ancillary neurons may be difficult. Besides, the training patterns need to classify into groups according to the values of the actual output pattern. It may take a longer time for a larger number of training patterns, and make this algorithm not reliable.

The improvement in this synthesis algorithm is a neural network can generate the network architecture without estimating the number of hidden layers and hidden neurons as what the backpropagation method does. Besides, the synthesis algorithm produces a zero error neural network while the minimization of the backpropagation method is not guaranteed to converge to an absolute minimum with zero error.

CHAPTER 3

METHODOLOGY

3.1 Research Methodology

In order to test the ordering effects of the training patterns and improve the training algorithm of the neural network, digit recognition was chosen as a sample problem in this research. A supervised learning was used in this research, so that the optimal ordering patterns could be developed. Supervised learning was accomplished by presenting a series of training patterns to neural network, each with an associated target output vector. Hence, data should be collected with associated target before develop the neural network program.

Initially, creating a completed neural network that does digit recognition would be too comprehensive for a research of this size, therefore it is necessary to place some restrictions on it. In data collecting process, 4 digits which are 0, 1, 2, and 3 were collected as training samples. The collected images included distinct scaled and skewed patterns. Each digit collected 100 sample images and the dimension of image size was set to 8x8. These restrictions help to simplify the testing on data ordering issues. Some of the ordering patterns were deal with digits interleaving, hence considering 4 different digits in the data set may create enough interleaving patterns and it is not too complicated as using whole chain of digit from 0-9. Besides, using only 4 digits and small dimension size may help to shorten the training time of the neural network, and at the end save time for research progress.

In advance, this data set was collected from ASCII Art Generator software. ASCII Art Generator can convert digit from different font style into an image, such as Time New Roman, Arial and etc. Furthermore, the software also provided free draw function to the user. Hence, data could also be collected by drawing the digits directly on the software and converted it into image.

Next, neural network program was developed. The developed network consisted of 64 (8x8) input nodes and 4 output nodes. Since the program is interested in the classification of the digits, so 4 grandmother output nodes are used where one and only one output node responds to a certain character. This neural network was implemented in C++ programming as stated by the supervisor and it can compile easily in Visual Studio 6.0.

By Homik-Stinchcombe-White theorem, one hidden layer neural network is sufficient to approximate any measureable mapping from a finite dimensional vector to another, provided sufficiently many hidden nodes in the layer. Hence, a multilayer feed-forward neural network with one hidden layer was developed in the research. This neural network was chosen to be globally-connected, in which each hidden unit is connected to all inputs units. Figure 3.1 shows the globally-connected neural network. Although locally-connected neural network was performed better than globally-connected neural network (Yamada K., Kami H., Tsukumo J. & Temma, T., 1989), but the local area in such a network must define precisely to avoid oversensitive which would lead the system to a terrible performance. Therefore, globally-connected architecture was chosen as the network architecture.

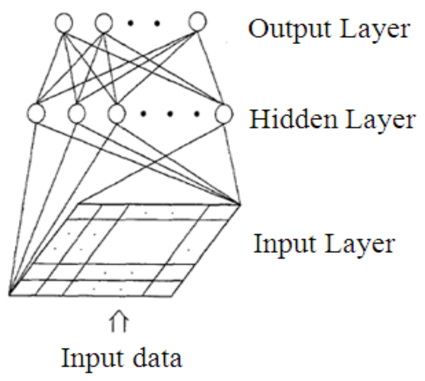


Figure 3.1: Globally-connected neural networks

A proposed algorithm was developed based on the investigation of existing supervised learning algorithm. The most universal algorithm implement in a neural network is backpropagation algorithm. However, it required a long training time due to slow convergence rate and large number of hidden node used. To overcome these drawbacks, many new algorithms were published. Momentum algorithm (M. Tanvir Islam & Yoichi Okabe, 2003) and adaptation learning rate algorithm (Chao Yang & Ruzhi Xu, 2009) are the techniques used to speed up the training process of backpropagation algorithm.

Momentum is basically a technique which enlarges the weight movement if there is a same direction change as previous step whereas keeps a small step for the different direction changes. It helps to improve the convergence speed in training process. Moreover, adaptation learning rate is the technique where the learning rate is exponentially proportional to the mean square error. It means that the learning rate are enlarged at the beginning of the training process, and then becomes smaller and smaller until hold as the initial learning rate at the end. It improves the convergence speed as what momentum algorithm does.

Both algorithms can apply easily into an existing backpropagation program by changing the weight-updating formula only. Hence, the proposed algorithm is a combination of backpropagation with momentum and adaptation learning rate method to increase the speed of convergence. Analysis was carried out to estimate the optimal value of the constant parameter in momentum and adaptation learning rate method to shorten the training time. Figure 3.2 below shows how the proposed algorithm formula comes from the existence formulas.

$$\omega = \omega + \Delta\omega$$

Back propagation

$$\Delta\omega = \alpha . a_{lo} . \delta_{hi}$$

$$\begin{aligned} where\ \alpha &= learning\ rate \\ a_{lo} &= activation\ output\ of\ lower\ layer \\ \delta_{hi} &= error\ gradient\ of\ higher\ layer \end{aligned}$$

Momentum

$$\Delta\omega = \alpha . a_{lo} . \delta_{hi} + \beta . \Delta\omega\ (t-1)$$

$$\begin{aligned} where\ \beta &= momentum\ constant \\ \Delta\omega\ (t-1) &= \text{previous state of weight change} \end{aligned}$$

Adaptive learning rate

$$\Delta\omega = \alpha_0 \exp(h\,.\,E)\,.\,a_{lo} . \delta_{hi}$$

$$\begin{aligned} where\ \alpha_0 &= initial\ learning\ rate \\ h &= amplifying\ factor \\ E &= mean\ square\ error \end{aligned}$$

Proposed algorithm

$$\Delta\omega = \alpha_0 \exp(h\,.\,E)\,.\,a_{lo} . \delta_{hi} + \beta . \Delta\omega\ (t-1)$$

Figure 3.2: Derivation of proposed algorithm

Furthermore, bipolar sigmoid function was used as the activation function in the proposed algorithm. This function was slanting than binary sigmoid function. Thus, the slant modification method (Yamada K. et al., 1989), in which modifies the slant parameter of binary sigmoid function to evade learning standstill condition, can be ignored in this new methodology development.

After built the neural network program, analysis on the ordering of training patterns was carried out. First, the ordering of training patterns was developed and then divided into certain characteristics. Each ordering characteristic was executed 5 times and average result was calculated to keep away from bias. The training data used in this research was fixed except the ordering sequences altered. After all, comparison of all the ordering patterns was accomplished and a best ordering pattern was chosen. This best ordering pattern was then set as optimal order to the network input.

Subsequently, optimization of the parameters in the training algorithm was carried out. Optimal value of momentum constant, learning rate and constant parameter H in adaptation learning rate were found to improve the convergence speed in learning process. At the end, comparison between initial and optimal condition was made to show the total improvement of the network. Flowchart shows in figure 3.3 is the overall methodology to carry out the research.

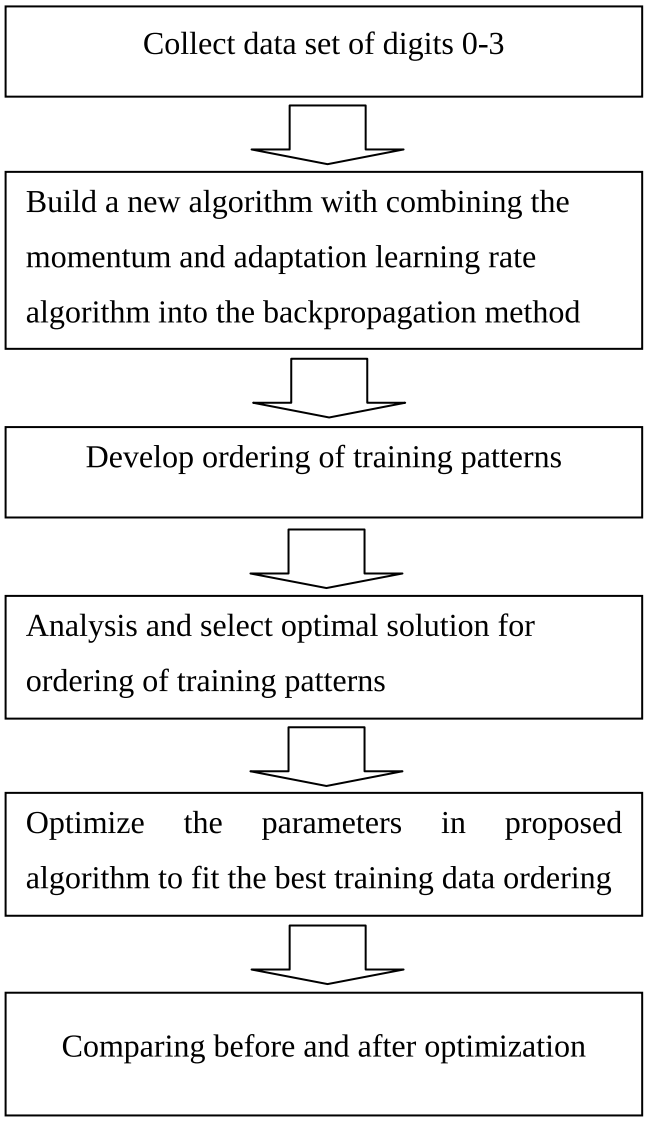

Collect data set of digits 0-3
Build a new algorithm with combining the momentum and adaptation learning rate algorithm into the backpropagation method
Develop ordering of training patterns
Analysis and select optimal solution for ordering of training patterns
Optimize the parameters in proposed algorithm to fit the best training data ordering
Comparing before and after optimization

Figure 3.3: Flowchart of overall methodology

3.1.1 Data set

The data set consisted of 400 elements, where each digit had 100 elements. It was then split into 3 parts with 60%, 20% and 20% for the training set, generalization set and validation set respectively. The splitting data were then stored into 3 different input files respectively. The figure 3.4 shows the splitting of the data set.

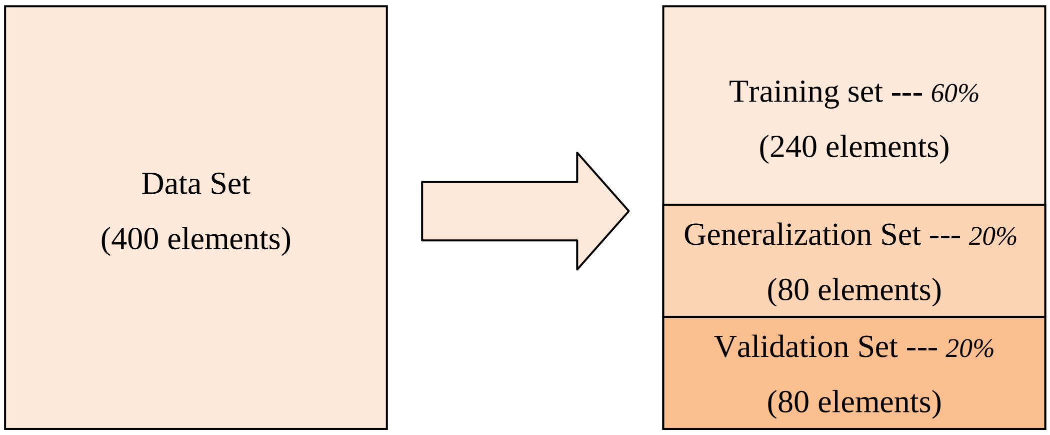


Figure 3.4: Splitting of data set

Training set is the data used to train the network and update the weights, so it must be the largest chunk among them. Generalization set is the data used to run through the neural network at the end of each epoch to see how well the network handles with unseen data while validation data set will only run through the network once the training has completed to give the final validation result. Both generalization and validation data set are used to avoid the over-fitting problem, where the network starts to memorize the input data and fail to handle the unseen data correctly. Thus, the data of generalization and validation set must include all the possible patterns of the digits such as scaled, skewed, transited and rotated images to test how well the network can handle the data in all possible directions.

3.1.2 Operating principle of proposed algorithm in C++ program

Initially, the neural network program is set up with loading the network configuration parameters into the program. After establish the network, the data input files are loaded into the program. These data input files include training, generalization and validation data set are then stored into arrays respectively.

Afterwards, program creates a series of random initial weights to both input-to-hidden and hidden-to-output weights. These initial weights are set to be the value from a random distribution between -1 to 1. It can be reached by using formula 3.1 below. The ratio of *rand( )/RAND_MAX* will create a value between 0 to 1. This ratio is then multiplied by 2 to create the value range from 0 to 2. Lastly, this value is subtracted from 1 to obtain the value range between -1 to 1.

$$Weight = \left(2.0 \times \frac{rand()}{RAND\_MAX}\right) - 1.0 \tag{3.1}$$

where
rand( ) = rand function which generate an integer between 0 to RAND_MAX
RAND_MAX = a symbolic constant defined in the <stdlib.h> header file

These initialized input-to-hidden weights are then adjusted through Nguyen-Widrow normalization. This normalization makes each hidden node linear over only a small interval of input-to-hidden weights. The magnitude of input-to-hidden weights are adjusted to 0.7*S to provide some overlapping between the intervals. The slices of interval, S is equal to the number of hidden nodes divide by number of input nodes. By using this normalization method, the network can achieve a lower mean square error in a much shorter time. It is used to improve the network's training speed.

Next, the neural network's training process is started. The training data are sent into the input layer. These training patterns are sent one at the time. It is then gone through the feed-forward process. The inputs are modulated by the weights. Then, these weighted inputs are added up and passed through the activation function to produce outputs for each hidden node. This process is repeated in hidden-to-output layer to determine each output node's value. Bipolar sigmoid function is used as the activation function. Its formula can refer to formula 1.2.

At the end of the feed-forward process, actual output values are produced. Followed by backpropagation process, the error of each output node is calculated and back-propagated to the network. Moreover, weights are updated to minimize the network error and at the same time produce the desired output. Error is defined as the different between target and actual output value which show in formula 3.2. When errors are propagated back to the network, error gradients or deltas for each output and hidden node are calculated. The deltas formula for output nodes and hidden nodes is shown in formula 3.3 and 3.4 respectively.

$$\varepsilon = t - a \tag{3.2}$$

where
ε = error of the neural network
t = target output value
a= actual output value

$$\begin{aligned}\delta_{output} &= f'(x) \times \varepsilon \\ &= \tfrac{1}{2}\,(1+a)(1-a)(t-a)\end{aligned} \tag{3.3}$$

where
δ_{output}= error gradient or delta for particular output node
f'(x) = inverse of activation function shows in formula 1.2
ε = error between target value, t and actual output, a

$$\delta_{hidden} = f'(x) \times \sum(\omega_{HO} \times \delta_{output}) \tag{3.4}$$

where
δ_{hidden}= error gradient or delta for particular hidden node
δ_{output}= error gradient or delta for particular output node
f'(x) = inverse of activation function shows in formula 1.2
ω_{HO} = hidden-to-output weight for particular output node

These error gradients or deltas are then used to update the weights in the neural network. To update the weights, a new weight-updating formula is formed here with adding the momentum and adaptation learning rate algorithm. The new weight changes formula is shown in formula 3.5. It is just a combination formula of backpropagation, momentum and adaptation learning rate algorithm. Then, the weights are simply updated by $\omega = \omega + \Delta\omega$ formula.

$$\Delta\omega = \alpha_0 \exp(h\,.E) \times a_{lo} \times \delta_{hi} + \beta \times \Delta\omega\ (t-1) \tag{3.5}$$

where

$\Delta\omega$= weight change

α_0= initial learning rate

E = mean square error when one iteration or epoch is finished

h = amplifying factor, $h > 0$

a_{lo} = output value of lower layer

δ_{hi} = error gradient or delta of higher layer

β= momentum constant

$\Delta\omega\ (t-1)$ = previous state of weight changes

After updating the weights, next training pattern is sent into input nodes and operations as previous are carried out again. When all the input patterns are passed through the neural network, one epoch is completed. At that moment, generalization data set are then run through the network to see how well the network performs. Subsequently, whole training process is continued by repeating previous process all over again until it reached the maximum setting value of the epoch. At that time, the validation data set are running through the program to get the final validation results.

Furthermore, a new set of random initial weights are generated for second rerun and whole training process is repeated until the maximum rerun condition reached. The rerun process used to ensure the average output result is fair and no bias to any initial weights. Therefore, the average output result of the rerun process is then used to show the overall neural network performance. Figure 3.5 below is the flowchart of the neural network program.

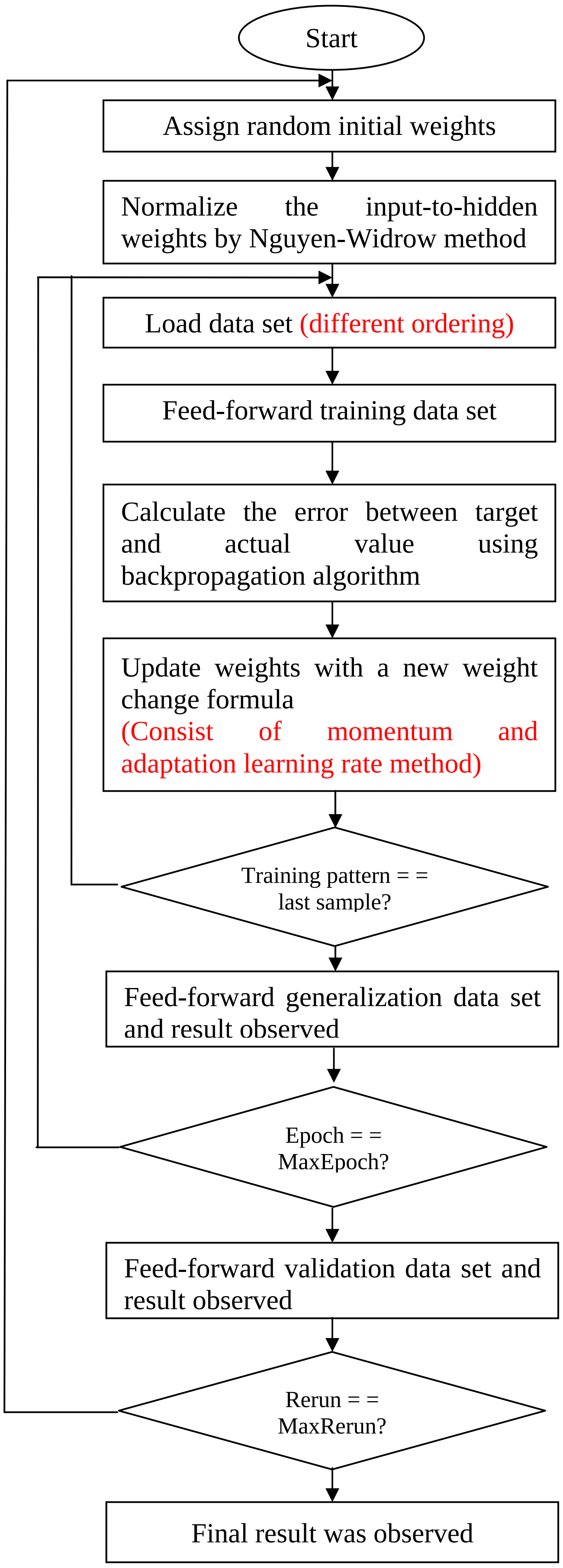

Start
Assign random initial weights
Normalize the input-to-hidden weights by Nguyen-Widrow method
Load data set (different ordering)
Feed-forward training data set
Calculate the error between target and actual value using backpropagation algorithm
Update weights with a new weight change formula (Consist of momentum and adaptation learning rate method)
Training pattern = = last sample?
Feed-forward generalization data set and result observed
Epoch = = MaxEpoch?
Feed-forward validation data set and result observed
Rerun = = MaxRerun?
Final result was observed

Figure 3.5: Flowchart of neural network program

3.1.3 Ordering of training patterns

Development on the training data ordering patterns has a great meaning in sequential supervised training. It is because improper order of elements in the training process may lead to terrible interference. Additionally, this mechanism can also occur during each training epoch and disturb the networks from learning more patterns. The worst case may happen where the connection weights cannot be corrected, even though the output layer has a large error value. Therefore, finding an optimal ordering of training data patterns are important. With a good ordering of training data, the network can minimize the final validation error. Hence, it may easily lead to recognition success.

After splitting the data set, generalization and validation set were fixed. This ordering analysis was only dealt with training data set, because it is the only data set passing through the supervised training.

In this project, several ordering approaches to the training data were proposed and experimentally evaluated. The ordering sequences were divided into 5 main parts, which are random, sequence, interleaving sequence, transition, and rotation order. However, transition and rotation order could not stand alone without any supported order, thus they were combined with the sequence and interleaving sequence to test their ordering effects. On the whole, 7 cases of ordering were tested in this project. Table below shows the 7 cases of data patterns ordering.

Table 3.1: 7 Type of data patterns ordering

Case	Condition
0	Random order
1	Sequence order
2	Interleaving sequence order
3	Sequence + transition order
4	Interleaving sequence + transition order
5	Sequence + rotation order
6	Interleaving sequence + rotation order

Random order was used as the reference point for this ordering analysis because it is the general order implement in the input network nowadays. All the training data are randomize distribution through whole training sequence.

Besides, sequence order was distributed the training data digit by digit over entire training sequence. After training all the patterns of one digit, the network will move on to train another digit patterns and so on. The data patterns between each digit were out of order. Furthermore, interleave sequence order was interleaved the data between digits. This interleaving digits model was then repeated with different training patterns through entire training process. The data patterns between each interleaving model were out of order.

Apart from that, the 4 other types of data ordering were based on the transition and rotation order. As a result of these two orders could not stand alone, sequence and interleaving sequence order were then merged into them. Transition order was dealt with the alignment of the image, such as left, centre and right, while rotation order was arrange with the sequence of normal to italic image or vice versa.

3.2 Planning

Table 3.2 shows the project planning for the final year project and Figure 3.6 shows the project progress in chart. The whole project was divided into small piece of tasks and then planning time to accomplish each task was set. Therefore, the project can be finished smoothly and on time.

Table 3.2: Project Planning Table

ID	Task Name	Duration	Start	Finish
1	Understanding about neural network	14 days	6 Jun 10	19 Jun 10
2	Study on backpropagation	14 days	20 Jun 10	3 Jul 10
3	Analysis other algorithm of neural network	21 days	4 Jul 10	24 Jul 10
4	Report Writing - Analysis and Develop new algorithm	28 days	25 Jul 10	21 Aug 10
5	Finalisation Progressive Report	7 days	22 Aug 10	28 Aug 10
6	Sample image collection	28 days	17 Oct 10	20 Nov 10
7	Developing the new program	14 days	31 Oct 10	13 Nov 10
8	Testing and Investigation on new program	21 days	14 Nov 10	4 Dec 10
9	Develop a ordering of training patterns	7 days	5 Dec 10	11 Dec 10
10	Testing for the ordering of training patterns	14 days	12 Dec 10	25 Dec 10
11	Analysis for the ordering of training patterns	35 days	26 Dec 10	29 Jan 11
12	Improvement on the new algorithm program	49 days	30 Jan 11	19 Mar 11
13	Final Testing	14 days	20 Mar 11	2 Apr 11
14	Final Report	28 days	3 Apr 11	30 Apr 11

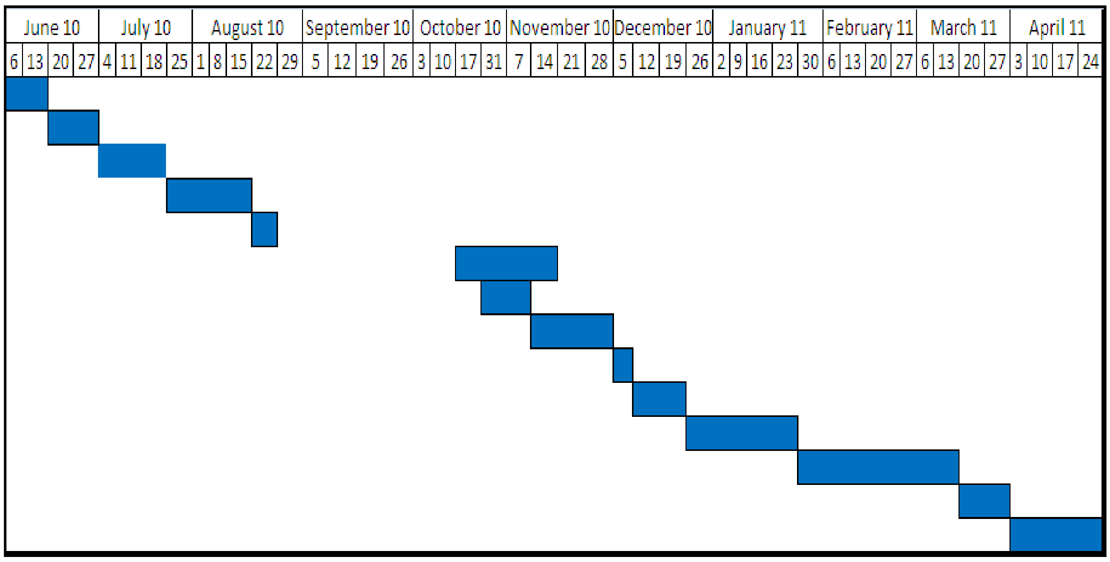


Figure 3.6: Project progress

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Network Setup

In this research, two main optimization studies were involved, where optimize the ordering of training data and training algorithm parameters. Before proceeding to these studies or analyses, data set for digits 0 to 3 was collected and network program was built. Figure below shows a few sample of the data set used in this research. Then, network configuration was set up and ordering of training patterns was generated.

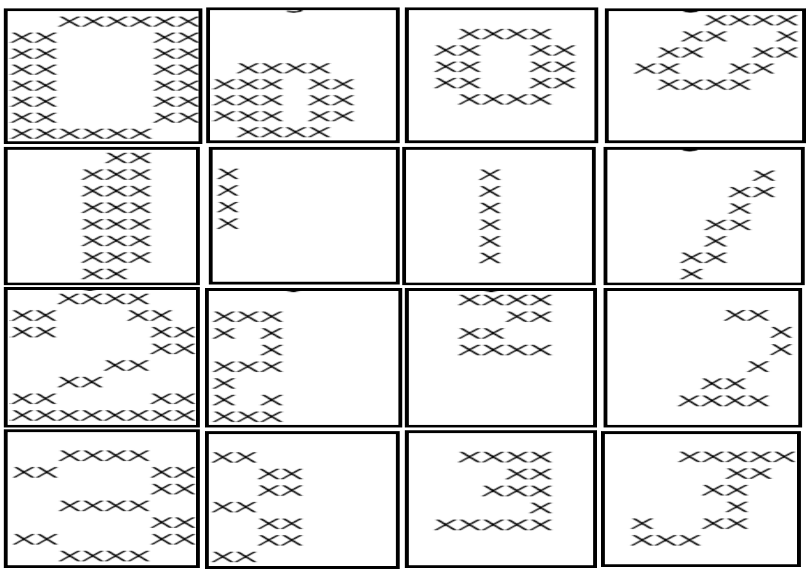


Figure 4.1: Input samples

4.1.1 Results and discussions of network configuration

Before starting the study of ordering patterns, configuration of neural network program was set up. Number of hidden node, epoch value, rerun value and etc were defined in this moment. The ordering of training data used in this setting are not optimized since the setting only used to achieve the basic performance of the network.

First, number of hidden node was set to 11. It is because if the program has very less hidden nodes such as 6, it may not have sufficient memory place to classify the digits. On the other hand, if the hidden nodes are too much such as 20, it will waste the memory place and somehow memorizing the data rather than classifying them. Therefore, a moderate number, 11 was used in this program.

The initial learning rate of the program was set to 0.2. The same case happens here where value set should be moderate. If learning rate set to a large value like 0.8, the network results may oscillate and distort the network stability. However, if learning rate set to very small value like 0.02, then convergence speed of the network will be very slow. A balance value, 0.2 was chosen as the initial learning rate.

Next, epoch value which reflected the number of looping used in the training process was considered. Training loop should end after completion of training process where system learnt no more patterns. It can be identified by the saturation of the network results. To determine it, epoch was first set to a huge value and then graph for training accuracy and generalization accuracy versus epoch were plotted. Figure 4.2 and 4.3 show the training accuracy versus epoch and generalization accuracy versus epoch respectively.

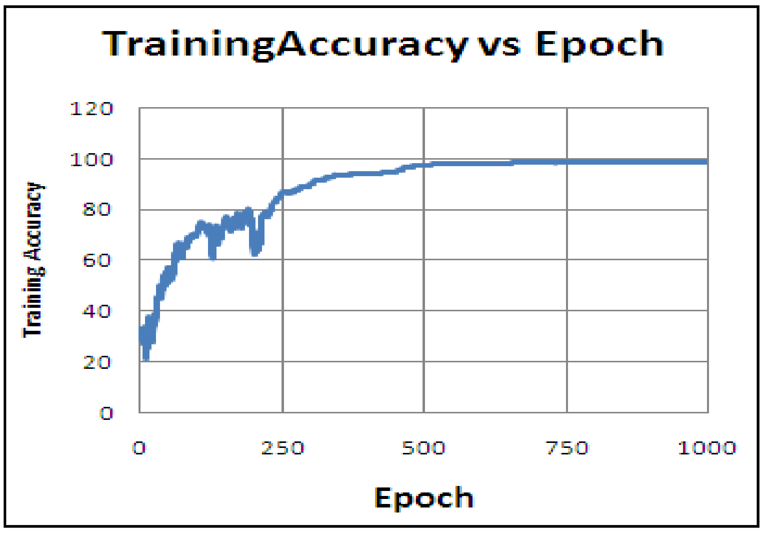


Figure 4.2: Training accuracy vs Epoch

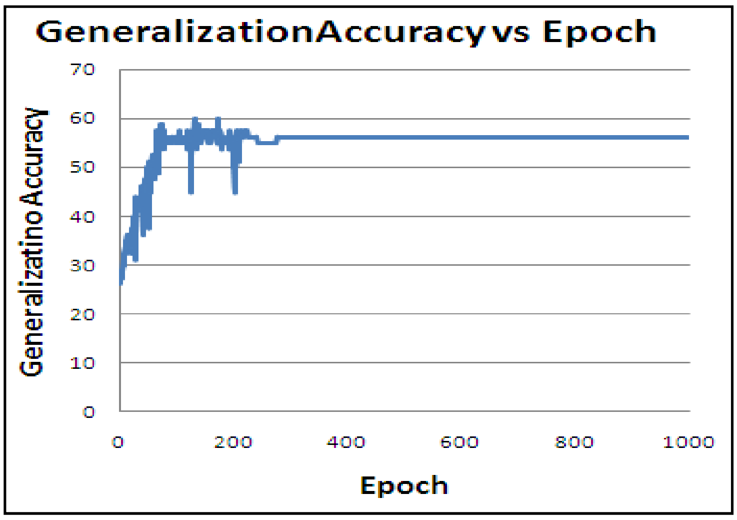


Figure 4.3: Generalization accuracy vs Epoch

Observation on these graphs found that the training accuracy was going to be constant at epoch equal to 500, and the generalization accuracy saturate at epoch value near to 300. From this observation, when epoch value equal to 500, both training and generalization accuracy reached their saturation state. Therefore, the minimum epoch required to complete the training process was set to 500.

In addition, rerun value was considered. Rerun condition ensures the results produced are fair to any initial weights. To determine this rerun value, the method used previously was repeated. Rerun value was set to a large value first, and then graph of training accuracy versus rerun was plotted. Figure 4.4 below shows the training accuracy of the rerun parameter over 1000 times.

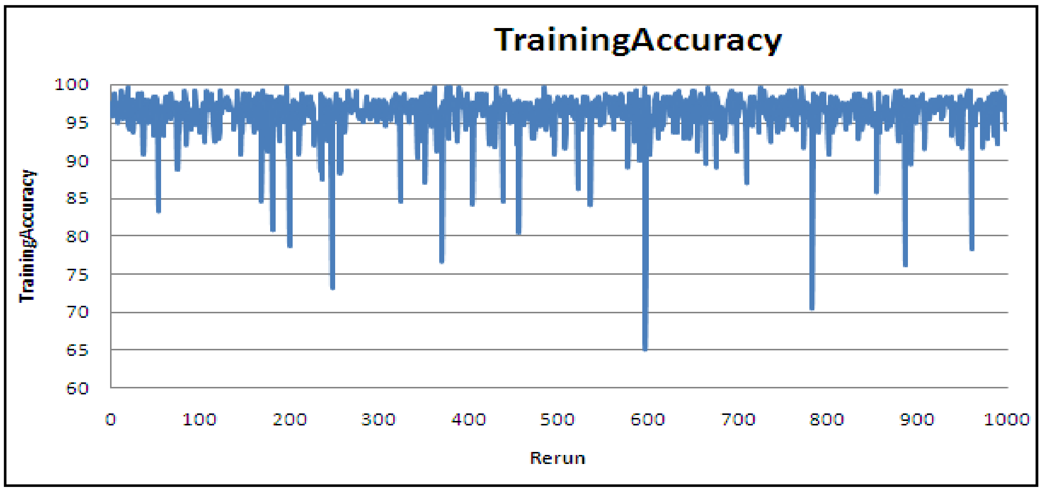


Figure 4.4: Training accuracy vs Rerun

The graph above showed that the training accuracy was distributed equally through whole the rerun times. Since it was distributed equally, it means that the rerun value can be cut down to a sufficient condition. Hence, a comparison between the rerun parameter was carried out to determine the rerun value. Table 4.1 shows the comparison on rerun parameters. From the table, training accuracy and training mean square error were shown not much different. In advance, the average deviations of these results were nearly the same. Consequently, rerun value was set to 100 since it could use to represent the 1000 times rerun condition where prove the results are not bias.

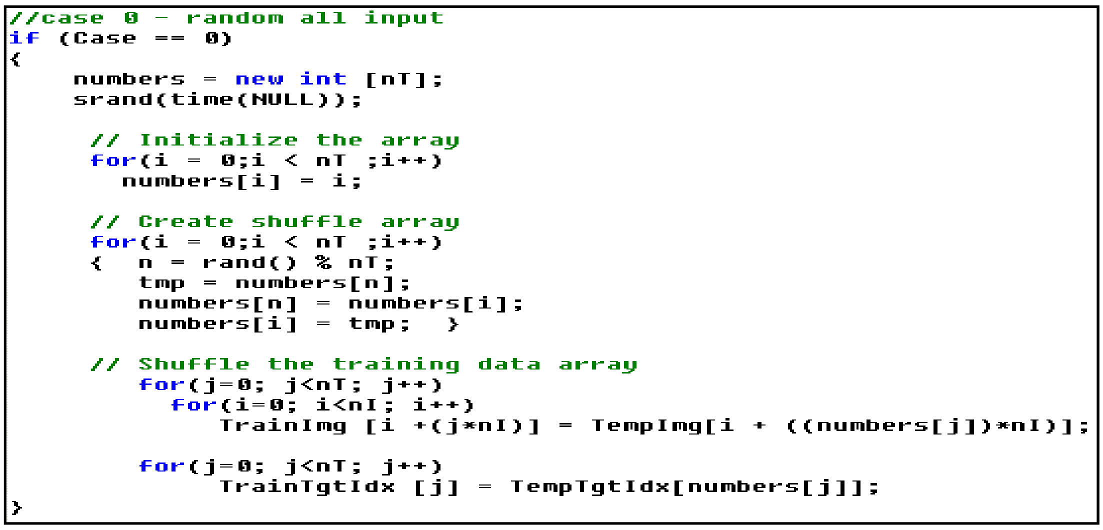
Table 4.1: Comparison on rerun parameters

Rerun	Training Accuracy	TrainingMSE	Average deviation
1000	96.18	0.02	1.79
500	96.12	0.02	1.81
100	96.37	0.02	1.51

4.1.2 Creation of ordering patterns

After setting the configuration of network program, ordering study was carried out. In this ordering study, the backpropagation algorithm was used, so momentum constant and amplifying factor of adaptation learning rate were set to zero. Backpropagation algorithm was used in this study rather than proposed algorithm, because it is the universal algorithm used to perform the research. Hence, the study results were identified.

The seven cases of the training data ordering were formed by using C++ program code. From case 0 to case 4, the training data was arranged from left to centre and then to right before loading into the program, while case 5 and case 6 were arranged the training data in rotation order, which is upright to skewed images before load into the program. All data input file were arranged in sequence order, which patterns follow digit by digit. Figure 4.5, 4.6 and 4.7 showed the codes use to generate random, sequence and interleaving sequence order respectively.

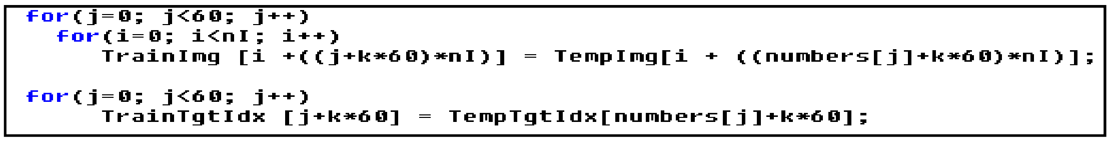


```
//case 0 - random all input
if (Case == 0)
{
    numbers = new int [nT];
    srand(time(NULL));

     // Initialize the array
     for(i = 0;i < nT ;i++)
       numbers[i] = i;

     // Create shuffle array
     for(i = 0;i < nT ;i++)
     {   n = rand() % nT;
         tmp = numbers[n];
         numbers[n] = numbers[i];
         numbers[i] = tmp;   }

     // Shuffle the training data array
         for(j=0; j<nT; j++)
           for(i=0; i<nI; i++)
              TrainImg [i +(j*nI)] = TempImg[i + ((numbers[j])*nI)];

         for(j=0; j<nT; j++)
              TrainTgtIdx [j] = TempTgtIdx[numbers[j]];
}
```

Figure 4.5: Case 0 Random Order

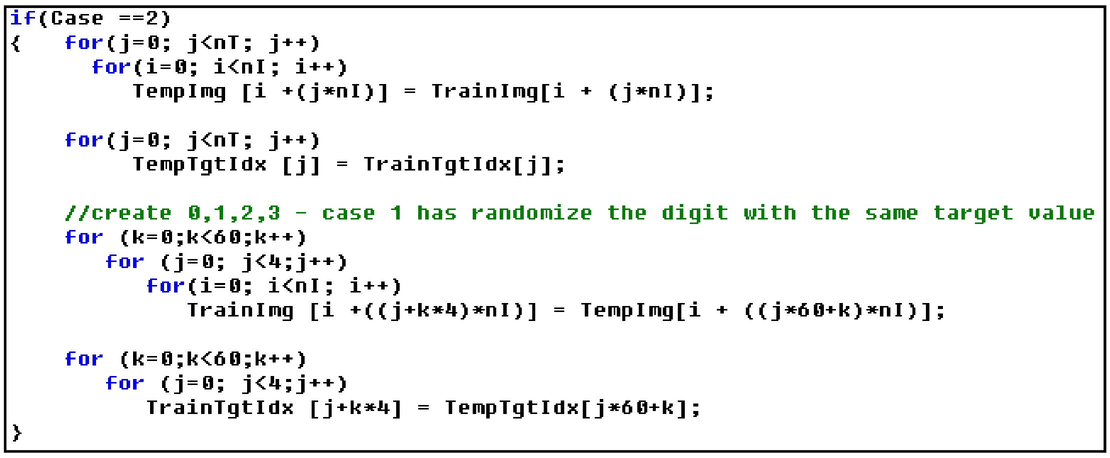
Code in figure 4.5 will generate a random order for the training data. Initially, it created a shuffle temporary array by formula rand()%nT. The total number of training data, nT was used because whole training data set should jumble up in this case 0. A series of random shuffle number from 0 to 239 were then generated. Subsequently, input data ordering were modified by this shuffle array to form its random ordering patterns.

Coding of case 1 which is sequence order was generated by modifying the formula from rand()%nT to rand()% (nT/4). In case 1, data was trained in the sequence from digit 0 to 3, so the data were only randomize between the same digit patterns. Hence, formula was modified to rand()% (nT/4). After shuffled the temporary array, input data ordering were updated by the coding shown in figure 4.6. Next, Case 2 which is interleaving sequence order was created based on coding of case 1. Extension of code was used to create the data interleaving patterns which are 0, 1, 2, 3, 0, 1, 2, 3, until the end of the training sequence. Figure 4.7 shows the extension code of case 2 to create the interleaving patterns.



```
 for(j=0; j<60; j++)
   for(i=0; i<nI; i++)
      TrainImg [i +((j+k*60)*nI)] = TempImg[i + ((numbers[j]+k*60)*nI)];

 for(j=0; j<60; j++)
      TrainTgtIdx [j+k*60] = TempTgtIdx[numbers[j]+k*60];
```

Figure 4.6: Case 1 Sequence Order



```
if(Case ==2)
{    for(j=0; j<nT; j++)
       for(i=0; i<nI; i++)
          TempImg [i +(j*nI)] = TrainImg[i + (j*nI)];

     for(j=0; j<nT; j++)
          TempTgtIdx [j] = TrainTgtIdx[j];

     //create 0,1,2,3 - case 1 has randomize the digit with the same target value
     for (k=0;k<60;k++)
        for (j=0; j<4;j++)
           for(i=0; i<nI; i++)
              TrainImg [i +((j+k*4)*nI)] = TempImg[i + ((j*60+k)*nI)];

     for (k=0;k<60;k++)
        for (j=0; j<4;j++)
           TrainTgtIdx [j+k*4] = TempTgtIdx[j*60+k];
}
```

Figure 4.7: Case 2 Interleaving sequence Order

Results of the ordering distribution of the random, sequence and interleaving sequence order were shown in figure 4.8 below. They were the ordering patterns generated by the coding above.

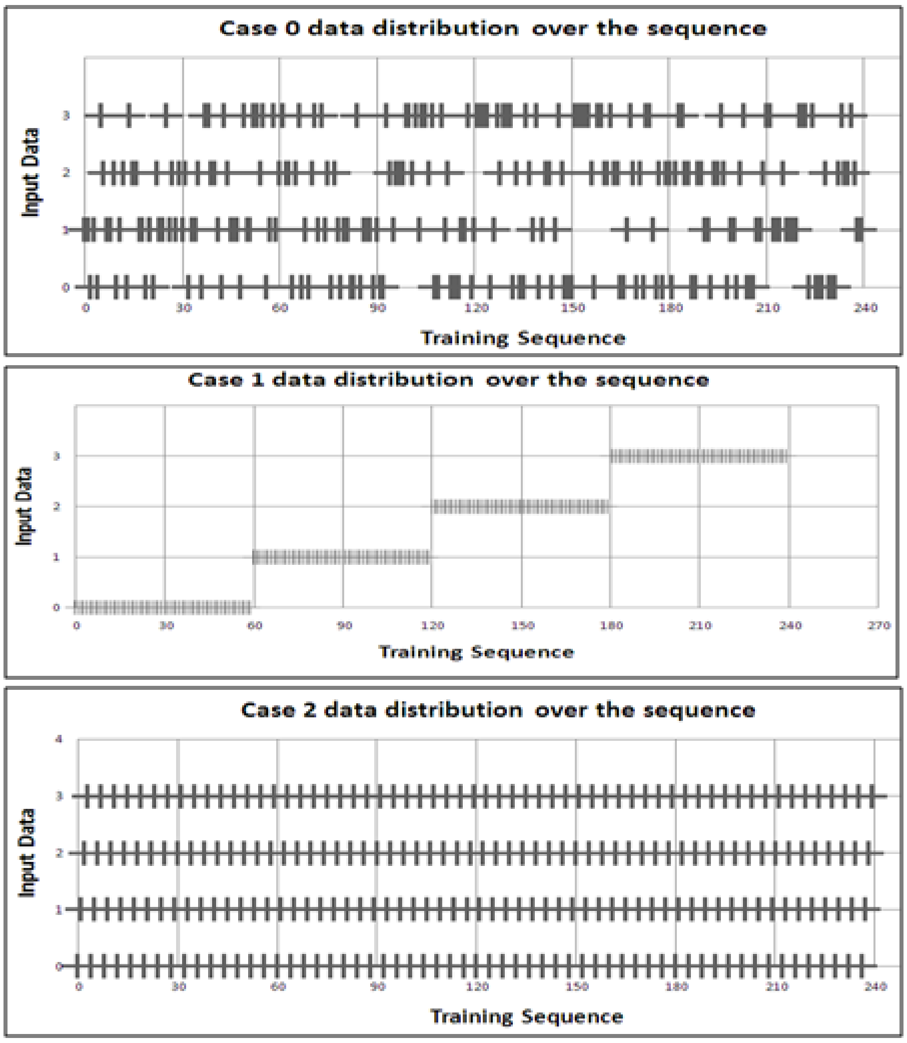


Figure 4.8: Results of ordering distribution for case 0, 1 and 2

Case 3 and case 4 considered the transition order for the training inputs. In these two cases, the training data was arranged from left to centre and then to right before loading into the program. The coding used to create sequence and interleaving training patterns only. Furthermore, case 5 and case 6 were using the same coding with case 3 and case 4 respectively except the training data was arranged in rotation order, which is upright to skewed images before load into the program. Figure 4.9 below shows the ordering results for case 3, 4, 5 and 6 which created by the program code.

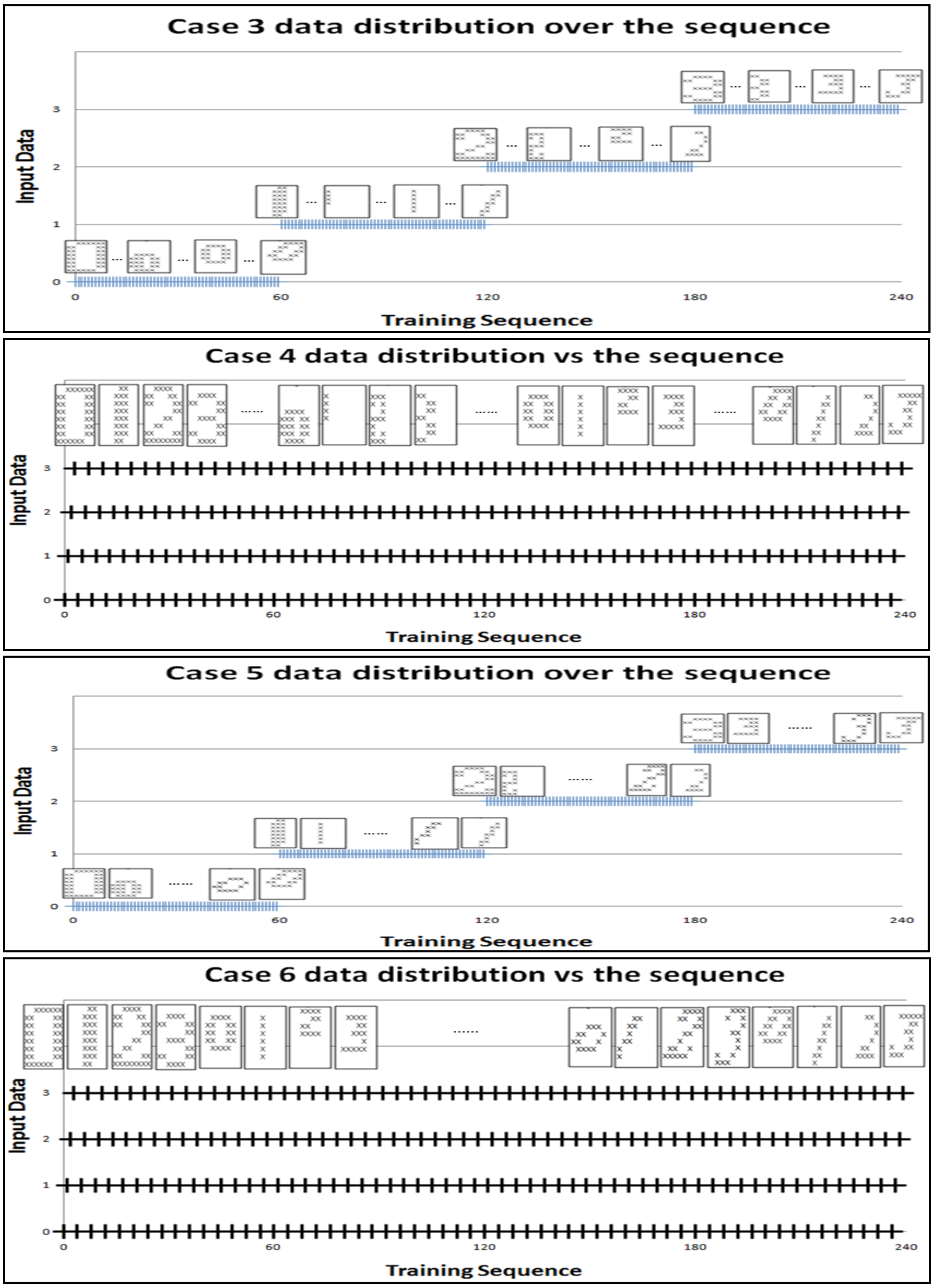


Figure 4.9: Results of ordering distribution for case 3, 4, 5 and 6

4.2 Results and discussion of ordering study for training data set

4.2.1 Results

Results for all the 7 ordering cases are shown in table 4.2. These results showed the average value of 5 execution times for each case. Total average is the percentage of corrected output numbers divided by total training sample. Whilst total perfect is the percentage score in the condition where corrected outputs equal to total training sample throughout whole training process. The minimum percentage score for average results is 50% due to the random prediction is either true or false, while perfect results has 0% for its minimum score. Generalization accuracy and validation accuracy are the average results for generalization and validation data set.

Table 4.2: Results for 7 ordering cases

Case	Condition	Total Average	Total Perfect	Total Generalization Accuracy	Total Validation Accuracy
0	Random order	99.32%	16.80%	61.13%	68.87%
1	Sequence order	99.60%	33.80%	60.51%	67.60%
2	Interleaving sequence order	99.24%	14.00%	60.56%	69.42%
3	Sequence + transition order	99.54%	46.00%	60.28%	66.55%
4	Interleaving sequence + transition order	99.28%	11.60%	61.67%	70.95%
5	Sequence + rotation order	99.44%	33.60%	61.15%	67.43%
6	Interleaving sequence + rotation order	99.34%	12.80%	61.73%	69.78%

From table 4.2, it illustrated that ordering patterns of training data affecting the results of total perfect drastically. Besides, the results of total average were slightly affected by the ordering patterns. It may due to the reference point of total average which 99.32 was almost reach its maximum value, hence the influence of ordering patterns were not demonstrated significantly. However, total generalization accuracy and total validation accuracy were not influenced by ordering patterns. The distribution of the perfect score in all the 5 execute time were plotted and showed in figure 4.10.

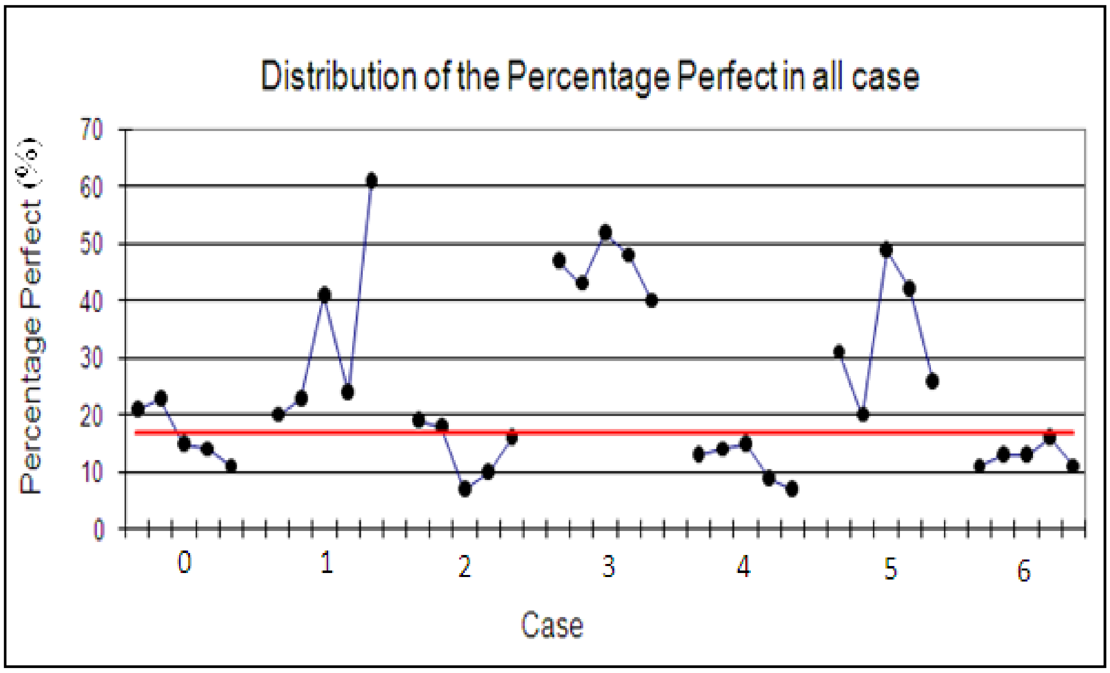


Figure 4.10: Distribution of the Percentage Perfect

The red line in the graph above represents the reference point. It was determined by the average of 5 total perfect score in case 0 or random order. From the graph above, case 1, 3 and 5 perform higher output compare to other. Moreover, case 1 and case 5 had higher variability than case 3. Therefore, case 3 was the best ordering to improve the network performance, because low variability can make the results more predictable. In contrast, case 2, 4 and 6 were performed lower output compare to the reference point.

After observing the overall results of the ordering cases, more detail investigations on these cases were carried out. It was beginning by comparing random, sequence and interleaving sequence order. Transition and rotation order were not considered yet since they cannot stand alone to test its own effects to the network. Table and graph below shows the comparison on the 3 different cases. In this comparison, 20 testing were taken for each case and average result was counted.

Table 4.3: Comparison on case 0, 1 and 2

Case	Condition	Total Average	Total Perfect
0	Random order	99.34%	18.10%
1	Sequence order	99.62%	40.25%
2	Interleaving sequence order	99.20%	12.85%

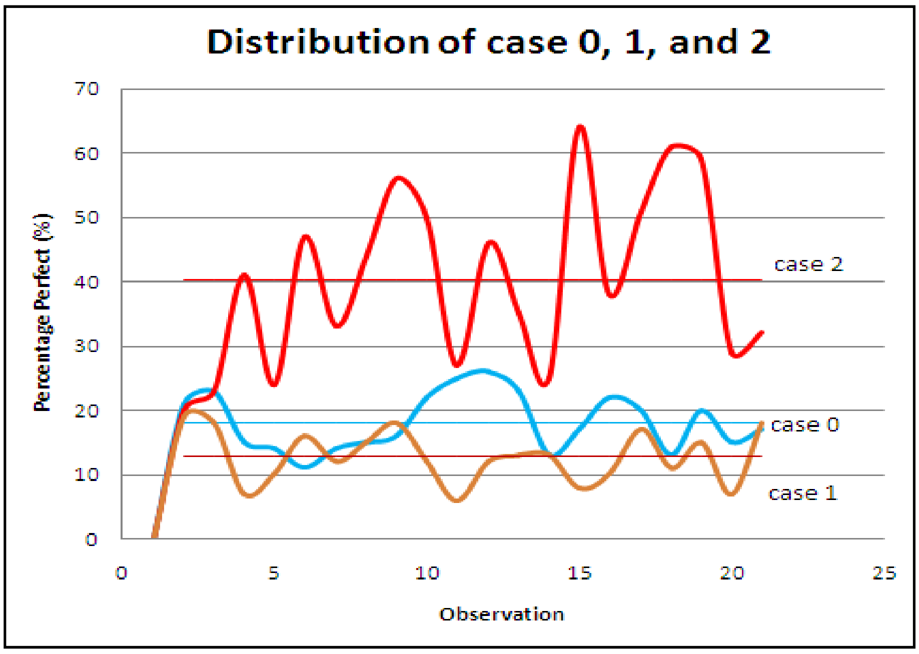


Figure 4.11: Distribution of case 0, 1, and 2

Random order was used as the reference point to calculate the percentage improvement of other cases. Equation 4.1 and 4.2 show the percentage improvement of case 1 while equation 4.3 and 4.4 represent case 2. The formula used to calculate the percentage improvement on total average and total perfect are different. Since the total average results were almost reached its maximum value of 100, then ratio improvement in that small portion was calculated. Nevertheless, percentage improvement for total perfect was counted based on the reference value. It shows how much improvement was achieved by the new results compare to the reference value.

$$\%\ improvement_{total\ average\ case\ 1} = \frac{99.62-99.34}{100-99.34} \times 100\% = 42.42\% \tag{4.1}$$

$$\%\ improvement_{total\ perfect\ case\ 1} = \frac{40.25-18{,}1}{18{,}1} \times 100\% = 122.38\% \tag{4.2}$$

$$\%\ improvement_{total\ average\ case\ 2} = \frac{99.20-99.34}{100-99.34} \times 100\% = -21.21\% \tag{4.3}$$

$$\%\ improvement_{total\ perfect\ case\ 2} = \frac{12.85-18.1}{18{,}1} \times 100\% = -29.01\% \tag{4.4}$$

From the calculation above, ordering of case 1 was proved performing better than the random order, while ordering of case 2 was degraded the performance of the network. Sequence order or case 1 enhanced the total perfect performance at least twice and at the same time rinse the total average output from 99.34 to 99.62.

Next, transition order investigation was considered. The comparison between both transition orders with different combination order was made based on the table 4.2. Random order was used as reference point to calculate the percentage improvement. This comparison was shown in table below. Percentage improvement of these two cases was calculated as well. Equation 4.5 and 4.6 represent improvement of case 3 while equation 4.7 and 4.8 represent the improvement of case 4.

Table 4.4: Comparison on case 3 and 4

Case	Condition	Total Average	Total Perfect
0	Random order	99.32%	16.80%
3	Sequence + transition order	99.54%	46.00%
4	Interleaving sequence + transition order	99.28%	11.60%

$$\%\ improvement_{total\ average\ case\ 3} = \frac{99.54-99.32}{100-99.32} \times 100\% = 32.35\% \qquad (4.5)$$

$$\%\ improvement_{total\ perfect\ case\ 3} = \frac{46.0-16.8}{16.8} \times 100\% = 173.81\% \qquad (4.6)$$

$$\%\ improvement_{total\ average\ case\ 4} = \frac{99.28-99.32}{100-99.32} \times 100\% = -5.88\% \qquad (4.7)$$

$$\%\ improvement_{total\ perfect\ case\ 4} = \frac{11.6-16.8}{16.8} \times 100\% = -30.95\% \qquad (4.8)$$

In table 4.4, transition order was showed improvement when combining with sequence order while it performs poorly when combining with interleaving sequence order. These results can be proved by the calculation made in equation 4.5 to 4.8. In conclusion, considering transition ordering patterns can improve the total perfect 2.75 times compare to the reference value.

After that, rotation order was examined as well. The comparison between both rotation orders with different combination order was made. This comparison was shown in table below. Percentage improvement of these two cases was calculated as case 3 and 4. Equation 4.9 and 4.10 represent case 5 calculations while equation 4.11 and 4.12 represent case 6 calculations.

Table 4.5: Comparison on case 5 and 6

Case	Condition	Total Average	Total Perfect
0	Random order	99.32%	16.80%
5	Sequence + rotation order	99.44%	33.60%
6	Interleaving sequence + rotation order	99.34%	12.80%

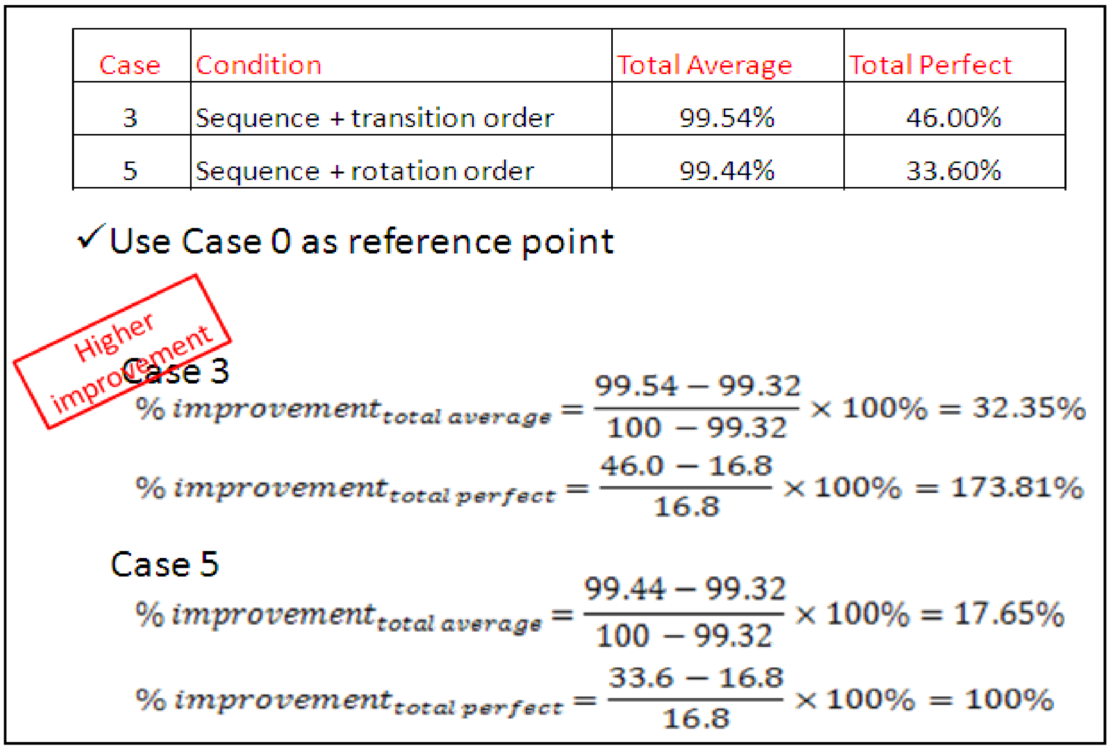
$$\%\ improvement_{total\ average\ case\ 5} = \frac{99.44-99.32}{100-99.32} \times 100\% = 17.65\% \qquad (4.9)$$

$$\%\ improvement_{total\ perfect\ case\ 5} = \frac{33.6-16.8}{16.8} \times 100\% = 100\% \tag{4.10}$$

$$\%\ improvement_{total\ average\ case\ 6} = \frac{99.34-99.32}{100-99.32} \times 100\% = 2.94\% \tag{4.11}$$

$$\%\ improvement_{total\ perfect\ case\ 6} = \frac{12.8-16.8}{16.8} \times 100\% = -23.81\% \tag{4.12}$$

Similar to transition cases, rotation order showed improvement when combining with sequence order while perform weakly when combining with interleaving sequence order. However, these rotation order effects were not so significant compared to transition cases. It can be proved by comparing the transition order and rotation order with the same combination order which is sequence order. Figure below showed that comparison between transition order and rotation order. Both cases showed enhancement to network performance, but transition order will much helpful in improving the performance compare to rotation order.



Case	Condition	Total Average	Total Perfect
3	Sequence + transition order	99.54%	46.00%
5	Sequence + rotation order	99.44%	33.60%

✓Use Case 0 as reference point

Higher improvement

Case 3

$$\%\ improvement_{total\ average} = \frac{99.54-99.32}{100-99.32} \times 100\% = 32.35\%$$

$$\%\ improvement_{total\ perfect} = \frac{46.0-16.8}{16.8} \times 100\% = 173.81\%$$

Case 5

$$\%\ improvement_{total\ average} = \frac{99.44-99.32}{100-99.32} \times 100\% = 17.65\%$$

$$\%\ improvement_{total\ perfect} = \frac{33.6-16.8}{16.8} \times 100\% = 100\%$$

Figure 4.12: Comparison between transition and rotation order

After calculated the percentage improvement for every cases, the ordering were divided into degraded and improved cases. The following graphs show in figure 4.13 and 4.14 were illustrated the degraded cases and improved cases respectively.

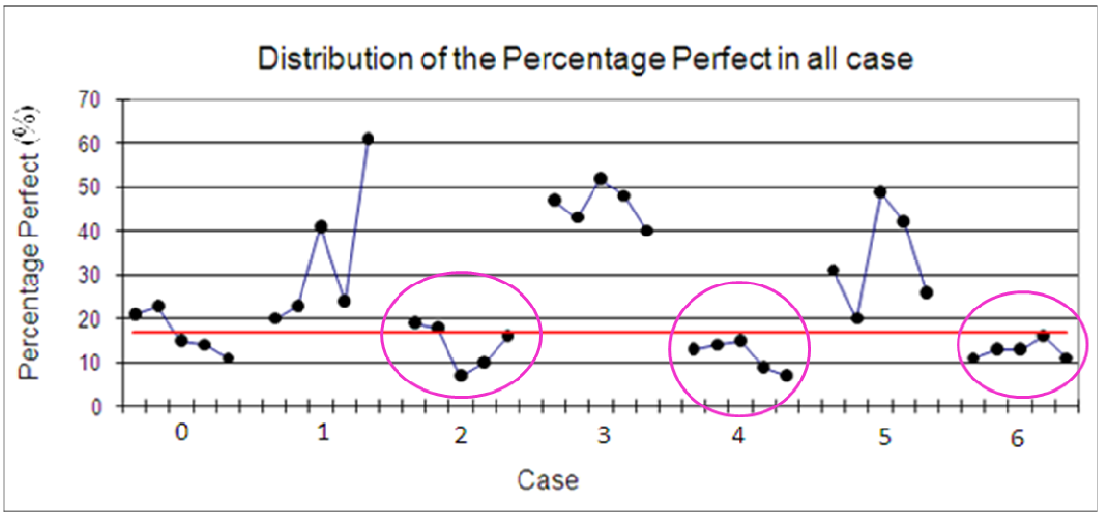


Figure 4.13: Degraded cases

From figure 4.13, all the degraded cases had a common ordering pattern which is interleaving sequence ordering. Therefore, this ordering pattern can be concluded as a poor ordering pattern to a supervised learning network.

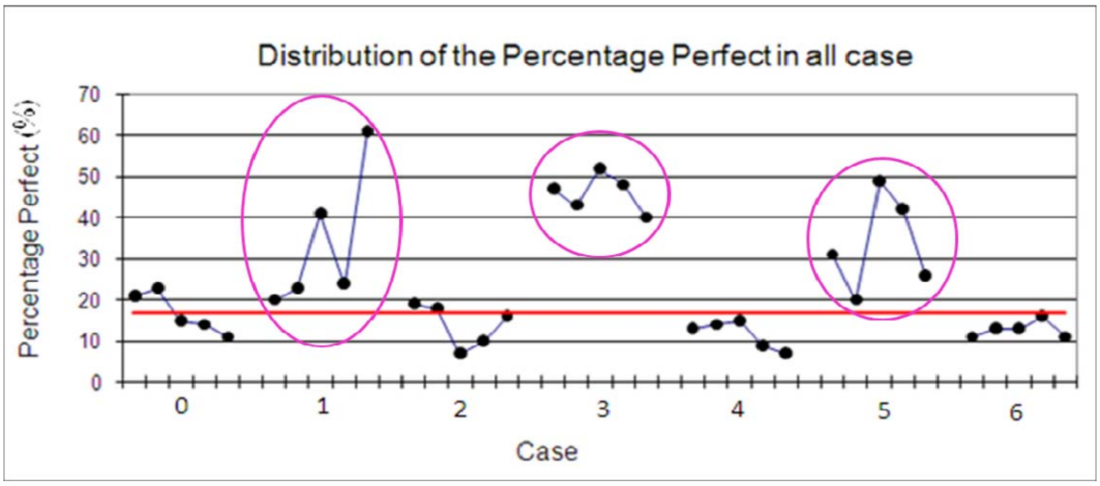


Figure 4.14: Improved cases

In contrast, there are three improved cases were determined in figure 4.14, which are sequence order, sequence plus transition order and sequence plus rotation order. The higher average performance was sequence plus transition order or case 3. However, these 3 cases were helping in enhance the network performance. Hence, a new additional ordering which is sequence plus transition plus rotation order was considered. The total perfect of new addition ordering was plotted in figure 4.15.

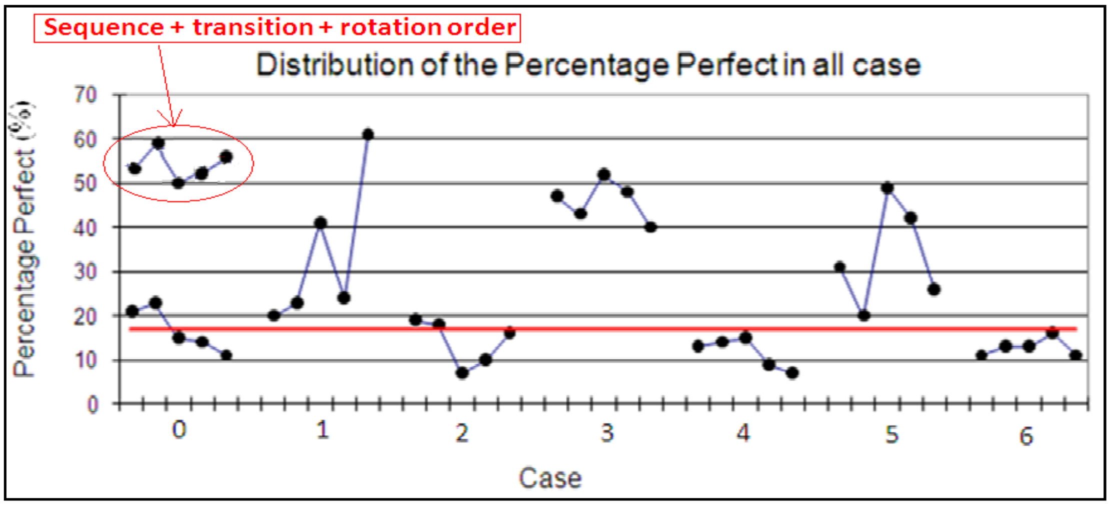


Figure 4.15: Comparison of new additional ordering to random order

From figure 4.14, the new additional ordering performs 3 times better than random order. It was also performed the highest perfect output compare to other. The average of 5 total perfect percentages was 56% and its total average was 99.7%. Percentage improvement was counted for both total average and total perfect which show in equations below.

$$\%\ improvement_{total\ average} = \frac{99.7-99.32}{100-99.32} \times 100\% = 55.88\% \qquad (4.13)$$

$$\%\ improvement_{total\ perfect} = \frac{56.0-16.8}{16.8} \times 100\% = 233.33\% \qquad (4.14)$$

Formula 4.13 and 4.14 show the highest improvement was achieved by this new additional case. It helps to enhance the network performance at least 3 times from the reference condition. Hence, optimal ordering patterns was chosen to be sequence plus transition plus rotation order.

4.2.2 Discussion

From the analysis of ordering of the training data, overall 7 cases' performances were plotted and observed. From these cases, results on their percentage score for total perfect was significantly influence by the ordering patterns. These total perfect score increment will lead to the improvement of total average score as well. However, it was not so significant in the total average score due to the score almost reaches its maximum value and increments are not shown significantly. When the total average score was almost reached 100, then the increment in total perfect score can use to represent the improvement of the network performance. Besides, the total generalization and total validation score from table 4.2 was not shown any influence of the ordering patterns. It may cause by some samples in the generalization and validation data set was too confusing, so even output was improved but the confusing images were still not recognized correctly.

Next, the distribution of cases which shows as figure 4.10 can use to determine the variability of the cases. This variability represents the predictability of the neural network. If the variability is small, it means the output results are more predictable. Predictability is important in neural network to show the consistent performance of the training results. When a best ordering patterns was chosen, not only the score of total average and total perfect are observing, the variability of the results should include into the consideration as well. If a case has a highest score of total perfect increment but variability of that score are large, then the worst case of that particular case should find out to confirm it is greater than the reference point before determine that particular case as the best ordering patterns.

Comparisons between small portions of 7 training ordering were made. First, table 4.3 and figure 4.11 show the comparison between case 0, 1, and 2. In this comparison, every case was performing 20 times to observe the variability of the ordering patterns. Results shows that sequence order performed better than random order and interleaving sequence order. To ensure the sequence order is the best among them, distribution of the cases was observed. In figure 4.11, even though sequence order had large variability compare to other two cases, it perform well over all the testing. The lowest performance of sequence order was still better than other. Hence, it can conclude that it was the best ordering among them. Besides, interleaving sequence order was perform poorly compare to random order. Their best result was just reached the average value of random order.

From results above, sequence ordering was showed to help in network performance most. Sequence order was learning all the different patterns of one digit before proceeding to another digit. Hence, program can learn a particular digit until expert in recognizing the particular digit in all different aspects before it proceeds to other. Even though the program proceeds to learn another new digit patterns, program will still have some memory to the digit it learn before. This can be explained in the way human being learning a new thing.

For example, when lecturer teaches a new equation or concept to his/her students, the lecturer will try to explain it and show the different application of that particular equation or concept. Hence, student will learn the equation deeply through all different points of view. After student familiar with the equation or concept, only then they will move to another new concept. Even though proceed to learn another new thing, the memory to recognize the learned knowledge was still there. However, if the equation or concept was touched and gone, then student will just ignored the knowledge and fail to recognize it in the future. Hence, sequence ordering help in learning process and interleaving sequence performed the worst results. Random order was perform better than interleaving sequence because it may generate a short block of sequence order to improve its classification ability rather than all different output targets.

Afterwards, transition and rotation order were found to be helpful in network performance when combining with sequence order. This phenomenon was also can be explained by human being learning process. For example, if a child starts learning the digit, the similar digit form are learned first before learning the confusing image. The variability between each sample should not too big, so that the child can classify the new sample easily based on his/her past knowledge. Therefore, transition and rotation order which produce the data in small variability changes was advised to apply. From figure 4.12, results showed that transition order was made a greater improvement in the network performance compared than rotation order. It was due to the variability change in transition order smaller than rotation order.

Adding all the issues help in learning process was produced the best results for the neural network. It was proved in figure 4.15 where sequence plus transition plus rotation order showed the best result in the study. In conclusion, learning enough samples for a particular digit changes with small variability between each learning sample may helps to improve the classification ability of the neural network.

4.3 Results and discussion of optimization for training algorithm

4.3.1 Results

After finding the optimal ordering patterns for training data, optimization on the training algorithm’s parameters was carried out. In this study, the optimal ordering patterns with total average equal to 99.7% and total perfect equal to 59% was used. To increase the convergence speed, parameters such as momentum constant, initial learning rate, and amplifying factor were optimized. These three parameters were affected each other. To simplify the analysis, optimal value of momentum constant and initial learning rate were determined first. Then, the determined value of initial learning rate was used to discover the optimal amplifying factor of adaptation learning rate.

The initial learning rate and momentum constant must be adjusted until the best network performance was obtained. Hence, the table below shows the different parameters testing for initial learning rate and momentum constant.

Table 4.6: Learning rate and momentum parameter determination

learning rate	momentum	average	perfect
0.2	0.05	99.50	36.00
0.2	0.1	99.40	36.00
0.2	0.5	98.00	4.00
0.1	0.1	99.70	55.00
0.1	0.05	99.70	59.00
0.05	0.1	99.80	69.00
0.05	0.2	99.70	58.00

In table 4.6, percentages of total average and total perfect were collected. The higher these two values, the better the network performs. In this part of study, learning rate equal to 0.05 and momentum constant equal to 0.1 was produced the highest output performance. Its total average was reached 99.8% and total perfect was gone to 69% at that state. This output value was improved compare to the optimal ordering output which set as the reference point in this part of study. It showed that the convergence speed of training algorithm was improved, so that the neural network can learn more in the same epoch value. Hence, these parameter values were chosen as the optimal value for initial learning rate and momentum constant. Furthermore, to determine the improved convergence speed, epoch value was being reduced until the output value was nearly equal to the reference point. In that moment, processing time of the algorithm was recorded down to compare with the reference case. Result recorded was shown in table 4.7 later.

Next, amplifying factor or constant parameter H in adaptation learning rate was considered. Using the initial learning rate which set from the previous analysis which is 0.05, optimal value of parameter H was found. Figure below show the graph of percentage perfect versus parameter H value.

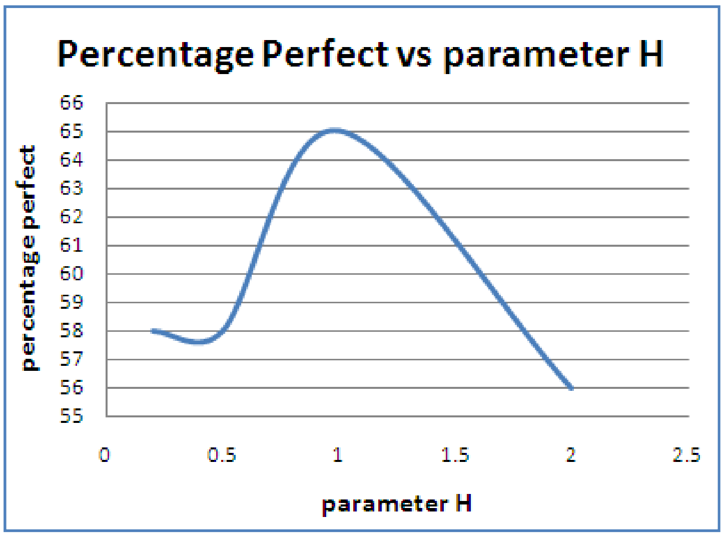


Figure 4.16: Percentage Perfect vs parameter H

Similar to previous case, the highest output performance showed the network convergence speed was increased. The amplifying factor, H was optimizing at value 1.00. In that value, percentage perfect was produced a higher output compare to the reference point. Hence, same procedure as previous state was carried out. Epoch value was reduced to achieve the similar output results with the reference point, and then processing time was recorded down. It shows in table 4.7 below.

Table 4.7: Comparing the algorithm's processing time

Method	time (minutes)
backpropagation	9.022
momentum	5.587
momentum + adaptive learning rate	4.095

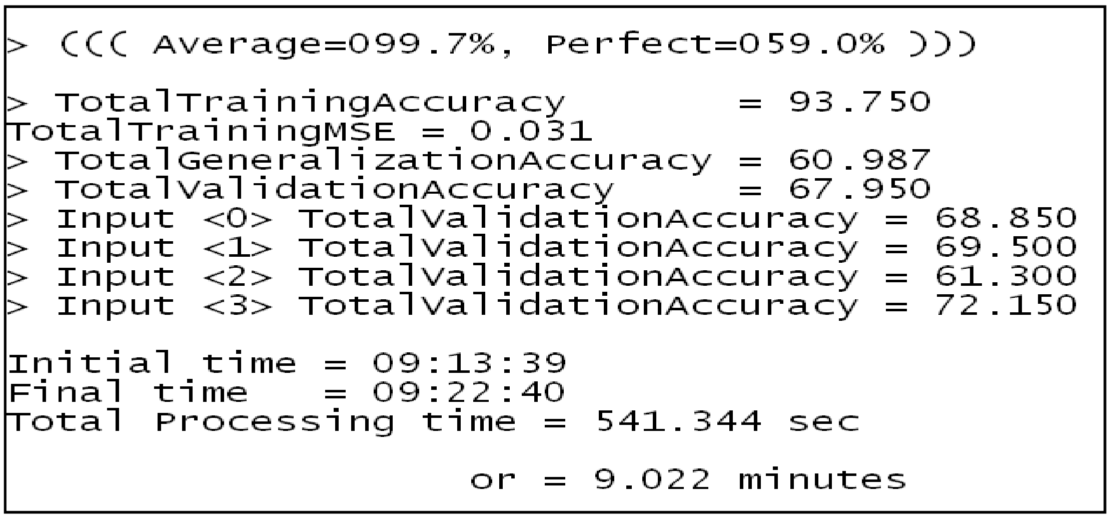
Proposed algorithm which is momentum plus adaptive learning rate had the most shorten training time. It was proved that it improve the convergence speed of neural network. Backpropagation algorithm was used as the reference point and percentage improvement for momentum and the proposed algorithm were counted as below.

$$\% \, improvement_{momentum} = \frac{9.022-5.587}{5.587} \times 100\% = 61.48\% \quad (4.15)$$

$$\% \, improvement_{proposed\ algorithm} = \frac{9.022-4.095}{4.095} \times 100\% = 120.32\% \quad (4.16)$$

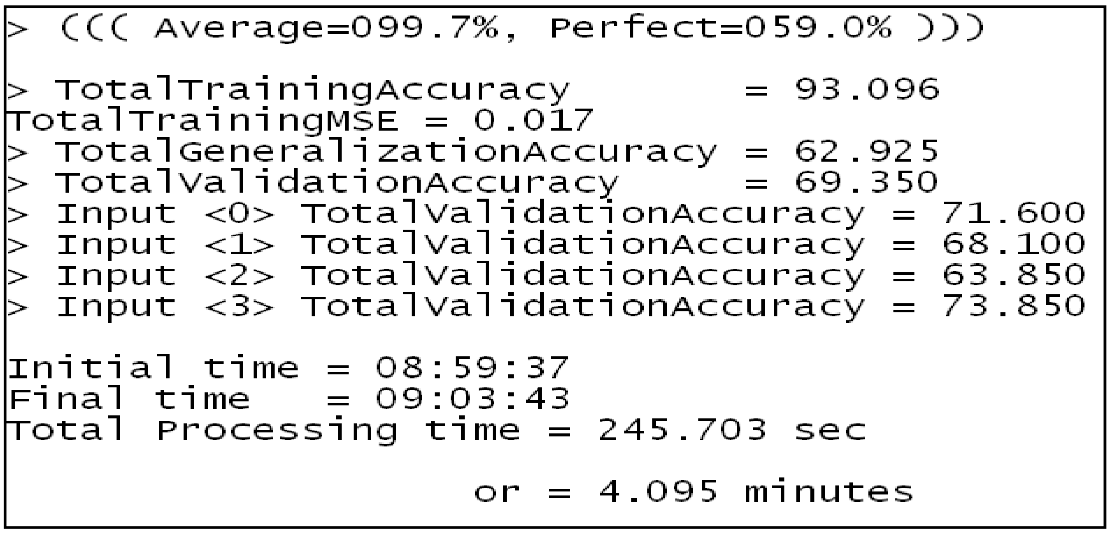
Equation 4.15 and 4.16 showed that proposed algorithm had higher improvement on the convergence speed compare to momentum algorithm. It reduced half of the processing time required to train the network in backpropagation algorithm. Hence, proposed algorithm was shown to improve the convergence speed of neural network with its optimal parameters.

Finally, the output results before and after optimizing the algorithm parameters were shown below.



```
> ((( Average=099.7%, Perfect=059.0% )))

> TotalTrainingAccuracy        = 93.750
TotalTrainingMSE = 0.031
> TotalGeneralizationAccuracy = 60.987
> TotalValidationAccuracy     = 67.950
> Input <0> TotalValidationAccuracy = 68.850
> Input <1> TotalValidationAccuracy = 69.500
> Input <2> TotalValidationAccuracy = 61.300
> Input <3> TotalValidationAccuracy = 72.150

Initial time = 09:13:39
Final time   = 09:22:40
Total Processing time = 541.344 sec

                   or = 9.022 minutes
```

Figure 4.17: Before parameter optimization



```
> ((( Average=099.7%, Perfect=059.0% )))

> TotalTrainingAccuracy       = 93.096
TotalTrainingMSE = 0.017
> TotalGeneralizationAccuracy = 62.925
> TotalValidationAccuracy     = 69.350
> Input <0> TotalValidationAccuracy = 71.600
> Input <1> TotalValidationAccuracy = 68.100
> Input <2> TotalValidationAccuracy = 63.850
> Input <3> TotalValidationAccuracy = 73.850

Initial time = 08:59:37
Final time   = 09:03:43
Total Processing time = 245.703 sec

                   or = 4.095 minutes
```

Figure 4.18: After parameter optimization

4.3.2 Discussions

In the interpretation of the parameters of training algorithm, parameters of learning rate, momentum constant and amplifying factor H are affecting each other. It is because all these 3 parameters are used together in the weight-updating formula, so they will influence each other.

The ratio between initial learning rate and momentum constant will affect whole program functions. If initial learning rate was too much larger than momentum constant, the momentum effects will disappear. Program will not help to converge faster in the same direction. Hence, whole program remain slow convergence rate. On the other hand, if momentum constant was too much bigger than initial learning rate, program required very long time when it needs to change direction towards its target. Worst case may be happened where program pass by the target and find local minimum target rather than global minimum target.

To evade both conditions above, a balance value had to achieve between these initial learning rate and momentum constant. However, value of momentum constant should larger than initial learning rate, so that the momentum effects will continue functioning. From the results produced, momentum constant was doubled the initial learning rate to produce highest convergence speed.

Next, adaptation learning rate was only functioning in certain amplifying factor, H value. If amplifying factor too large, network output will be oscillated and not stable. It is because when amplifying factor H has a high value, the exponential function will enlarge it again and make whole program oversensitive to every new data, and then lead to system failure. The proper range of amplifying factor was found to be 0 to 2 which shown in graph 4.16. The graph in figure 4.16 was proved this proper range was helped in network performance also. In this range, amplifying factor showed its highest convergence speed on value 1.00.

The proposed algorithm increases the network convergence speed at least twice to shorten the training time. Adding adaptation learning rate rather than using momentum method only was proved to have highest convergence speed than only used momentum method.

Overall, the proposed algorithm was proved to enhance the convergence speed of the network.

CHAPTER 5

CONCLUSION

5.1 Overall summary

In this project, an optimal ordering of the data patterns was developed to improve the performance of the neural network. Besides, a combined algorithm from existing algorithm to train the network was carried on to improve the convergence speed or training time in a network.

From the results of the data patterns ordering analysis, a best way to improve the network performance are the sequence order plus transition and rotation order. It helps to improve at least twice the output performance of the neural network compare to the random order. It showed that learning all the similar target output patterns to the expert condition before proceed to other may helps in network performance. The optimal ordering shows that the neural network learning process was work in the same manner of human being learning process. Hence, the improvement in the results proved that the ordering patterns of the training data will affect the generalization ability of the neural network.

Next, the optimization of the training algorithm has made in this project. The optimization of momentum constant, initial learning rate and amplifying factor H will help to shorten the training time. Their optimal values are 0.1, 0.05 and 1.0 respectively. Results showed that using a training program with combination of momentum and adaptive learning rate will require the 50% of the training time compare to backpropagation.

5.2 Future direction

The neural network is an interdisciplinary field, both in its development and in its application. Nowadays, the neural network was found to be applied in many fields. However, some problems are encountered by the neural network such as global minimum problem and convergence speed still cannot be solved.

Apart from the algorithm analyze in this research, there are several advanced concept to improve the training speed in the neural network, especially deal with the massive input data sets. Data partitioning methods can be used to shorten the training time when dealing with big data sets or data dimensions. In nature, the neural network processes all the patterns in the training over and over. If the data set is large, it is very time-consuming to run through all the data in the program. Data partitioning may help to partition the training set to provide shorten training time. Here are 2 types of data partitioning which can apply into the research to advance the training process time of the network. First approaches in data partitioning is a growing data set in the training process. Initially, a small subset of data is trained, and this subset is growing with a fixed percentage in each time training has completed. This growing data set will carry on until the whole training set is included in the learning process. This growing data set helps to reduce the training time in the beginning. However, it is still cover all the training set at the end to ensure the training process is done completely. The second approach is windowed data set which has a fixed size of the subset that move across the data set. It required a caution step to set the window size and the step size of the subset to ensure the performance of the network is not affected. Both approaches may help in shorten the training process especially in the massive input data sets. Figure below showed both approaches.

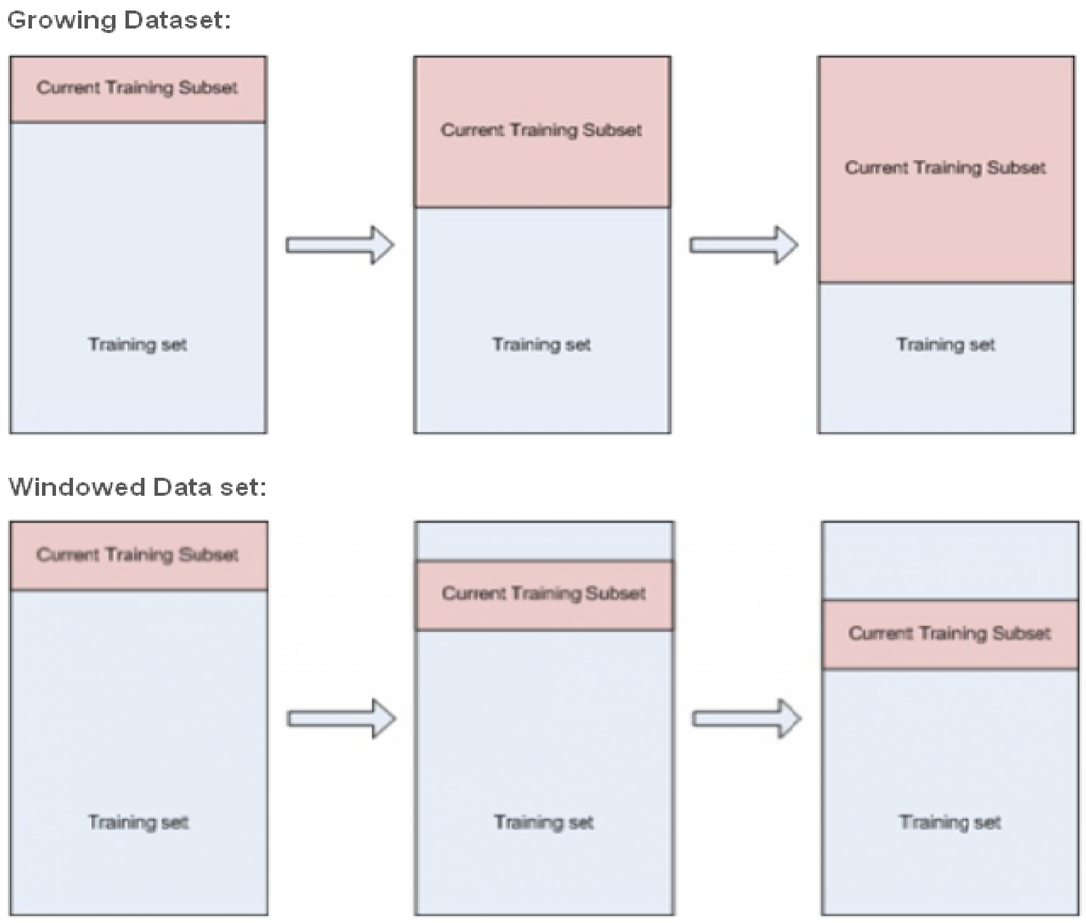


Figure 5.1: Growing and windowed datasets

Furthermore, the neural network performance can be improved by scaling and centering the input data. Even a small 8x8 dimensions used in this research faces this scaling and centering problem. The neural network receives the data in form of input stimuli. Thus, it cannot recognize the different sizes and alignment of a character without having to learn all possible samples. Scaling and centering the input data will helps to reduce the patterns of the learning process, it may shorten the learning time of the neural network. With this scaling and centering input data, the ordering of the training data patterns can reduce to the point where considering only the sequence order. It helps to simplify the ordering patterns as well. Figure below showed the scaling and centering method.

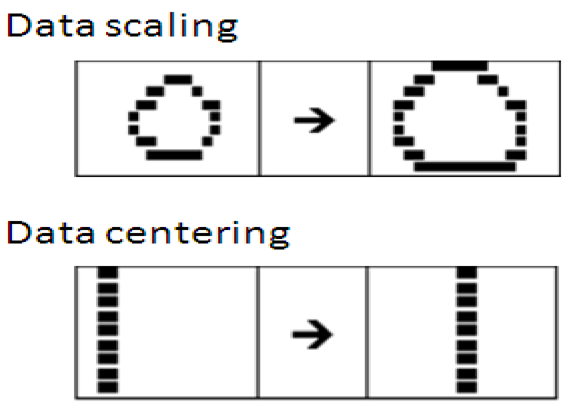


Figure 5.2: Data scaling and centering

REFERENCES

Chao Yang, & Ruzhi Xu. (2009, December 19-20). Adaptation Learning Rate Algorithm of Feed-Forward Neural Networks. *International Conference on*, vol., no., pp.1-3.

Gallant S.I. (1990, Jun). Perceptron-based learning algorithms. *IEEE Transactions on*, vol.1, no.2, pp.179-191.

Gou-Jen Wang, & Chih-Cheng Chen. (1996, May). A fast multilayer neural-network training algorithm based on the layer-by-layer optimizing procedures. *IEEE Transactions on* , vol.7, no.3, pp.768-775.

K. Hornik, M. Stinchcombe, & H. White. (1988). Multilayer feedforward networks are universal approximators. *Neural Networks*, vol., no., pp. 359-366

Laurene Fausett. (1994). *Fundamentals of Neural Network- Architectures, algorithms, and applications.* U.S. : Prentice Hall

Nguyen, D., & Widrow, B. (1990, June 17-21). Improving the learning speed of 2 layer neural networks by choosing initial values of the adaptive weights. *IJCNN International Joint Conference on* , vol., no., pp.21 26 vol.3.

Yamada K., Kami H., Tsukumo J., & Temma T. (1989, Jun 18-22). Handwritten numeral recognition by multilayered neural network with improved learning algorithm." *International Joint Conference on* , vol., no., pp.259-266 vol.2.

Yu X., Loh N.K., & Miller W.C. (1993, May 3-6). A new algorithm for training multilayer feedforward neural networks. *IEEE International Symposium on* , vol., no., pp.2403-2406 vol.4.

APPENDICES

APPENDIX A: C Programme Listing