University of Birmingham

School of Computer Science

Comparing Neural Networks for Hand Writing Recognition

Author:
Chan Man Yi 1391904
BSc Computer Science

Supervisor:
Hamid Dehghani

April 10, 2017

Abstract

This project aims to discover and evaluate the efficiency and accuracy of two kinds of neural networks in recognising hand written characters. The multi-layered perceptrons network and the radial basis fuction network algorithm were implemented in java alongside with an graphical user interface which served as a helping tool to perform testing.

A hand written English character recognition were designed to simulate the computation of the neural networks. Images of English hand written character were first processed into binary representation which serves as input neurons in the network, through constantly training the network to update the weights, the network would recognise a set of unseen testing data set. Different combination of parameters were used to train the network and looked at how they affect the results. Number of hidden neurons, learning rates and the number of iterations were experimented. During the process, the training and testing time were captured as well as the proportion of new data being recognised.

The experimental result showed the performance of the two neural networks differed varying different factors. In terms of of efficiency, both algorithms would yield a longer training time when the number of epochs increases, however, RBF network performed better in terms of training. In terms of accuracy, both algorithms were able to detect more than 60% of the testing set within a combination of parameters. But MLP performed better as the accuracy could go up to 93%.

Despite the experimental results, the issue of neural network generalisation had not been considered, therefore evaluation was made on how to improve the accuracy for the recognition. Finally, improvements and extendability were mentioned on how the project could be made better.

Contents

1 Introduction

As computerisation become more prominent, recognising unique handwriting of individuals is gaining importance in various fields, such as collecting forensic evidence, authentication of signatures in banks and recognising postcodes and addresses on letters. Within our human visual system, it is stupendously easy for our brain to interpret the visual world as our primary visual cortex are progressively performing complex image processing via billions of connected neurons.

Comparatively, the difficulty of visual pattern recognition become apparent when we are trying to write a computer program to recognise characters. Because of such issue, I am curious to explore further on this recognition problem. Use of artificial neural networks will be my approach to this problem. This approach takes a large sample of handwritten characters as a training set and then develops a system which learn from those training sets in order to correctly recognise new inputs.

2 Background

The purpose of this project is to understand the theory of different types of artificial neural networks and try to implement them in practice. Therefore the project is designed at investigating the effect of recognising English characters using this concept. Handwritten English characters are chosen as inputs, processed, and trained with neural network algorithms, afterwards, the network will try to recognise new unseen inputs to see how accurate and efficient the network can perform.

One of the primary means by which computers simulate how human brains work is through the use of neural network. Neural network has a remarkable ability to derive meaning from complicated or imprecise data. One of the advantages of neural network is that it can extract patterns and predict trends that are too complex to be noticed by human or computer techniques[1]. A correctly trained neural network is able precisely analyses category of information, such technique is known as classification which was mainly used in the course of the project.

Hand written recognition is a classification problem which is also known as logistic regression, the neural network will determine from a set of inputs (hand written characters) which class(alphabet) a given object belongs to, such decision is governed by an activation function. This function will predict the probability of a given example that belongs to the correct class versus the probability that it belongs to another class. A commonly used activation function is the sigmoid function, which is a non linear activation function:

$$f(x) = \frac{1}{1 + e^{-x}}$$

The way neural network operates involves three decisions. First, a network topology has to be chosen to decide how many layers of neurons will the network consists have, and how many neurons are there in each layer. Secondly,a suitable activation function has to be chosen. Last but not least, a learning process has to be determined for the neural network to learn the given data so that it can classify unseen data sets. In this project, two kinds of neural network learning process were implemented, they are the multi-layered perceptrons with back propagation network and the radial basis function neural network. The decision of choosing these two neural networks will be justified in the next section and the network design will be detail in section 4.

3 Project Specification

3.1 Neural Network

During the course of the project, experiments were carried out to compare the efficiency and accuracy of different neural networks while implementing a tool to test for hand written character recognition. Decision was justified on which neural networks would best suit the nature of the classification problem.

3.1.1 Available Options

1. Multi-layered Perceptron Network with back propagation(MLP network): A supervised machine learning task which copes with non-linearly separable problems via adding extra layers to the network as well as minimising a cost function. The situation of classifying 26 English characters from a set of data is non-binary, which meets the requirements for performing linear regression. The notion of back propagating weights to minimise a cost function provides a fast matrix based approach to compute the output from a neural network.

2. Radial Basis Function Network(RBF network): This network is derived from the theory of interpolation which approximates an underlying function from a set of noisy data in order to predict the output. Such method makes use of a Gaussian function to extract features for detection which is appropriate to learn the features of hand written character to recognise them.

3. Competitive learning neural network: An unsupervised technique in which output neuron with the highest activation is declared as winner and weights will be awarded to the winning neuron. Such method is able to train data sets into sub classes but has no information about the groupings which may not be favorable for further processing of the character recognition.

4. Binary Hopfield Network: This is also an unsupervised technique which recognises data by producing a set of trained binary values and find it's closest matching pattern. Such method is robust against hardware damage because of its ability to give a content-addressable memory, it also guarantees to converge to a local minimum but will sometimes converge to a false minimum. The network also has a draw-

back on not being able to interpret memory, which makes it harder to interpret characters for further usage.

3.1.2 Decision

Both competitive learning neural network and binary hopfield network are unsupervised learning techniques, although they are able to classified data using only the input data, they both lack the ability to interpret the classified data which will be used to compare the accuracy of recognising characters in later stage of the project. However a similar piece of software can also be implemented with these two networks. On the other hand, multi-layered perceptron network and radial basis function network undergo a supervised training strategy which training inputs and their expected targets are provided for learning. Such technique is achievable when interpreting hand written characters.

In addition, both networks possess a similar network topology but completely different learning approach, this allowed me to suggest hypotheses beforehand based on different experimental variables while maintaining a set of controlled variables. The hypotheses of comparing these two algorithms will be detailed in project analysis (Section 5). As a result, MLP network and RBF network were chosen as the algorithms which were implemented for recognising English characters and to perform comparisons. Approach of designing and implementing the algorithm will be explained in solution designed (Section 7) while comparisons between the two will be mentioned in result and evaluation (Section 9).

3.2 Hand Written Character Recognition

Alongside with solely implementing two neural networks algorithms, a character recognition software was written as a practical application to test the efficiency and accuracy of the networks. The networks were trained to recognised basic upper and lower case English characters, such decision would allow a further build up on interpreting words, which was beyond the scope of this project.

The form in which characters were recognised were via training images of English hand written characters. Digital images of hand writting English characters were collected from The Chars74K dataset [2][3]. 2860 English hand printed characters samples generated by 55 volunteers were introduced and the dataset was classified into 62 classes. The hand-printed data set was captured using a tablet PC with uniform thickness set matching the average

thickness. Hand printed character images were used instead of images of hand written characters, this is to minimise the effect of shading and colour inconsistency when scanning the image into the algorithm which would affect the accuracy during detection.

In order to train the collected data set, images were processed into a one dimensional binary representation where each pixel represented an input neuron in the network. Similarly, the output layer was designed to be a one dimensional binary array representing 62 classifications. In section 7 (Design Solution), detailed implementation decisions and design justification will be made on image processing.

4 Research

Artificial neural networks are defined as unprocessed electronic network of neurons based on the neural structure of the brain, where they process data one at a time and learn by comparing the record with a known classification. There are a few different ways comparing and predicting records.

4.1 Multi-layered Perceptron with Back Propagation

4.1.1 Topology

The topology of a multi-layered Perceptron(MLP) neural network is a multiple layers network with one input layer, one or more hidden layers and a single output layer. Each layer is linked by weights, these weights are real numbers expressing the importance of respective inputs to its output. Each perceptron will then weight up different kind of evidence in order to make decisions. Each layer in the network serves different functionality, the higher the layer of the network, the more complex and abstract will the layer be as it has to weigh input evidence from the previous layer, this also determines the expressive power of the network where it cause the neural network to fit the noise of input.

4.1.2 Activation Function

The aim of finding the desire output in a neural network is to allow the weighted sum from all the neurons in the hidden layers to pass through an activation function to determine how likely is the neuron going to fire. An activation function which was chosen in the project is the sigmoid function:

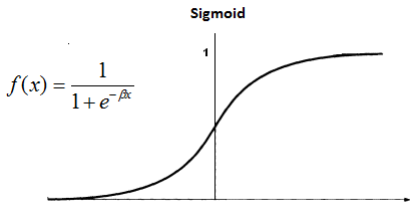


Figure 1: Sigmoid Function plot

where x-axis represents the weighted sum from the input neurons(x) and the y-axis is the sigmoid function result(f(x)). The smoothness of the

function represents the gradually modifying weights where the network get closer to the desire behaviour. The smaller the change in x, the smaller the change in the output will be. At each layer, each neuron will calculate the weighted sum of all incoming neurons and then pass through this function to produce an output for the next layer. This process is called the feed-forward model.

4.1.3 Learning Process

After performing feed forward model,the network will have to learn to approximate a function for all training inputs. The approximation is divided into two phases: back propagation and weight update. In back propagation, each output values is compared with the target value to produce a value of an error function. In classification, a cross entropy function is used.

$$Ece(w) = \sum_{p=1}^{outputsize} t^p log(y^p) + (1 - t^p) log(1 - y^p)$$

where t^p refers to the target output and y^p refers to the actual output the network compute. During training, this error function will be minimised by repeatedly updating the weight using this error until the performance of the network is good enough. The way weight is update is by differentiation.

4.2 Radial Basis Function

4.2.1 Topology

Similar to MLP network, RBF network possesses three layers. The input layer represents an n-dimensional training data expressed in the form of a vector, and every vector value is shown to the each neurons in the hidden layer. The hidden layer comprises of a set of vectors which is chosen from the training data set, they represent a feature which the network will learn to classify input data, they are called centres. Each input vector calculates an euclidean distance with the centres and the network determines their significance to that feature. Output layer is a set of node that represents different classes. Each output will compute a score for the associated category using a set of predefined weights and the final output is decided based on the highest score in the set.

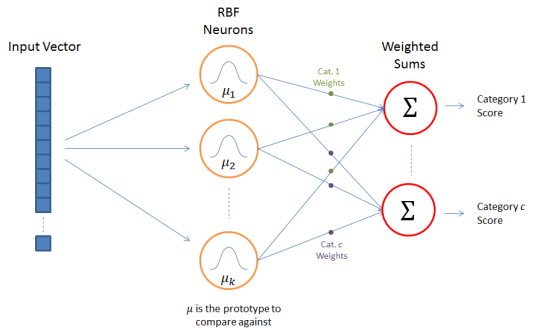


Figure 2: RBF network topology

4.2.2 Activation Function

As mentioned above the neurons in the hidden layer compute a notion of similarity between and the centres, this similarity is a range of values between 0 and 1. when the similarity is high, value is closer to 1, when two neurons are completely different, value tends to be 0. Such calculation of similarity uses the Gaussian function:

$$exp(-\frac{||x-C_j||^2}{2\theta_j})$$

Figure 3: Gaussian Function

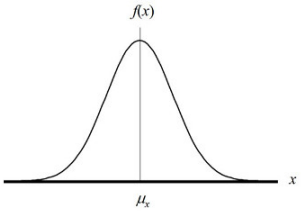


Figure 4: Gaussian Bell Curve

where x is the input, C_j is the jth centre in the hidden layer, and θ_j is the standard deviation controlling the width of the Gaussian bell curve. $||x-C_j||$ represents the calculation of euclidean distance. Each C_j is centred at the mean which is μ in the Gaussian curve. The hidden neurons will produce the largest response when the input has very similar features with the centre. As shown in the curve, the response of neuron will fall off exponentially when similarity reduces. This means that it has less influence over the classification decision[4].

4.2.3 Learning Process

The learning process involves two stages. First, several parameters have to be selected to decide the number of centres, selection of the centres from the training set and the width of the radial basis function curve. There are a few ways to select ways to select the parameter, decision will be justified in Solution Design(section 7). In the second stage, the weights associated to the hidden and output links have to be determined to perform the training process.

4.3 Related works

The aim of this project is to implement the two algorithms and try to look at their efficiency and accuracy on recognising hand written characters. In the past, similar field of work had been done using neural networks to perform classification which gave some insight on how to approach my project.

A similar project carried out in 2010 was about recognising writers based on their hand writings[5], a feature extraction method was used in order to process the hand writing images, such method gave me an insight on processing my data set to fit into the neural network. This paper also suggested the efficiency of RBF network is performing better than MLP, yet it had not show detailed comparison between the two neural network. Therefore this will be an area to look into.

Another project was to detect and loacte leaks in pipeline transporting gas using artificial neural networks[6]. In the paper, apart from comparing the two networks in terms of running time, the calculation of an maximum absolute error between the location of the real data and the predicted location would also influence the accuracy of predicting new data. Since the nature of this project is similar to this project, it will be worthwhile looking at the error produced so as to determine its accuracy.

Last but not least, the project on predicting sediment flow in rivers[7] also made use of neural networks to perform the task. It pointed out that MLP performed slightly better in terms of its mean square error. In the paper, the concept of robustness of neural network was also mentioned but had not thoroughly discussed. As it is very important to ensure my project can handle large amount of training and testing data, robustness was taken into consideration during the course of the project.

5 Problem Analysis

5.1 Hypotheses

After researching on the concept of the two neural networks and their related field work carried out by other people, several hypotheses about the neural network were made about comparing their efficiency and accuracy.

First of all, as mentioned in several papers above, the training and testing time for the neural networks has an effect on the network accuracy, therefore I would like to look at how time changes in training the network will affect the recognition rate.

Secondly, I observed that both neural networks had a very similar topology where they both contained a hidden layer and number of hidden neurons can vary. Furthermore, there are also rule of thumbs[8] on choosing the number of neurons. Therefore I hypothesised that varying number of hidden neurons would have an effect towards the recognition. And I would like to experiment on different amount of hidden neurons to see how accurate the network performs.

Last but not least, apart from time and number of hidden neurons, there are also several parameters contributing to the computation of a neural network, such as the learning rate as well as the number of epochs for training the network. Therefore I will also take these parameters into account during experimenting to see how they affect the result.

5.2 Helping tool

As mentioned previously the aim of the project was to evaluate the efficiency and accuracy of neural networks in the performance of recognising hand written characters. In order to quantify the training results, a graphical user interface was built to simulate the training and testing process. This helping tool allowed users to process images into binary data sets, visualise data sets on screen, as well as modify algorithm parameters for training.

A software requirement specification for this helping tool could be found in appendix A at the end of this report.

6 System Requirements

6.1 Software Languages

The choice of software language should proceed the ability to build up a neural network, perform certain amount of matrix manipulation and fetch suitable images and process them, whilst being compatible with major operating systems. The options were restricted to Java, C++ and Matlab as they were the three main languages I have had previous experience with.

6.1.1 Available Options

1. Java - This language is compatible with major operating system required as it can work on a virtual machine. Libraries for algrebra are easily accessible for matrix multiplication which will be required when implementing the Radial Basis Function. The swing library also provides sufficient methods for building a guiding tool to test my networks.

2. C++ - This language is compatible with major operating system and allows for the requirements of the system. It also has additional classes allowing greater functionality and additional exceptions for event handling. However, using C++ for high level program may be very complex. As the software designing does not require too many unique features provided by it, I would not recommend its use for this project.

3. Matlab - This language is advantageous for its very large database of build-in algorithms for image processing and computer vision application, which is suitable for character recognition. It is also powerful in matrix manipulation which is useful in my algorithm implementation.

6.1.2 Decision

With C++ removed from the decision, Java and Matlab remain in consideration. With greater personal knowledge, experience and confidence in Java, Java has been chosen for the development of the project. However, a similar piece of software could also be implemented in Matlab.

6.2 Libraries

This system requires the use of extended libraries to provide full functionality. The libraries used is listed below:

6.2.1 Matrix manipulation

As java is not proficient in algebra manipulation, external libraries is required to perform this function. Linear Algrebra for java(la4j) is open source that provides linear algrebra primitives and algorithms. This library allows for the implementation of radial basis function which required the technique of pseudo-inverse and transpose of a matrix to find the weights.

6.2.2 GUI widget

Swing is used throughout the entire implementation of the user interface as it provides sufficient foundations and methods of me to build a testable tool for my neural networks.

7 Solution Design

During the design phase of the project, important decisions were made in regards to which data structures would most suit for the two neural network algorithms. The data structures chosen were basic, low level components throughout the program. Standard Java implementation such as arrays and array lists were used to ensure reliability and to ensure correctness.

7.1 Neural Network

The topology of all neural networks in the project were the same. Each layer in the network was represented as a one dimensional array and each item in the array would store value of each neuron, each neuron represented a pixel in the character image. In between layers, weights were associated between them and each of them were linked to all neurons in the next layer. Such association was represented as a 2-dimensional array in java, for each dimension represented as a layer of neurons. The number of rows in the weight array was the size of input layer and the number of columns was the size of output layer.

For the number of neurons associated to each layer, the input layer consisted of 900 neurons, which was equivalent to the number of pixels in each input image. The number of output neurons consisted of 52 neurons, each representing a class of upper and lower case English characters. The number of hidden neurons were freely chosen, such that experiments would be carried out to look at how recognition would performed varying number of hidden neurons.

With regards to the activation function, a maths utility class(MathsUtils.java) was constructed to provide suitable activation functions as well as other mathematical computations, such as derivatives, cross entropy functions and matrix multiplication for the neural network classes.

7.1.1 Multi-layered Perceptron network

The multi-layered perceptron network(MLP) class(BackPropagation.java) was designed to perform learning and testing methods for the network. Initially, random weights were allocated in between all the three layers, such allocation was achieved by using the nextDouble() method in the java random Random class. Such decision was chosen as the random class provided a more flexible way to generate uniformly distributed random numbers.

Learning was performed in which two procedures were involved. Feed-forward model first took place to calculate the input value for each hidden

neurons. For each neurons in the hidden layer, a weighted sum was computed by adding up all the input values multiplying the associated weights. Afterwards a Sigmoid activation function was applied to each weighted sum in the hidden neurons, resulting an output value. Similar procedure would took place between the hidden layer and the output layer where the hidden neurons would make use of the output value as an input to calculate the weighted sum for the output layer. In the output layer, whose output neuron had the highest value would become the predicted classification of that input.

```
public static void feedForward()
{
    double weightedSumIH;
    double weightedSumHO;
    //(step 1)calculate the input value for each hidden neuron
    for(int h=0;h<HIDDEN_NEURON_NUM;h++)
    {
        weightedSumIH=0.0;
        for(int i=0; i<INPUT_NEURON_NUM;i++)
        {
            weightedSumIH += input[i]*weightIH[i][h];
        }
        hidden[h]=(MathsUtils.sigmoid(weightedSumIH));
    }

    //(step 2)calculate the output value
    for (int o=0;o<OUTPUT_NEURON_NUM;o++)
    {
        weightedSumHO=0.0;
        for(int h=0;h<HIDDEN_NEURON_NUM;h++)
        {
            weightedSumHO += hidden[h]*weightHO[h][o];
        }
        actual[o]=(MathsUtils.sigmoid(weightedSumHO));
    }
    return;
```

Figure 5: Feed Forward Model

The neural network would then learn by back propagation to update the weights. A cross entropy error function as well as a learning rate were chosen to apply a weight update process to each weight for each training pattern. Each set of updates to the weights for all the training patterns was called an epoch. Weight update would took place according to the assigned number of epochs, for which the error function would gradually reduce to a minimum.

```
//(Step 1)calculate error in output layer
for(int x=0;x<OUTPUT_NEURON_NUM;x++){
    outputError[x]=(target[x]-actual[x])*(MathsUtils.sigmoidDerivative(actual[x]));
}

//(Step 2)update weight in output layer
for(int o=0;o<OUTPUT_NEURON_NUM;o++){
    for(int h=0;h<HIDDEN_NEURON_NUM;h++){
        weightHO[h][o]+=(LEARNING_RATE*outputError[o]*hidden[h]);
    }
}

//(Step 3)calculate error in hidden layer
for(int h=0;h<HIDDEN_NEURON_NUM;h++){
    hiddenError[h]=0.0;
    for(int out=0; out<OUTPUT_NEURON_NUM; out++){
        hiddenError[h]+= outputError[out] * weightHO[h][out];
    }
    hiddenError[h] *= (MathsUtils.sigmoidDerivative(hidden[h]));
}

//(Step 4)update weight in hidden layer
for(int h=0;h<HIDDEN_NEURON_NUM;h++){
    for(int i=0;i<INPUT_NEURON_NUM;i++){
        weightIH[i][h]+=(LEARNING_RATE*hiddenError[h]*input[i]);
```

Figure 6: Back Propagation - Weight update process

7.1.2 Radial Basis Function network

The Radial Basis Function class(RadialBasisFunction.java)was designed to provide training and testing of the neural network. As compared to MLP, the network first selected a portion of data decided by the user from the training set as the hidden neurons(also known as the centres for a Gaussian distribution). The approach for choosing these data was by random. The reason for that was because it was quick and simple to implement and all the widths for the data would be equal and fixed at an appropriate size for the distribution of data points.

Regarding to choosing an appropriate size for the width of the data, maximum distance between the centres were considered to ensure that individual radial basis functions are neither too wide or too narrow for the given training data, this approach would gave reasonable result even for large training sets.

```
/*
 * This method calculates the variance
 * @param kernalSet - set of hidden neurons
 * @return variance - variance
 */
private static double calculateVariance(ArrayList<double[][]> kernalSet)
{
    double variance = maxDistance(kernalSet)/Math.sqrt(2*numberOfSamples);
    return variance;
}
```

Figure 7: Calculate the width of each RBF function

During training, weights between input layer and hidden layer were determined based on the Euclidean distance. It would then pass through a Gaussian activation function to compute the output for each hidden neurons. Weights between the hidden layer and the output layer were randomly allocated and then updated via back propagation. Output neurons value were computed by the weighted sum of the output of hidden neurons and the weights allocated.

Decision was made on choosing weights between the hidden layer and output layers. A fast linear matrix inversion technique called least squared error could be used to calculate the weights based on the fixed hidden unit activation. Even though such supervised training of basis function parameter would lead to better results than the unsupervised procedure explained above, the complexity for matrix inversion was enormous $O(n^3)$ and problems such as matrix not invertable might occur. Therefore, a random weight selection approach for hidden layer weights was chosen over least square method.

7.2 Image Processing

In order to train the neural networks, suitable data would be required to represent each input neurons. Images of English characters were preprocessed into a binary document containing a set of 30x30 pixel values as well as their target value representation. Four steps were involved in turning an image into the training data set.

7.2.1 Image Cropping

First, image cropping was done to trim away irrelevant content of the image which created a rectangular boundary along the target character. This was done by initially setting the top left pixel as a base line for cropping,

then go through each pixels to get its red green blue(RGB) value and decide if the pixel is out of the tolerant percentage of the colour in the left pixel. Tolerant percentage refers to the range of RGB values which the method would still consider as the colour suggested. If the percentage of the colour combination is greater than the tolerance value, the pixel will be preserved and it would then decide the corner coordinates of the newly cropped image for all those pixels preserved, cropping the image out.

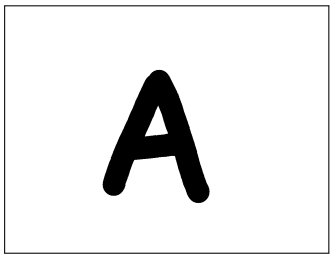


Figure 8: Original Image

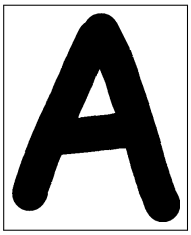


Figure 9: Image after cropping

7.2.2 Rescale Image

After cropping, the image with the bounded english character would be rescale to fit onto a 30x30 pixels bounding box while maintaining the aspect ration between the width and height. This was done by first comparing the width, if the image width was greater than the bounding width, the image width will became the bounding width. Afterwards, image height would have to scaled in order to maintain the aspect ration. The same strategy was repeated on the height to make sure all cases were considered.

```java
public static void getScaledDimension(Dimension imgSize, Dimension boundary) {

    int original_width = imgSize.width;
    int original_height = imgSize.height;
    int bound_width = boundary.width;
    int bound_height = boundary.height;
    int new_width = original_width;
    int new_height = original_height;
    // first check if we need to scale width
    if (original_width > bound_width) {

        //scale width to fit
        new_width = bound_width;

        //scale height to maintain aspect ratio
        new_height = (new_width * original_height) / original_width;
    }

    // then check if we need to scale even with the new height
    if (new_height > bound_height) {

        //scale height to fit instead
        new_height = bound_height;

        //scale width to maintain aspect ratio
        new_width = (new_height * original_width) / original_height;
    }
```

Figure 10: Method for image recalling

7.2.3 Image Padding

When the image was scaled with reference to the desire boundary, this character image would be padded onto a 30x30 white square image to make up a 30x30 pixels character image. This character image would be drawn starting from ((30-newWidth)/2)unit of the white square and ((30-newHeight)/2)unit of the white square.

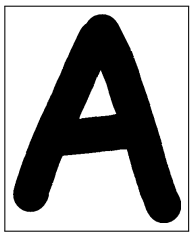


Figure 11: Original Image

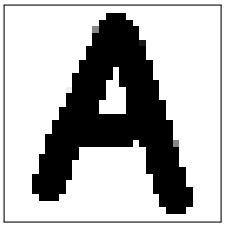


Figure 12: Image after cropping

7.2.4 Binary Conversion

The image was then converted into a binary image, which was represented by black and white pixels. Such conversion was achieved by a method in BufferedImage class which created a 1-bit image with a default colour space of $\{0,0,0\}$ and $\{255,255,255\}$[9]. Finally, each pixels would be read to output its value onto a .txt file.

Each image in the data set would be represented as an double array, the first array stored each pixel value in the image, while the second array stored the class of the image. Each class was represented by a 52 bits value (There were 52 classes in total).

7.3 Helping Tool

To facilitate the image processing and training process described above, a helping tool was designed to visualise the whole procedure. Several kinds of tools and java libraries were explored to improve the quality of the graphical interface. GridBagLayout was used as it was a very powerful and flexible layout management tool which allow finely customised user interface. All the features of this user interface were listed below.

7.3.1 Image Processing Panel

This image processing top panel allowed users to convert images of their choice into data sets which could be used as training or testing set. A directory of images could be chosen at once so that a single .txt file with all images data would be produced. User would also be needed to specify the name of the .txt file. Since both MLP and RBF neural networks were supervised, users were requied to name the each image files with the image's target output, otherwise the prediction would be wrong.

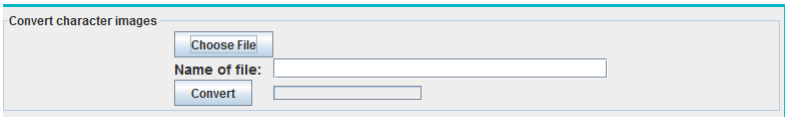


Figure 13: Image Processing Panel

7.3.2 Image Display Panel

When conversion was completed, users could make use of this panel to visualise the converted images in a 30x30 grid. On the left showed a JSe-

lection List which users can choose to display data sets from a particular class, in this project it would be upper case and lower case English characters. When users selected a particular class, all the data of that class would appear on the 30x30 grid on the right, where they can preview them one by one via clicking the next and previous buttons at the bottom.

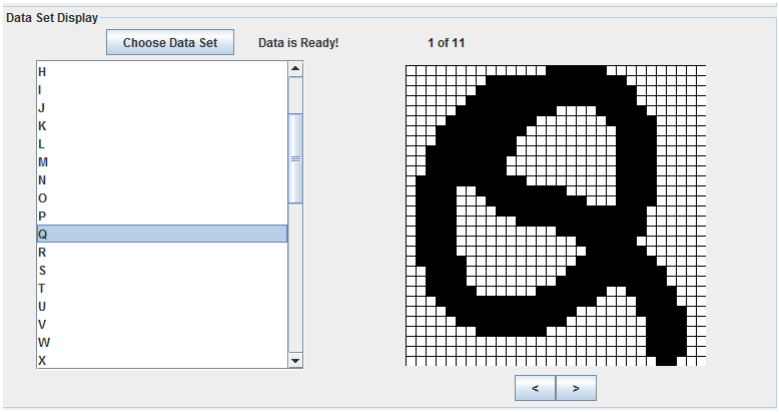


Figure 14: Image Display Panel

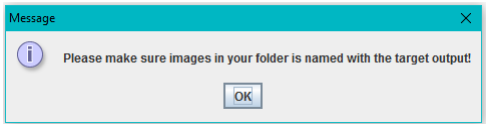


Figure 15: Warning to remind users to mane the image file according to target output

7.3.3 Neural Network Panel

The neural network panel allowed users to train and test the network using their own converted data sets. Several parameters were required to set before training. Different parameter combinations would be experimented and justified in results and discussion(Section 9)

The panel on the left and right represents the two neural networks respectively. Users would be able to fill in the parameters and the network would train according to the parameters set. During Training, an instant update line graph would show up indicating how the cross entropy cost function was reducing which contributed to how the learning would perform on testing unseen data sets. A progress bar would also be showed indicating

the training process. After training, execution time would be displayed in seconds.

During testing, new sets of unseen data would be provided to test the network and similar progress bar would show to indicate testing progress. After testing, a percentage of recognised characters would be displayed on the panel to indicate how well the network performed on the set of unseen data set.

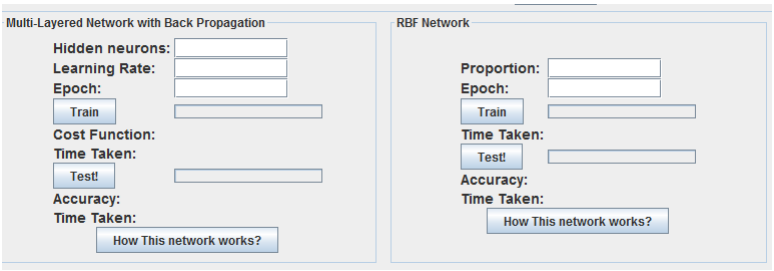


Figure 16: Neural Network Panel

7.4 Software Architecture

Throughout the implementation of the software, a model-View-controller(MVC) software architecture pattern were used to organise the code. such pattern aims to separate internal representations of different information so that separate components could be reused and developed at the same time without affecting each other[10].

The neural network classes (BackPropagation.java, RadialBasisFunction.java) was the model which stored data about all the neuron values, weights and output decisions. These data were retrieved from the controller and the view component. The helping tool, which was the graphical user interface classes(all classes under guiPanelPackage), represented the view. It was responsible to generate training statistics such as output error and time taken to train onto the panel based on changes in the model. The cross entropy function graph was also part of the view components. Last but not least,the controller class(Controller.java) was written as a controller to accept inputs and convent it to commands for the model and view. Methods for the controller class included handling button click actions from the user interface and then pass the all inputs parameter values to the neural network class for training and testing. In addition, the controller class also sent commands to the respective view to present the changes.

8 Implementation

8.1 Validation & Verification

Testing was a key part of this to ensure that bugs weren't introduced and classes creating had the necessary data and methods to provide the functionality needed. Often refactoring was necessary to ensure that the product stuck to the right design principles in mind such as being object orientated and having variables with clear names. Verification of the software was an important topic, ideas was considered as to whether certain changes were necessary or would provide an improvement over the current implementation. This helped to ensure the software aligned with the aims before attempting to implement any major features.

Validation of the software was undertaken throughout the course of development. In the start, it was ensured that necessary features were meeting the requirements and the scope of the project would capture these requirements. During the development of the neural networks, a number of testing methods were performed to ensure the algorithms were correctly implemented. Overall the project did fit the intended use and that our requirements that were met.

8.2 Testing

During the testing of my character recognition project, two kinds of test methodologies were carried out and were reported in detail below. Each type has brief explanation as well as the data collected for those test during development.

8.2.1 Test Coverage

Testing has been done to check errors within the whole application to make sure it is user friendly and meets the high level requirements as stated in the software requirement specification. Areas being tested include: 1) UI of application 2) All neural network algorithms 3) other Utility classes.

8.2.2 Test Method

2 testing methods were adopted within the project: JUnit Testing and Integration Testing. JUnit test was performed on specific component and classes such as the two neural network algorithms; methods were executed to ensure the function returns the expected output during training and testing.

This also ensures the correctness of the algorithms. Integration tests were used when combining different components together.

8.2.3 Pass/Fail Criteria

1. JUnit Testing - Build-in JUnit testing software was installed in eclipse to test for respective classes. It will pass the test if all the methods run successfully in the test cases. By using assertEqual, it can check if the return statements will return the expected answer whilst also checking that if exceptions are handled correctly.

2. Integration Testing- Different components successfully combined together without producing errors. Components which worked successfully when separated from other classes and features should continue to work as expected when having running as part of the project as a whole. This verifies that data across classes were not causing issues to each other.

8.2.4 JUnit Testing

A summary of all data from the JUnit test case can be found in appendix B at the end of the report. Throughout the course of the project, certain amount of test cases was created to minimize all possible errors in runtime. If error were found, it could be dealt with appropriately.

Radial Basis Function JUnit test - Test case for each method constituting this algorithm were created to make sure the network was correctly built with appropriate weights and chosen centres and variance. While performing matrix multiplications between each characters images, some matrices may not be able to invert due to its zero determinant nature. Therefore test cases were also created to make sure these situations were correctly handled.

Multi-layered Perceptron algorithm test – In order to test the correctness of my algorithm, a piece of open source MLP algorithm was used to compare their results[11]. Inputs and testing data sets were input into both the MLP source code and my code, similar output should be expected as the randomness of generating weights between the hidden neurons and output.

8.2.5 Inspection Method

The inspection method is used to evaluate the function and the feature of the helping tool user interface. The following is a list of features/functions which had been tested.

Test No.	Input	Expected Output	Output	Test Date
Display Character Panel				
1	Select display data set.	Document matches the display criteria.	Dataset correctly Display.	7 March 2017
2	Select other documents which are not the data sets ie., not a ".txt" file.	A warning should be displayed.	JOptionPane displayed, telling the user this is not a valid file.	7 March 2017
3	A ".txt" file is selected but is not the format of our data set.	A warning will be displayed when the user tries to select the list items.	JOptionPane displayed, telling the user this is not a valid data set file.	7 March 2017
4	Select another document to display.	The panel will clear up and allow new documents to be displayed.	Grid Panel clears up.	7 March 2017
5	Select a character from the list panel.	The corresponding letter will be shown on the grid	The corresponding letter is shown on the grid	7 March 2017
6	Select next or previous button on top of the screen.	The grid should change the sample of selected character.	Grid repainted and changed the sample of selected character.	7 March 2017
7	Hitting the end of the selection.	The grid should remain on the last sample.	When user continued to hit the next button, grid remained on the last sample selected.	7 March 2017
Multilayered Perceptron with back propagation panel				

8	Select train button.	A progress bar will start moving, after a certain time, training time indication will show up.	Progress bar start moving, after a certain time, time indication showed up and progress bar disappear.	7 March 2017
9	Select train without filling in parameters.	Training will not start, a pop-up window will show up asking user to fill in parameters	Pop up window showed up, showing which field is not filled.	7 March 2017
10	Select train with the wrong type of parameters.	Training will not start, a pop-up window will show up asking user to fill in parameters	JOptionPane appeared telling users to fill in the correct parameter type.	7 March 2017
11	Select train button	The user will not be able to click train button or test button until the training finishes.	Train and test buttons were disabled until the training is finished.	7 March 2017
12	Select Test button without training for the first time	Not able to test	JOptionPane appeared asking the user to train the network first.	7 March 2017
13	Select Test button.	JFileChooser showed up asking users to choose their testing file	JFileChooser showed up.	10 March 2017

14	Select Test button.	A progress bar will start moving, after a certain time, a testing indication will show up.	Progress bar starts moving, after a certain time, a testing indication will show up as well as the accuracy for recognising characters.	7 March 2017
Radial Basis Function Panel				
15	Select train button.	A progress bar will start moving, after a certain time, training time indication will show up.	Progress bar start moving, after a certain time, time indication showed up and progress bar disappear.	8 March 2017
16	Select train without filling in parameters.	Training will not start, a pop-up window will show up asking user to fill in parameters	Pop up window showed up, showing which field is not filled.	8 March 2017
17	Select train with the wrong type of parameters.	Training will not start, a pop-up window will show up asking user to fill in parameters	JOptionPane appeared telling users to fill in the correct parameter type.	8 March 2017
18	Select train button	The user will not be able to click train button or test button until the training finishes.	Train and test buttons were disabled until the training is finished.	8 March 2017

19	Select Test button without training for the first time	Not able to test	JOptionPane appeared asking the user to train the network first.	8 March 2017
20	Select Test button.	A progress bar will start moving, after a certain time, a testing indication will show up.	Progress bar starts moving, after a certain time, a testing indication will show up as well as the accuracy for recognising characters.	8 March 2017
21	Select Test button.	JFileChooser showed up asking users to choose their testing file	JFileChooser showed up.	10 March 2017

9 Result and Discussion

In this section, results on the performances of the neural network varying parameter set would be presented using visual data representation such as graph and charts. Justification will be made based on observations to see if it aligned with hypotheses made in first place. A detailed testing results could be found in appendix C.

9.1 Hypothesis 1: Effect of number of hidden neurons

Reminded that in project analysis, it was hypothesised that the number of hidden neurons in the hidden layer would have an effect on the accuracy of recognising English characters. For each neural network, the effect of hidden neurons to accuracy were evaluated.

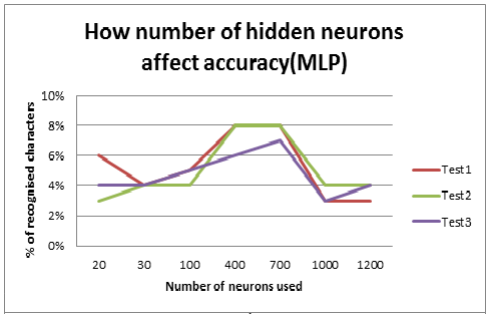


Figure 17: Effect of accuracy in MLP varying number of hidden neurons

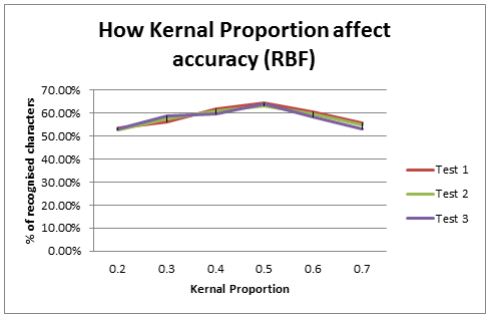


Figure 18: Effect of accuracy in RBF varing proportion of hidden neurons

For each neural network experiments, three tests had been carried out for each amount of hidden neurons. While the use of training set, testing set, number of epochs, learning rate, number of input and output neurons were kept constant:

- Number of training data: 2288
- Number of testing data: 572
- Number of input neurons: 900
- Number of output neurons: 52
- Epochs: 10,000

- Learning Rate: 0.2

Figure 13 showed the effect of accuracy in MLP varing different numbers of hidden neurons. In general, as the number of hidden neurons increased, there was a trend that the time used for training will increase proportionally, this was due to the increase of links for the network to compute the weighted sum and then passing through a sigmoid function.

In terms of the accuracy of testing data, there were little relationship between the values of the cost function to the accuracy. One possible reason for this was due to the effect on other parameters, such as the number of epoch as well as the learning rate. These parameters would be looked into in the next sub-sections. The low accuracy rate showed that the number of iterations for training the network were insufficient where total error between actual and target output was relatively large. More iterations had to be carried out in order to find out the true underlying effect on the number of hidden neurons. The only observation which could be seen from the testing data was that network tends to perform better when the amount of hidden neurons is between the number of input and output neurons. This might due to that fact that if there were too few hidden units, the quality of prediction would drop as the network did not have sufficient 'brains' to converge to the desired result, however, using too many hidden neurons would cause the network to remember the training cases rather than generalized it, causing a drop in accuracy on recognizing new data.

Figure 14 showed the effect of accuracy in RBF varying proportions of hidden neurons. In general, as the number of hidden neurons increase, the average time for training increase as the number of weights associated to the hidden neurons proportionally increased. As a result, the amount of weighted sum computation for each output neuron grew.

In terms of the amount of correctly recognised character, there was a trend that as proportion of hidden neurons increase from 0.2 to 0.5, the amount of accuracy detected improved, and the deviation of accuracy is around plus and minus 2%. However when the proportion further grow from 0.5, the percentage started to reduce. One possible reason for this phenomenon was because when amount of hidden neurons increase, more features from the same characters were extracted and summed up together, resulting in a higher contributions, therefore when there were new inputs, the distance between with the radial basis function would be very small which tends to fall on the correct category. However, selecting too much radial basis functions (hidden neurons) would result in overfitting, where network tends to remember categories of training set instead of generalizing into an

approximating function. Overfitting would then deteriorate the ability for the network to classify unseen inputs.

9.2 Hypothesis 2: Effect on learning rate

As mentioned in hypothesis 1, other parameters might also affect the accuracy. Learning rate controls how fast the network is learning within the assigned epoch, the higher the number, the faster the network will learn and vice versa. As learning rate only applied to MLP neural networks, the discussion would only focus on the results in MLP, but such parameter would be taken into account for later comparisons on RBF.

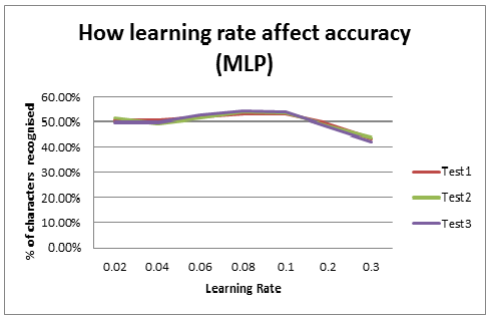


Figure 19: Effect of accuracy in MLP varying learning rate

Learning Rate		Test 1	Test 2	Test 3
0.02	Cost Function:	0.86	0.76	0.83
	Accuracy:	50.44%	51.34%	49.78%
0.04	Cost Function:	0.36	0.45	0.29
	Accuracy:	50.76%	49.24%	49.54%
0.06	Cost Function:	0.36	0.28	0.34
	Accuracy:	51.92%	51.57%	52.85%
0.08	Cost Function:	0.24	0.31	0.34
	Accuracy:	53.12%	54.06%	54.35%
0.1	Cost Function:	0.25	0.32	0.28
	Accuracy:	53.23%	53.4%	54.16%
0.2	Cost Function:	0.14	0.23	0.15
	Accuracy:	49.36%	48.51%	47.95%
0.3	Cost Function:	0.12	0.15	0.25
	Accuracy:	43.35%	43.7%	42.04%

Figure 20: All test results for different learning rates

Three tests were carried in each of the learning rate, the number of hidden neurons and epoch had been kept constant for the experiment. The choice of the number of neurons is based on the best result collected from the previous experiment:

- Number of training data: 2288
- Number of testing data: 572
- Number of input neurons: 900
- Number of output neurons: 52
- Epochs: 10,000
- Number of hidden neurons: 300

First of all, the use of a learning rate was to ensure convergence. If the learning rate was set low, the network would learn very slowly and may fail to approach a local optimal minimum within the training progress. Yet if a learning was set too high, each step down size is too big, there will be a chance to overshoot the optimal point, and never settling but oscillating about the point.

Based on the testing result, it showed insignificant changes in accuracy between 0.02 to 0.06 learning rate and their respective accuracy were around 49% to 51%. More so, there was a slight increment in accuracy when the learning rate was adjusted from 0.08 to 0.1. After that any learning rate higher than 0.2 will result in a drop in accuracy. This could deduced that the network could be optimized with 0.08-0.1 learning rate. Yet, choosing such a small step size might result a higher iterations in order to obtain a better accuracy. The following test will investigate how the number of iterations will affect testing accuracy.

9.3 Hypothesis 3: Effect of epoch in training time and accuracy

Last but not least, this subsection looked at how the number of epochs used in training neural network would affect the training time as well as the accuracy.

Three tests were carried out to for different amount of epoch to see how the epoch changes with the recognition rate. The number of hidden neurons, learning rate and the testing and training data set were kept constant. The number of hidden neurons and the learning rate chosen were based on the best result gathered from previous experiments.

- Number of training data: 2288
- Number of testing data: 572
- Number of input neurons: 900
- Number of output neurons: 52
- Proportion of hidden neurons(RBF): 0.4
- Learning Rate(MLP): 0.08
- Number of hidden neurons: 300

9.3.1 Effects on MLP network

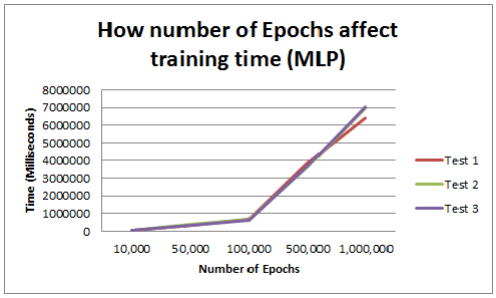


Figure 21: Effect on training time varying epochs

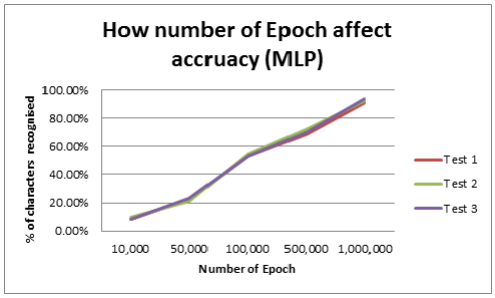


Figure 22: Effect on accuracy verying epochs

According to the testing result in figure 18, there was a significant relationship between the number of iterations and accuracy for recognizing unseen data. As the number of epoch increases, the percentage of unseen data being recognised increases proportionally. On top of that, the cost function for each test also reduced when the iteration continued, meaning that the network was trying to converge to the minimum. However, the time taken to train the network had proportionally increased when number of iteration went up as shown in figure 17, in order to reach 70% of accuracy, the network needed 50 minutes to train. As compared to RBF network, MLP network was slower in terms of training in order to result in more than 60% accuracy.

One of the reasons for the long time to achieve accuracy was due to the weight updating algorithm. Back propagation was a single global supervised algorithm which required the comparison of the input and output mapping. This slowed down the computation. Where as compared to RBF network, the learning of the first layer is unsupervised, which allowed them to make good use of unlabeled data without worrying about points which were less similar.

9.3.2 Effects on RBF network

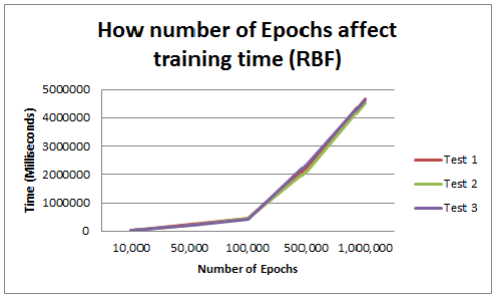


Figure 23: Effect on training time varying epochs

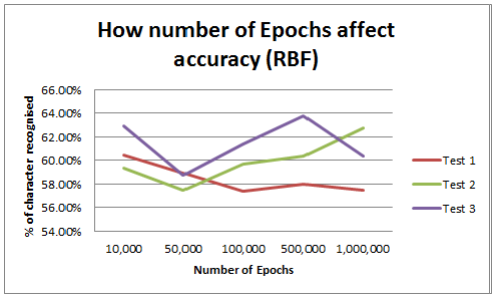


Figure 24: Effect on accuracy varying epochs

Similar to MLP, as the number of epochs increases, the time used for training will increase proportionally, as shown in figure 19, the time went up at a higher rate after around 100,000 iterations. Despite having an increase in training time, the training accuracy on recognising unseen data had little to no relationship with the number of epochs. The percentage of new data recognised maintained at around 58% to 64% no matter how the number of iterations changed. It looked like the accuracy of a RBF network was less significant than that of MLP.

One possible reason for that was due to the effect of over-fitting of the data. During training, as the number of iterations increase, training data would be replicated to estimate the weights of the network. On top of that, there were many patterns to be classified (52 upper and lower case English Characters) and the number of training data might not be sufficient. Therefore it would be relatively easy for the network to manipulate the weights to reproduce the training set but less likely to learn characteristics of new data. Generalisation strategy should be implemented to avoid the problem of over-fitting.

9.4 Discussion

Based on the above experiments and test, the performance of the two neural networks differed varying different factors. In terms of of efficiency, both algorithms would yield a longer training time when the number of epochs increases, however, RBF network performed better in terms of training. The average time for RBF to train the network in a certain iterations took roughly 2/3 the time of a MLP network used.

In terms of accuracy, both algorithms were able to detect more than 60% of the testing set within a combination of parameters. But MLP performed better as the accuracy could go up to 93%, yet the highest rate RBF network could detect was 63%. Such conclusion could be argued based on the issue of generalisation which will be evaluated in the next section.

10 Evaluation

10.1 Quality of Product

In terms of the neural networks, both algorithms were built with maximal simplicity such that the resulting program would encounter fewer subtle bugs even when unexpected inputs were encounter during prolonged execution. Also, the algorithms were documented clearly within the program, such as putting java doc on every methods and classes, this ensured the code could be easily maintained by others.

In terms of the quality of the graphical user interface, the sets of qualities specified during the start of the project were revisited to evaluate whether the final product met those attributes. A quantitative evaluation technique called feature analysis(detailed in the next sub-section) was carried out to identify the requirements that users suggested supported what the interface should possess.

10.2 Quality Evaluation - Feature Analysis

From the start of the project, crucial attributes were mentioned. All attributes were well balanced throughout the course of the project.

- Reusability - The organisation of the code in the program could be easily reused on pattern recognition rather than limited to English character recognition, the number of input and output neurons could be altered to fit into recognising other patterns without affecting other functions. The process of image processing could also be reused independently without affecting other parts of the program.

- Testability - As the entire program was software engineered, one very important procedure was to test the validity and verification of the piece of software. The use of JUnit and integration testing supported such attribute.

- Usability - The helping tool was designed to provide a platform for user to perform neural network training and comparisons. In terms of learnability, step by step learning guide on how network train was embedded into the interface. In terms of user interface, inspection tests ensured edge cases were handled smoothly without any errors.

10.3 Improvements and Extendability

Due to low coupling between classes and the use of MVC design pattern. Neural network functionality and user user interface classes were kept separate. Therefore, any additional features or improvements could be easily amended and keeping the a consistent style.

10.3.1 Improvements

1. In order to solve the problem of generalisation of algorithm. A k-fold cross validation method should be used in order to prevent overfitting during training of neural network. Such improvement could be made by separating the training data set into a k-fold validation set, for each fold, declare a one of the validation set and train the rest of the model yield a validation error, finally averaging the validation error to obtain the cross validation error of that neural network. By comparing the cross validation error between different training methods, we could estimate how well the algorithm performed.

2. Another improvement that could be made was to provide an instant hand writing panel for user to write their own character to test rather than converting exising images into binary data type. This would reduce the process time and help user better understand how the algorithm works.

3. Use of JavaFX would be an improvement to build a better visual helping tool in visualising the neural network training and testing process. As JavaFx would provide a richer and more sophisticated graphical user interface.

10.3.2 Extendability

1. Learning tool - This project could be further extended into a learning tool which would help university students who were taking the neural network course to get familiar with the two supervised neural networks, especially in exploring more on how MLP and RBF would differ in terms of their training efficiency and test accuracy.

2. Recognising different pattern - Since the project intended to look at performance of neural network, it could be easily extended to recognise patterns other than English hand written characters. In addition, it

can also be extended to recognise more meaningful patterns such as words, or phases.

11 Summary and Conclusion

To sum up the conclusion of this project, the objective of the dissertation is to compare and evaluate the performance of 2 neural networks: multi-layered perceptrons neural network and Radial Basis Function neural network on their performance in training the testing hand written English character recognition. Specifications and requirements of the project were initialised at the start of the project but were subjected to modifications during the course of the project, yet it was still comparable to the initial design.

All requirements defined in the specification were met while the implementation of the graphical user interface were well designed and structured to minimise bugs and errors. The implementation of the user interface was meant to provide a helping tool to input combinations of parameters to experiment the effect on the two networks. It also allowed me to put in practise many different evaluation strategies such as JUnit, usability, and integration testing. In addition, codes were made adaptable and readable by well documenting and putting java doc on all code written.

Based on the experiments and comparisons between the two networks, both of them showed a positive relationship between the number of hidden neurons and the rate of accuracy. The number of hidden neurons would better recognise the data for a range between the number of input neurons and output neurons. Secondly, the effect of learning rate will also affect how well the network weight will converge to recognise new unseen data and the MLP network should set the rate at around 0.08 on the set of training data to yield the best result. Finally, by measuring the time taken to training and test the network, the RBF network showed a better performance in testing efficiency under the same setting with MLP network, however MLP network showed a greater degree of recognition rate despite its long training time.

Towards the end of the project, challenges were experienced about realising the problem of generalisation, where over fitting of the data were encountered. Even though the program was not completely perfect, such issue was mentioned during the demonstration and was well documented as an extendability. With the limited amount of time in completing the coding as well as thorough testing, the result would have been more successful if more time could be given.

Overall, I had a good experience working on an individual research project, gaining new skills and improving on software engineering techniques. Efforts had been made to work towards schedule as well as communicating effectively with my supervisor.

12 User Guide

All the files needed to run the user interface and the algorithms are available at https://git-teaching.cs.bham.ac.uk/mod-40cr-proj-2016/myc304. The .zip file submission for my project includes:

1. CharacterRecognition folder for the main project
2. JUnit testing folder
3. Final Report

The CharacterRecognition folder contains a source folder which holds all the classes regarding the algorithms and the user interface. In the source code file, the user interface were subdivided into different panels which were represented in individual files. The main running class is in the mainGUI folder. There are several options to run the user interface:

1. Run in an IDE: To run the classes in an IDE such as eclipse, copy the entire content from the source folder to a project in the IDE, and then run the MainGUI.java class in the main GUI folder. Note: do not copy the subfolder itself; open the folder and copy the content.
2. Compile and run on the command line: The classes can also be complied using a command line. The command *javac *.java* compiles the classes from the folder. Your compiler should be able to support Java 7 or higher in order to compile it. To run the whole user interface, use the command *java MainGui*.

More description of the folders inside CharacterRocognition could be found in the README.txt file.

The JUnit testing folder contains a separate project for JUnit testing This aims to provide a cleaner environment for testing. The RBFTest.java is the main JUnit testing class and the rest were the network and utility classes duplicates from the main project.

13 Bibliography

References

[1] Christos Stergiou and Dimitrios Siganos *Neural Networks* https://www.doc.ic.ac.uk/ nd/surprise_96/journal/vol4/cs11/report.html#Why use neural networks *Access date: 06 March 2017*

[2] The Chars74K dataset *Character Recognition in Natural Images* http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/ *Access date: 14 January 2017*

[3] T. E. de Campos, B. R. Babu and M. Varma *Character recognition in natural images,* In Proceedings of the International Conference on Computer Vision Theory and Applications February 2009.

[4] Chris McCormick *Radial Basis Function Network(RBFN)Tutorial* http://mccormickml.com/2013/08/15/radial-basis-function-network-rbfn-tutorial/ *Access date: 20 February 2017*

[5] J.Ashok, Dr.E.G.Rajan *Off-Line Hand Written Character Recognition Using Radial Basis Function* Int.J. Advanced Networking and Applications 792, Volume 2, Issue 4, Pages 792-795,2011

[6] Rejane B. Santos, Markus Rupp, Santiago J. Bonzi, Ana Maria F. Filetia *Comparison Between Multilayer Feedforward Neural Networks and a Radial Basis Function Network to Detect and Locate Leaks in Pipelines Transporting Gas* CHEMICAL ENGINEERING TRANSACTIONS VOL. 32, 2013 The Italian Association of Chemical Engineering

[7] Hadi Memarian, Siva Kumar Balasundram *Comparison between Multi-Layer Perceptron and Radial Basis Function Networks for Sediment Load Estimation in a Tropical Watershed* Journal of Water Resource and Protection, 2012, 4, 870-876 October 19, 2012

[8] Mic *Selecting the number of neurons in the hidden layer of a neural network* https://www.r-bloggers.com/selecting-the-number-of-neurons-in-the-hidden-layer-of-a-neural-network/ Access Date on March 19 2017

[9] *Class BufferedImage Jave Platform Standard Ed.7*

[10] Trygve Reenskaug, James Coplien *The DCI Architecture: A New Vision of Object-Oriented Programming 20-03-2009*

[11] John McCullock *BackPropagation**
`http://mnemstudio.org/neural-networks-4x6x14.htm`
`Access Date on Feb 5 2017`

14 Appendix

A Software Requirement Specification

University of Birmingham

School of Computer Science

Comparing Neural Networks for Hand Writing Recognition

Software Requirement Specification

Author:
Chan Man Yi 1391904
BSc Computer Science

Supervisor:
Hamid Dehghani

April 1, 2017

Contents

Revision History

Name	Date	Reason For Changes	Version
Chan Man Yi	25 February 2017	polished the whole specification again to further fit the objective of the project	3
Chan Man Yi	18 February 2017	Changed scope of project, instead of asking users to draw characters, prepared training sets and testing sets were used.User Interface changed.	2

1 Introduction

1.1 Purpose

The purpose of this project is to present a hand written character recognition tool using the basis of artificial neural networks. This problem will be solved using a classic multi-layer neural network with backpropagation as well as Radial Basis function in Java.
This software requirement specification documents the overall software engineering description of the character recognition, external interface requirements, system features and non-functional requirements. Product schedule will also be attached at the end.

1.2 Document Conventions

Abbreviation	**Meaning**
ANN	Artificial Neural Networks
GUI	Graphical User Interface
HCI	Human Computer Interaction
JRE	Java Runtime Environment
JVM	Java Virtual Machine
API	Application Programming Interface
MLP	Multi-layered Perceptron
RBF	Radial Basis Function

1.3 Intended Audience and Reading Suggestions

This document is intended for project managers and developer who are working with the project. The sequence for reading the document is as followers:
Project Manager: managers should look at section 2 to 5 to gain necessary information for managing the flow of the project.
Developer: Developers should look at the overview and consequently focus on the requirements and system features in details.

1.4 Product Scope

This application will provide an platform on training character data sets, and allow users to train and test how neural network performs varying different parameters.
The aim of it is to explore the effect of recognizing unseen hand write characters varying different training parameters and the organisation of neural networks. The software will include a functional GUI suitable for visualisation and navigation.

2 Overall Description

2.1 Product Perspective

This product is a brand new self-contained piece of software inspired by the optical character recognition technique used in wand reader. It was decided to build the product using artificial neural network due to own previous experience on learning machine learning techniques in the past academic year. It was agreed that in retrospect, building such application would be of great help in fully understanding the context of artificial neural network.

2.2 Product Functions

1. Ability to add training data sets of hand written characters into the application, and view them on screen one by one.
2. Based on the training set imported, User can customize the learning rate, number of hidden layer and the number of epoch to build a neural network and train the data.
3. After training, users can try to test it using a prepared training set.
4. Upon detecting characters, product can also display statistical data about the training process, which allow users to understand how accurate is the neural network performing.

2.3 User Classes and Characteristics

1. Users with knowledge of neural networks: University students taking the Neural Network or Machine learning modules will utilize all the functions of the software as it has been designed to provide a solid understand on the logistic regression problem, which is the module's classic topic. This user class will have medium to high level of technical understanding with the algorithms the software presents. Therefore, the product will present sufficient details to allow this user class to gain a solid understanding on the procedure of training a neural network.
2. Users without knowledge on neural networks: Application may be used by non-computer science students, like people who started to understand machine learning or biology students who want to understand how neurons in our brains solve problems in a computer program. Even though this user class is not our most important user group, the application will still be beneficial to them as it shows how newly insert characters can be recognised based on learning a set of similar characters.

2.4 Operating Environment

The software is expected to run on all machines and operating systems manufactured within the past 15 years as long as a recent version of JRE is installed. The software should be capable of running on a system with low-end hardware and not be too demanding on the system. In addition, it should also require only a generous amount of memory to install due to the amount of training and testing sets available.

2.5 Design and Implementation Constraints

Functional constraints:

1. Set of training data: Due to the scale of the project, the number of data sets for training cannot exceed the memory limit in which the program can process, or else the application will become computationally slow and inefficient, affecting the overall performance. Therefore the amount of hand written characters in our database has to be limited.

2. Size constraint for writing panel: Since the aim of the project focuses on the understanding of the neural network computations, extracting characters and words from images will not be discussed in this context, therefore, all input and output characters are limited to a fixed size grid panel with each grid representing a pixel of an image. And all input datasets are presented in mono colour space.

Non-functional constraints

1. Time constraints: This project is subject to due in 10 weeks' time, therefore, richness and depth of the application may be constrained to a certain level.

2. Experience constraint: This project is completed solely in Java due to a lack of experience in other programming languages.

2.6 User Documentation

A simple instruction page will be accessible from a "Help" button on the GUI of the software. This will explain how to use train the data sets and how to interact with the character recognition panel. During development, technical documentation will be created which describes the visualizes the design of the software.

2.7 User Interfaces

The interface of the program involves 3 sections; A character display panel, a panel constructing multi-layered with back propagation neural network; and a panel constructing radial basis function neural network.

Character Display Panel:

	Screen Name	Description
1	Add data set	Select a training data set from database and then display it one by one on the grid panel. On top of this panel, a selector is presented to show the sequence of letter showing.
2	Character Selector	Characters are recorded on a list panel indicating each character in the training data set. Users can select each character to show the data set on screen.
3	Grid	A 30x30 grid display used to display dataset.

Panel constructing multi-layered neural network

	Screen Name	Description
1	Parameters Panel	Add, update and modify parameters for training data sets. Including 1) Learning rate, 2) Number of hidden units, and 3) Epoch.
2	Help	Guiding users to set up the neural network and how the network works.
3	Train	Start the training process.
4	Test	Start recognizing unseen characters based on the training set.

Panel constructing radial basis function neural network

	Screen Name	Description
1	Parameters Panel	Add, update and modify parameters for training data sets. Including 1) Number of centres, 2) width of Gaussian function.
2	Help	Guiding users to set up the neural network and how the network works.
3	Train	Start the training process.
4	Test	Start recognizing unseen characters based on the training set.

2.8 Hardware Interfaces

Since the project is mainly written in Java, hardware requirements are limited to devices which are capable of running the JVM as well as a mouse and monitor to control the training settings and displaying the recognised character respectively.

2.9 Software Interfaces

The software will not be reliant on other piece of external software to run on. The set of training data which is presented in binary will be stored within text

files as an inport into the application. Any API used will be identified upon completing the project.

3 System Features

3.1 User Interface

The Interface involve three panels; one displaying the character data sets(top panel), one specifying the parameters for the neural network (bottom left panel); and finally the training and testing buttons for users to run the neural network(bottom right).

3.2 Training UI (Main Panel) Features.

	Feature	**Description**	**Priority**	**Functional Requirements**	**Stimulus/ Response**
1	Adding data set into neural network	Button to add a training set to prepare building the neural network	9	Each character in the data set must be of the same size and is in binary form	When the user choose the data set, a file chooser will pop up for the user.
2	Training set character display panel	A panel showing each character in the training set.	7	Each character is displayed in a grid panel. Ability to switch to next character. A text panel listing all the training data.	Each character is diaplayed in a grid panel upon user selecting the training data set.
3	Training parameter panel	Options provided for users to set the learning rate, number of hidden nodes, and epoch to build the neural network.	9	Text boxes are provided for users to fill in all the parameters. Pop up box should show up when one of the boxes are not filled.	All three parameters will constitute to the learning algorithm in calculating the weights and output values.
4	Guideline for training and testing neural network	A demonstration on how each neural networks perform training	6	Users can study about how MLP and RBF network works	An Animation or slides will be provided for users to preview the slides.

4 Other Nonfunctional Requirements

4.1 Performance Requirements

1. The system should interpret the java graphics interface (Testing data sets) to generate a graphic representation on the panel in no longer than 60 seconds, even with the maximum number of data sets. This performance will ensure the system to response efficiently when interacting with the user.
2. Under normal working conditions, no matter what operating system user is using, the system should not delay its response for more than 2 seconds.
3. Upon detecting user's input in the character recognition window, the system should not take more than 5 seconds to classify the character and display the percentage on the accuracy panel.

4.2 Error Handling and Reliability

1. When an inout error occurs, they system must display an error message to the user and the user must acknowledge the error before they can run the program. For example, when user did not fill in any of the neural networks parameters, window will pop up to prompt users to fill it in before training.
2. The system should be able to catch any exceptions when a user input will cause an infinity loop. Therefore a limit for the parameter has to be set in order to maintain its reliability.
3. The system needs to be able to create a correct neural network based on users inputs every time, so users will not be confused.
4. The software should be designed in such a way that serious errors do not occue however should a fatal exception happen the program should close and produce a detailed description of the cause of the error.

4.3 Security Requirements

No security requirements have been identified because there is no collection of any personal data for the project and there are no external policies or regulations which affect the software.

4.4 Software Quality Attributes

1. Flexibility: This application must be flexible in term of building different neural networks. As different networks may have slightly different topology and activation function, the interface and data processing must be flexible to adapt to those changes.

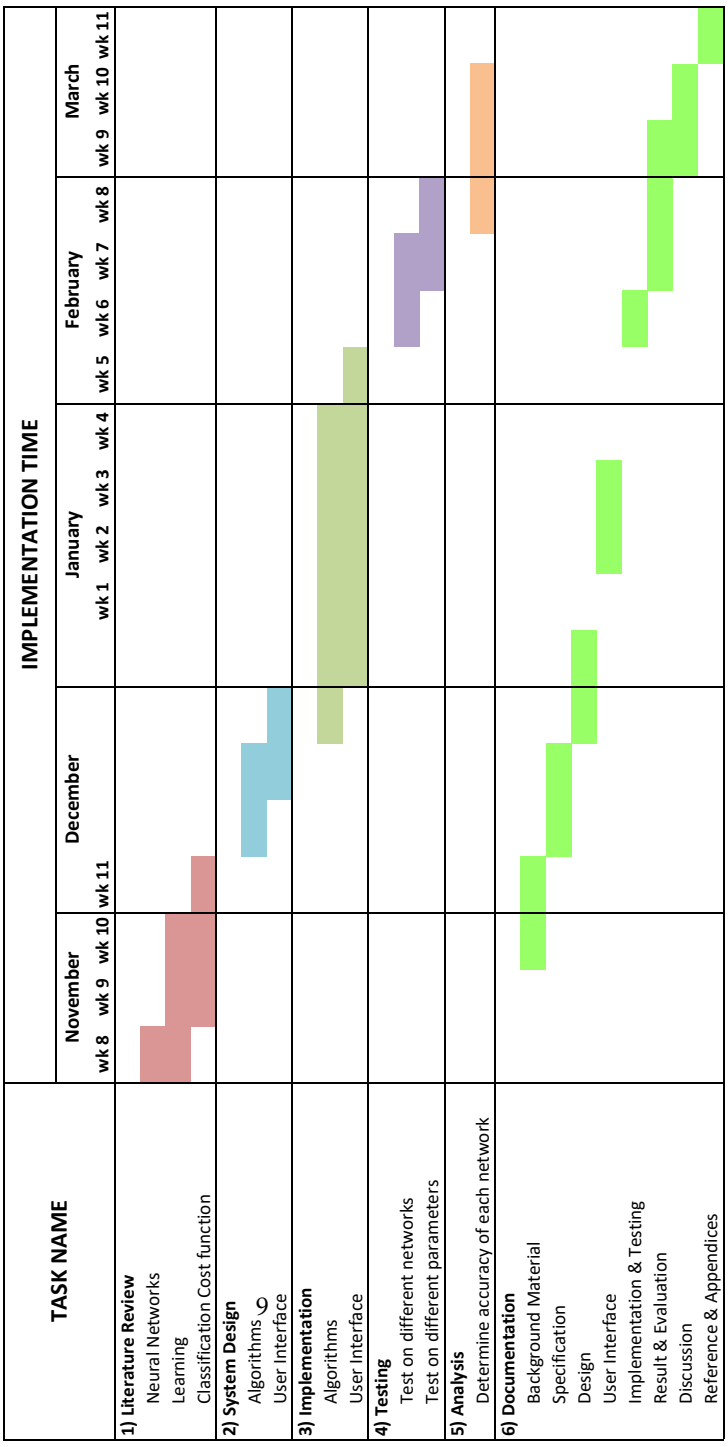
2. Reusability: Since the aim of the project is to evaluate the different classification technique for neural networks, such application can be reused in other character recognition applications using the most efficient neural networks to classify letters. Such application can also be extended to recognize characters other than English letters, such as Chinese characters or Latin Characters.

3. Robustness: This is an important attribute to ascertain for developers as well as users. Allowing the user to input data into the program, the application needs to handle all possibilities, including edge cases. If not, the application may crash and will; fail to provide accurate classification of characters.

4. Testability: Being able to test the accuracy of how neural networks recognize character is the key to evaluate which neural network would give a better clarification result. For which is also the main focus of the project. Therefore the piece of software should be easily configured in order to perform testing.

5. Usability: This is essential for the users as no matter how 'good' the application is, if the application appears to be too complicated with lots of information provided, users may not receive the full benefit for recognizing characters, learning to partial or incorrect knowledge. The interface must therefore be user friendly and easy to follow.

5 Product Schedule



6 Risk Analysis

Risk	Probability	Effect	Type of Strategy	Strategy
Underestimate complexity of feature to be implemented	High	Other features may not be completed in time or not as successful as expected. May also reduce time available for testing	Minimisation Strategy	Avoid aiming features which are too complex. Begin with something less complex and build up on it. If a feature does become too difficult to solve, either reduce it's complexity of move on to another feature.
Code are deleted or corrupted	Medium	Work may be lost and can't be recovered which will set back progress. Functions which work fine may be affected by newly built functions	Avoidance Strategy	Keep code stored within the remote repositories and on multiple devices. Regularly update code to keep track on the latest commits. Make use of the branching function in Git Hub to work on new features and then merge to main branch when features are completed.
Unable to complete feature due to a lack of understanding	Medium	This may set back the project if feature cannot be completed on time A feature may need to be removed entirely. If features affect other parts of the software, it will affect overall result of the project.	Minimisation Strategy	Make a literature review for better understanding before beginning to work on them. Can make use of supervisor meeting to ask about the relevant concept/algorithm. Peer-to-peer help.

B JUnit Test Record

B JUnit Test Record

Test Case	Description	Input	Output	Expected Output	Pass/ Fail
Maximum Distance Test	Find the maximum distance between all the hidden neurons- This contributes to build up activation function.	All hidden neurons {{0.0,0.0},{0.0}} {{1.0,1.0},{0.0}}	√2	√2	Pass
Calculate Variance Test	Calculate the variance- This forms the width of the RBF function.	All hidden neurons {{0.0,0.0},{0.0}} {{1.0,1.0},{0.0}}	√2/2	√2/2	Pass
Index Test	Return the index of the target output.	{0.0, 0.0, 0.0, 1, 0.0, 0.0, 0.0}	3	3	Pass
Sample to Kernel Distance Test	Compute the distance between each input sample with the sample in the hidden neuron.	Sample: {0.0,1.0}, Hidden neurons: {{0.0,0.0},{0.0}} {{1.0,1.0},{0.0}} Variance: √2/2	{1.0,0.36787944117144424,0.36787944117144424}	{1.0,0.36787944117144424,0.36787944117144424}	Pass
Transpose Matrix Test	Calculate the transpose of a matrix.	$\begin{bmatrix}1,2,3\\4,5,6\\7,8,9\\10,11,12\end{bmatrix}$	$\begin{bmatrix}1,4,7,10\\2,5,8,11\\3,6,9,12\end{bmatrix}$	$\begin{bmatrix}1,4,7,10\\2,5,8,11\\3,6,9,12\end{bmatrix}$	Pass
Inverse Matrix Which Will throw Exception Test	Ensure if the matrix cannot be inversed, Illegal Argument Exception will be caught.	$\begin{bmatrix}1,2,3\\4,5,6\\7,8,9\end{bmatrix}$	Illegal Argument Exception: matrix cannot be inversed.	Illegal Argument Exception: matrix cannot be inversed	Pass
Matrix Multiplication Test	Check matrix multiplication.	$\begin{bmatrix}1,2\\3,4\\5,6\end{bmatrix}\begin{bmatrix}1,2,3\\4,5,6\end{bmatrix}$	$\begin{bmatrix}9,12,15\\19,26,33\\29,40,51\end{bmatrix}$	$\begin{bmatrix}9,12,15\\19,26,33\\29,40,51\end{bmatrix}$	Pass

Pseudo Inverse Test	Calculate the pseudo inverse of a matrix.	$\begin{bmatrix} 1,1 \\ 1,2 \\ 1,3 \\ 1,4 \end{bmatrix}$	$\begin{bmatrix} 1,0.5,0,-0.5 \\ -0.3,-0.1,0.1,0.3 \end{bmatrix}$	$\begin{bmatrix} 1,0.5,0,-0.5 \\ -0.3,-0.1,0.1,0.3 \end{bmatrix}$	Pass

C Performace Test Results

C Performance Test Results

Radial Basis Function

Amount of kernel proportion to testing rate

Number of Epoch: 10,000

Number of training data: 2288

Number of testing data: 572

Kernel Proportion		Test1	Test2	Test3
0.2	Time to train (milliseconds)	22589	23480	21814
(457 data)	Accuracy (%)	53.36%	52.44%	52.97%
0.3	Time to train (milliseconds)	39456	37095	31537
(686 data)	Accuracy (%)	56.11%	57.51%	58.74%
0.4	Time to train (milliseconds)	40841	41487	43042
(915 data)	Accuracy (%)	61.71%	60.95%	59.73%
0.5	Time to train (milliseconds)	58249	55746	65798
(1144 data)	Accuracy (%)	64.51%	63.11%	63.81%
0.6	Time to train (milliseconds)	72572	73671	76972
(1368 data)	Accuracy (%)	60.45%	59.52%	58.30%
0.7	Time to train (milliseconds)	101019	100405	100007
(1601 data)	Accuracy (%)	55.72%	54.67%	53.16%

Multi-layered Perceptron

Amount of hidden neurons to testing rate

Epoch: 10,000

Learning Rate: 0.2

Number of training data: 2288

Number of testing data: 572

Number of input neurons: 900

Number of output neurons: 52

Hidden neurons		Test 1	Test 2	Test 3
20	Cost Function:	2.89	4.07	3.68
	Time(miliseconds):	2341	2683	2745
	Accuracy:	4%	3%	4%
30	Cost Function:	0.66	0.55	2.98
	Time(miliseconds):	4071	4762	4276

	Accuracy:	4%	6%	4%
100	Cost Function:	0.23	0.37	0.26
	Time(miliseconds):	6645	6850	6812
	Accuracy:	5%	4%	5%
400	Cost Function:	0.07	0.08	0.07
	Time(miliseconds):	8621	9568	10735
	Accuracy:	8%	8%	6%
700	Cost Function:	0.12	0.08	0.06
	Time(miliseconds):	37568	35610	37501
	Accuracy:	8%	8%	7%
1000	Cost Function:	1.56	1.37	1.13
	Time(miliseconds):	68715	64231	67828
	Accuracy:	3%	4%	3%
1200	Cost Function:	2.42	2.23	3.43
	Time(miliseconds):	102473	127839	110937
	Accuracy:	3%	4%	4%

Multi-Layered Perceptron

Effect of learning to testing result

Epoch: 100,000

Hidden neurons: 300

Number of training data: 2288

Number of testing data: 572

Number of input neurons: 900

Number of output neurons: 52

Learning Rate		Test 1	Test 2	Test 3
0.02	Cost Function:	0.86	0.76	0.83
	Accuracy:	50.44%	51.34%	49.78%
0.04	Cost Function:	0.36	0.45	0.29
	Accuracy:	50.76%	49.24%	49.54%
0.06	Cost Function:	0.36	0.28	0.34
	Accuracy:	51.92%	51.57%	52.85%
0.08	Cost Function:	0.24	0.31	0.34
	Accuracy:	53.12%	54.06%	54.35%
0.1	Cost Function:	0.25	0.32	0.28
	Accuracy:	53.23%	53.4%	54.16%
0.2	Cost Function:	0.14	0.23	0.15

	Accuracy:	49.36%	48.51%	47.95%
0.3	Cost Function:	0.12	0.15	0.25
	Accuracy:	43.35%	43.7%	42.04%

Radial Basis Function

Training epoch to testing time and rate

Proportion of centres: 0.4

Learning Rate: 0.06

Number of training data: 2288

Number of testing data: 572

Number of input neurons: 900

Number of output neurons: 52

Epoch		Test1	Test2	Test3
10,000	Time to train (milliseconds)	40189	41927	39182
	Accuracy (%)	60.48%	59.37%	62.92%
50,000	Time to train (milliseconds)	228739	208173	210335
	Accuracy (%)	58.91%	57.51%	58.74%
100,000	Time to train (milliseconds)	459839	446218	436903
	Accuracy (%)	57.36%	59.68%	61.37%
500,000	Time to train (milliseconds)	2273817	2137892	2381729
	Accuracy (%)	58.01%	60.37%	63.81%
1,000,000	Time to train (milliseconds)	4678211	4519028	4637280
	Accuracy (%)	57.49%	62.78%	60.37%

Multi-layered Perceptron

Training epoch to testing time and rate

Hidden neurons: 400

Learning Rate: 0.06

Number of training data: 2288

Number of testing data: 572

Number of input neurons: 900

Number of output neurons: 52

Epoch		Test 1	Test 2	Test 3
10,000	Time (Milliseconds)	56824	54627	57246
	Accuracy (%)	8.6%	9.5%	8.2%
	Cost Function	0.29	0.18	0.31
50,000	Time (Milliseconds)	322786	365271	325617

	Accuracy (%)	22.7%	20.85%	23.50%
	Cost Function	0.15	0.18	0.23
100,000	Time (Milliseconds)	649839	657026	617033
	Accuracy (%)	52.79%	54.82%	53.27%
	Cost Function	0.09	0.10	0.08
500,000	Time (Milliseconds)	3859283	3627301	3700936
	Accuracy (%)	69%	72%	70%
	Cost Function	0.064	0.062	0.063
1,000,000	Time (Milliseconds)	6396372	6983011	7043925
	Accuracy (%)	90.54%	92.39%	93.72%
	Cost Function	0.05	0.04	0.04