CHINESE CHARACTER RECOGNITION

USING

NEURAL NETWORK

LOW POH TIAN

WEK 98264

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

UNIVERSITY OF MALAYA

SESSION 2002/2003

CONTENTS

Abstract

This project explores to the application of neural networks to the problem of identifying handwritten Chinese Characters in an automated manner. In particular, a backpropagation net is trained on a 5 Chinese Character fonts. The scope originally was to recognize five characters; it could easily be trained to recognize Chinese fonts.

*In process of training data, an original set of chine font was first generated. From this set, 4 other set were derived with different rotation angles. This provide a total of 20 images for the training data. In this system, each character is captured in a 320*240 pixel, black and white BITMAP file. During image processing a Bitmap file is broken up into a 100 components, each being a 32*24 pixel in dimension. If the "on" pixels in this region is more than 10% of the total pixels in the region, the component is set to "1" otherwise, it is set to "0".*

According to the implementation of this system, the input layer consists of 100 nodes and the output layer 5 nodes, each representing a character. Experiment has been carried out to determine the hidden layer size before it is set to 25 nodes. The training parameters of the network i.e. the learning rate and momentum has both fixed to 0.6.The extracted image data is then feed into the network, the weights of which are modified by error-backpropagation algorithm. The network has successfully been train to the error tolerance of 0.0001. As a result, it is able to recognize all the characters with which it is trained. However, other free-form (handwritten) cannot be recognize due to the limited training data.

CHAPTER 1

INTRODUCTION
CHINESE CHARACTER RECOGNITION SYSTEM.

CHAPTER 1: INTRODUCTION CHINESE HARACTER RECOGNITION SYSTEM.

1.1 Objective

The aim of this thesis is to simulate a neural network for character recognition. Initially, the target is to recognize at least 5 characters. The rationale being if it could recognizes at least 5 characters; it could easily be trained to recognize other characters as well. However, the network has been trained to recognize 20 Chinese characters.

To accomplish this mission, a multi-layer back propagation neural network would be simulated. A method need to be found to read and decode the image files which need to be preprocessed into a format which could be used as input to the neural network. Besides that, quite a number of image files need to be prepared to train and test the network.

1.2 CHINESE CHARACTER RECOGNITION

Recognition of Chinese characters is a challenging problem that has been approached using a variety of techniques. Two of these are feature extraction and structural analysis. To date, though, no dominant method has emerged and the recognition rates being obtained are still far behind what has been achieved for texts in the Latin alphabet. The disparity in recognition rates is not due to lack of effort; the issues faced in each task are strikingly different.

The number of fonts in common use in modern Chinese texts is small, probably as a result of the large number of characters required. There are two main character sets in use today: simplified and traditional.

Optical character recognition for Chinese texts poses new and interesting pattern recognition problems when compared to recognition of Latin or other Western scripts. The size of the corpus is daunting: Chinese fonts contain several thousand characters, including the few hundred English letters and symbols. Since Chinese dose not use an alphabet, one cannot employ the use of a dictionary to improve accuracy.

From a pattern recognition standpoint, Chinese character are widely varying complexity, consisting of one to thirty or more distinct strokes. Different between individual characters can quite small. In contrast, the differences between fonts can be significant. Both of these factors together are a potential problem for some methods of omni font recognition techniques.

Handwritten or printed character?

Difference pattern of character require difference method learning and training for the neural network. For the example, a hybrid neural network model is apply in handwritten word recognition. For printed Chinese Characters, we can apply an advanced system of classification using probabilistic neural network. Have, there are so many type of neural network can use to pursue automation recognition in both handwritten and printed characters.

In this project, we just focus to the recognition of Chinese handwritten Characters.

About Chinese character

Most recognition system have deait with only a limited set of fonts. The number of fonts in common use in modern Chinese texts is small, probably as a result of the large number of characters required. There is two main character sets in used today: simplified and traditional. The traditional set is larger and more elaborate, having evolved over hundreds of years. It is a complex system of pictograms, ideograms, phonograms, and phonon-ideograms, generating about 12 000 entries in a relatively complete dictionary. The number of stroke it contains can measure the complexity of a single character. In the Latin alphabet, characters contain from one to four strokes, while typical Chinese character contain anywhere from one to 36 strokes.

From 1956 to 1964, the People' s Republic of China introduced over 2000 simplified characters, The effort was undertaken in order to reduce the number of strokes necessary to form common characters. A standard dictionary of the simplified character set contains about 7 000 of the characters in general use. The People' s republic of china created the simplified standard for its own use. The traditional character set is still the norm in Taiwan, Hong Kong, Macau, and in overseas Chinese publications. Also in common use are four main font styles: *songti, fongsongti, kaiti, and heiti.*

1.2 Aims, Scope, relevance Significance and motivation of the project

Aims

Developing the Chinese character recognition can contributed to the automation process. With such a system, data need only be scanned into the system and recognized automatically. This recognition process must time efficient.

The aim of this project is to simulate a neural network for character recognition. Hopefully, this project can contribute some effort to the automation recognition of Chinese character in the country. This system can provide a good service to the Chinese education sector at primary, secondary and also higher level and also Chinese publication Sector.

Scope

The system that developing for this project just to simulate a neural network. For this project, that is impossible to build a Chinese OCR engine that included the number of recognition classes in so short period. (With about 7000 characters in general use, it is necessary to build a system that can efficiently and reliably recognize far more classes than are required for Latin text.) Basically, the scope is to build an engine that allows the system to recognize 1-5 character of Chinese. The rationale being if it could recognize one character, it could easily be trained to recognize other character as well.

The recognition system which develop for this project can only apply to the Simplified Chinese character sets.

Relevance

One of the neural network' s applications is character recognition. In this project, the system that developed is one type of intelligent system that can keep on learning. This mean that the capability of the system can be improve through training. This is the more wonderful technology in Artifilicial intelligent. You can imagine that the computer like a child can learn something new from the training!!!! That is great!!! This project is relevant with the technology image processing that provides learning ability to the system. Recognition is one of the branch for the image processing.

Significance and the future of Chinese character using neural network.

This project can be extending to include more Chinese Character class. Nowadays, the world of technology is requires that the computer act as human brain. One of the advanced features of intelligent system is the ability of reasoning, learning, training to updates the focused problem-solving skill.

Huge amount of Chinese character are need to recognize for the business, educational and also research purpose. Using manpower to doing the recognition

works are time consuming and also inefficient. So, a system that can act human-like and increasing the capability time by time is highly needed.

The Chinese character recognition can develop as a business that is expected will return great profit. The population that using Chinese language as main language is the attractive property to convert the character recognition technology to a profitable business.

1.3 MOTIVATION

When I start to run this project, sincerely, I know no thing about character recognition, neural network and also image processing. Maybe I am doing something that out of my ability. But, I just want to try something new that related with my majoring. Through the study about the project, I have learned how to appreciate the technology. As a student computer science that majoring Artificial Intelligent, I would like to take risk to develop something that maybe difficult to achieve the good results but can get me a opportunity to appreciate the wonderful of AI technology.

1.4 PROJECT PROBLEM DEFINITION

Problems

Character segmentation is the process of isolating the individual characters in a handwritten image block. Ideally, handwritten word recognition system would use a sequential process in which character segmentation was performed prior to character recognition. Unfortunately, as is often true in image pattern recognition, it is difficult to segment without recognizing and it is difficult to recognize without segmenting.

Differences between individual characters can be quite small, as can seen in Figure 1. This features made the process of recognition more difficult because the recognition system maybe cannot recognize the actual character in case of similarity of the characters.

Figure 1.1 Examples of visually similar character in the Chinese character font.

1.5 PROJECT SCHEDULE

A Gantt chart is an easy way to schedule tasks. It is essentially a chart on which bars represent each task or activity. The length of each bar represents the relative length of the task. Figure 1.1 below is an example of Gantt chart where time is indicated on the horizontal dimension and description of activities makes up the vertical dimension. This is the project planning for the system.

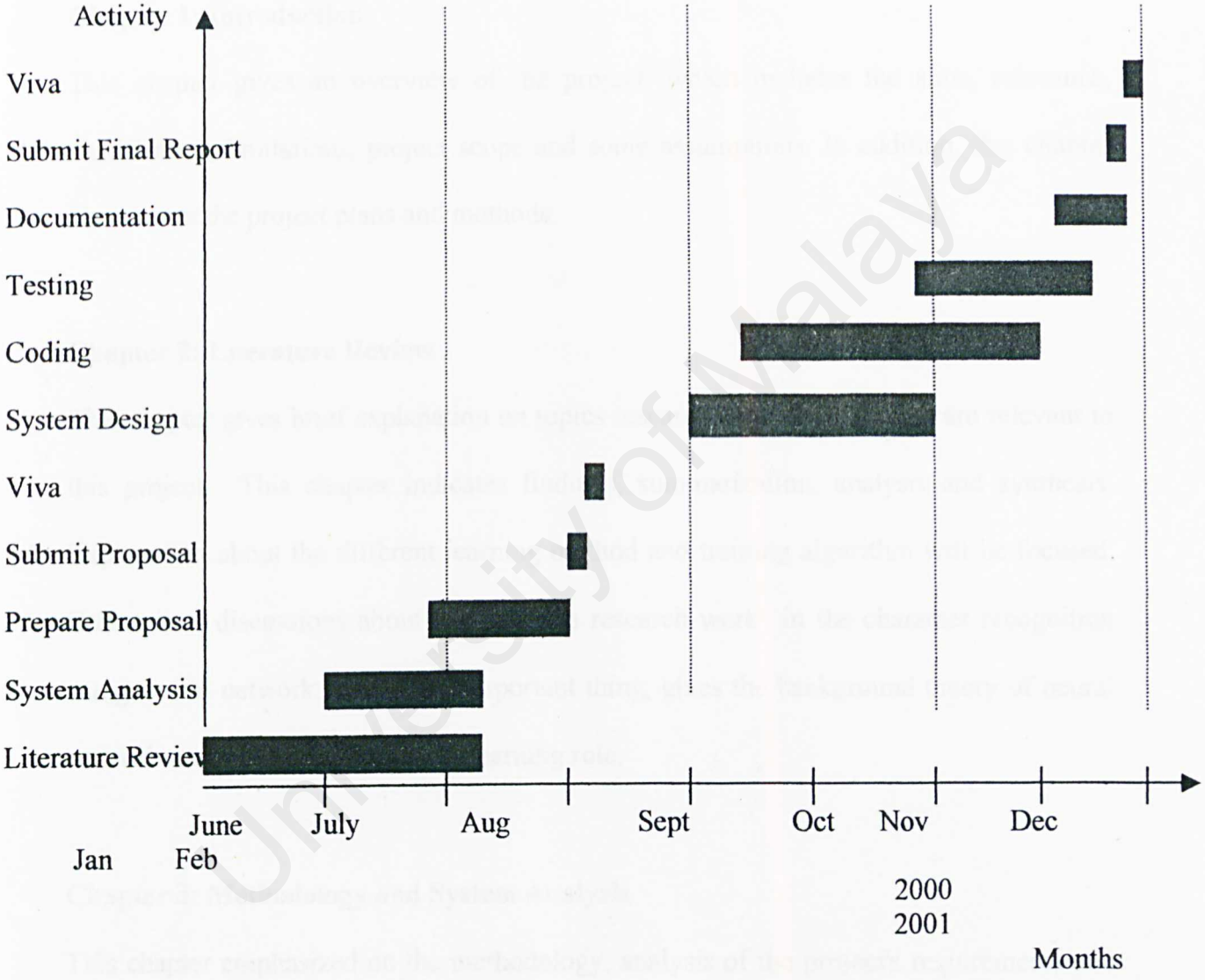


Figure 1.2: Using a two-dimensional Gantt chart for planning activities [7]

1.6 REPORT LAYOUT

The purpose of this project layout is to give an overall overview of the major contents, which will be included and involved during the development of this project. Below is the report layout:

Chapter 1: Introduction

This chapter gives an overview of the project, which includes the aims, relevance, significance, limitations, project scope and some assumptions. In addition, this chapter also covers the project plans and methods.

Chapter 2: Literature Review

This chapter gives brief explanation on topics researched and studied that are relevant to this project. This chapter indicates findings, summarization, analysis and synthesis. Explanation about the different learning method and training algorithm will be focused. Otherwise, discussions about the previous research work in the character recognition using neural network. The more important thing, gives the background theory of neural network and the back propagation learning rule.

Chapter 3: Methodology and System Analysis

This chapter emphasized on the methodology, analysis of the project's requirements and development tools. It explains how the requirements for this project were acquired and the analysis of the results. Beside that, it also analyst the development tools available to choose the best tools/software to develop the system..

Chapter 4: System Design

This chapter explains the conceptual and technical design processes of the system. We also look at the design strategy. Explains how the character recognition system is being designed. The design is also in the form of a diagram.

Chapter 5: System Implementation and Testing

This chapter gives a description of the environment in which the system are developed and implemented after completion. It also includes the verifications and validations of the system to make sure the errors are in the minimum level.

Chapter 6: System Evaluation and Conclusion

This chapter will evaluate the system in terms of strengths and limitations together with suggestions for further enhancements for the system. The problems encountered during the development of the system will also be illustrated here. This chapter ends with a conclusion of the whole project.

Chapter 2

Artificial Neural Networks

Chapter 2 Artificial Neural Networks

Artificial Neural Network is a system loosely modeled on the human brain. The field goes by many names, such as connectionism; parallel distributed processing, neuron-computing, natural intelligent systems, machine learning algorithms, and artificial neural networks. It is an attempt to simulate within specialized hardware or sophisticated software, the multiple layers of simple processing elements called neurons. Each neuron is linked to certain of its neighbors with varying coefficients of connectivity that represent the strengths of these connections. Learning is accomplished by adjusting these strengths to cause the overall network to output appropriate results.

2.1 The Analogy to the Brain

The most basic components of neural networks are modeled after the structure of the brain. Some neural network structures are not closely to the brain and some does not have a biological counterpart in the brain. However, neural networks have a strong similarity to the biological brain and therefore a great deal of the terminology is borrowed from neuroscience.

2.1.1 The Biological Neuron

The most basic element of the human brain is a specific type of cell, which provides us with the abilities to remember, think, and apply previous experiences to our every action. These cells are known as neurons; each of these neurons can connect with up to 200000 other neurons. The power of the brain comes from the numbers of these basic components and the multiple connections between them.

All natural neurons have four basic components, which are dendrites, soma, axon, and synapses. Basically, a biological neuron receives inputs from other sources, combines them in some way, performs a generally nonlinear operation on the result, and then output the final result. The figure below shows a simplified biological neuron and the relationship of its four components.

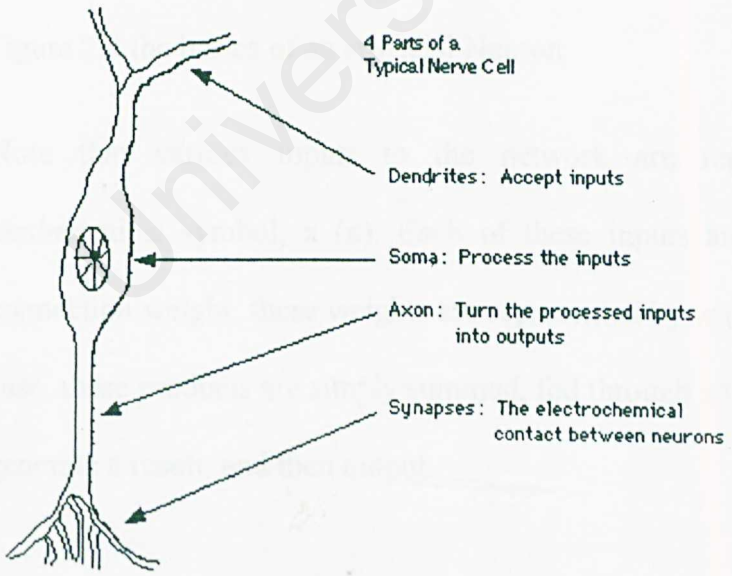


Figure 2.1 a biological neuron

2.1.2 The Artificial Neuron

The basic unit of neural networks, the artificial neurons, simulates the four basic functions of natural neurons. Artificial neurons are much simpler than the biological neuron; the figure below shows the basics of an artificial neuron.

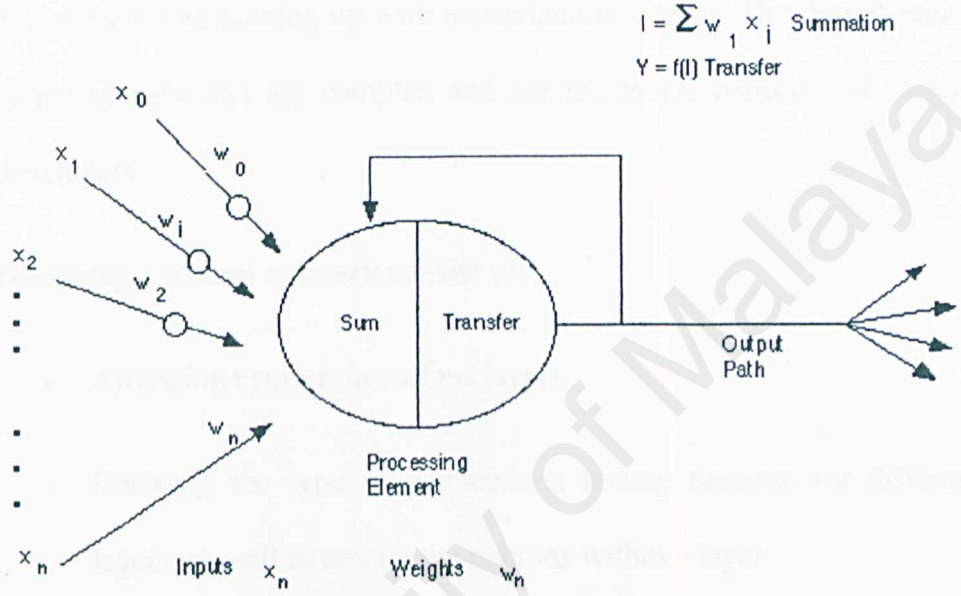


Figure 2.2 the basics of an artificial Neuron

Note that various inputs to the network are represented by the mathematical symbol, x (n). Each of these inputs are multiplied by a connection weight, these weights are represented by w (n). In the simplest case, these products are simply summed, fed through a transfer function to generate a result, and then output.

Even though all artificial neural networks are constructed from this basic building block the fundamentals may vary in these building blocks and there are differences.

2.2 Design

The developer must go through a period of trial and error in the design decisions before coming up with a satisfactory design. The design issues in neural networks are complex and are the major concerns of system developers.

Designing a neural network consist of:

- Arranging neurons in various layers.
- Deciding the type of connections among neurons for different layers, as well as among the neurons within a layer.
- Deciding the way a neuron receives input and produces output.
- Determining the strength of connection within the network by allowing the network learns the appropriate values of connection weights by using a training data set.

The process of designing a neural network is an iterative process; the figure below describes its basic steps.

Layers

Biologically, neural networks are constructed in a three dimensional way from microscopic components. These neurons seem capable of nearly unrestricted interconnections. This is not true in any man-made network. Artificial neural networks are the simple clustering of the primitive artificial neurons. This clustering occurs by creating layers, which are then connected to one another. How these layers connect may also vary. Basically, all artificial neural networks have a similar structure of topology. Some of the neurons interface the real world to receive its inputs and other neurons provide the real world with the network' s outputs. All the rest of the neurons are hidden form view.

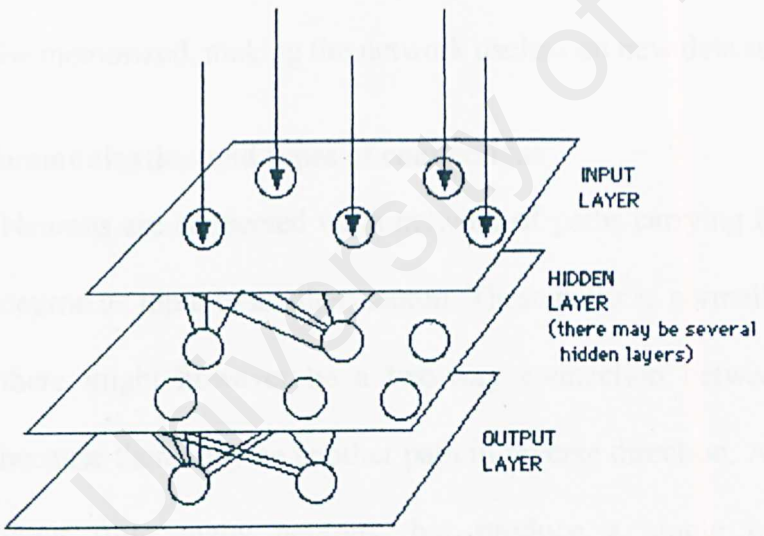


Figure 2.3 A neural network layer

As the figure above shows, the neurons are grouped into layers .The input layer consist of neurons that receive input form the external environment. The output layer consists of neurons that communicate the output of the

system to the user or external environment. There are usually a number of hidden layers between these two layers; the figure above shows a simple structure with only one hidden layer.

When the input layer receives the input its neurons produce output, which becomes input to the other layers of the system. The process continues until a certain condition is satisfied or until the output layer is invoked and fires their output to the external environment.

To determine the number of hidden neurons the network should have to perform its best, one are often left out to the method trial and error. If you increase the hidden number of neurons too much you will get an over fit, that is the net will have problem to generalize. The training set of data will be memorized, making the network useless on new data sets.

2.2.2 Communication and types of connections

Neurons are connected via a network of paths carrying the output of one neuron as input to another neuron. These paths is normally unidirectional, there might however be a two-way connection between two neurons, because there may be another path in reverse direction. A neuron receives input from many neurons, but produce a single output, which is communicated to other neurons.

The neuron in a layer may communicate with each other, or they may not have any connections. The neurons of one layer are always connected to the neurons of at least another layer.

2.2.2.1 Inter-layer connections

There are different types of connections used between layers; these connections between layers are called inter-layer connections.

- **Fully connected**

  Each neuron on the first layer is connected to every neuron on the second layer.

- **Partially connected**

  A neuron of the first layer does not have to be connected to all neurons on the second layer.

- **Feed forward**

  The neurons on the first layer send their output to the neurons on the second layer, but they do not receive any input back form the neurons on the second layer.

- **Bi-directional**

  There is another set of connections carrying the output of the neurons of the second layer into the neurons of the first layer.

Feed forward and bi-directional connections could be fully- or partially connected.

- **Hierarchical**

  If a neural network has a hierarchical structure, the neurons of a

lower layer may only communicate with neurons on the next level of layer.

- **Resonance**

  The layers have bi-directional connections, and they can continue sending messages across the connections a number of times until a certain condition is achieved.

2.2.2.2 Intra-layer connections

In more complex structures the neurons communicate among themselves within a layer, this is known as intra-layer connections. There are two types of intra-layer connections.

- **Recurrent**

  The neurons within a layer are fully- or partially connected to one another. After these neurons receive input form another layer, they communicate their outputs with one another a number of times before they are allowed to send their outputs to another layer. Generally some conditions among the neurons of the layer should be achieved before they communicate their outputs to another layer.

- **On-center/off surround**

A neuron within a layer has excitatory connections to itself and its immediate neighbors, and has inhibitory connections to other neurons. One can imagine this type of connection as a competitive gang of neurons. Each gang excites itself and its gang members and inhibits all members of other gangs. After a few rounds of signal interchange, the neurons with an active output value will win, and is allowed to update its and its gang member' s weights. (There are two types of connections between two neurons, excitatory or inhibitory. In the excitatory connection, the output of one neuron increases the action potential of the neuron to which it is connected. When the connection type between two neurons is inhibitory, then the output of the neuron sending a message would reduce the activity or action potential of the receiving neuron. One causes the summing mechanism of the next neuron to add while the other causes it to subtract. One excites while the other inhibits.)

2.3 Learning

The brain basically learns from experience. Neural networks are sometimes called machine-learning algorithms, because changing of its connection weights (training) causes the network to learn the solution to a problem. The strength of connection between the neurons is stored as a

weight-value for the specific connection. The system learns new knowledge by adjusting these connection weights.

The learning ability of a neural network is determined by its architecture and by the algorithmic method chosen for training.

The training method usually consists of one of three schemes:

1. **Unsupervised Learning**

   The hidden neurons must find a way to organize themselves without help from the outside. In this approach, no sample outputs are provided to the network against which it can measure its predictive performance for a given vector of inputs. This is learning by doing.

2. **Reinforcement learning**

   This method works on reinforcement from the outside. The connections among the neurons in the hidden layer are randomly arranged, then reshuffled as the network is told how close it is to solving the problem. Reinforcement learning is also called supervised learning, because it requires a teacher. The teacher may be a training set of data or an observer who grades the performance of the network results.

Both unsupervised and reinforcement suffers from relative slowness and inefficiency relying on a random shuffling to find the proper connection weights.

3. **Back propagation**

This method is proven highly successful in training of multilayered neural nets. The network is not just given reinforcement for how it is doing on a task. Information about errors is also filtered back through the system and is used to adjust the connections between the layers, thus improving performance. A form of supervised learning. Back propagation, which uses the data to adjust the network' s weights and thresholds so as to minimize the error in its predications on the training set. If the network is relates the input variables to the output variables, and can subsequently be used to make predictions where the output is not known.

2.3.1 Off-line or On-line

One can categorize the learning methods into yet another group, off-line or on-line. When the system uses input data to change its weights to learn the domain knowledge, the system could be in training mode or learning mode. When the system is being used as a decision aid to make recommendations, it is in the operation mode, this is also sometimes called recall.

- **Off-line**

  In the off-line learning methods, once the systems enters into the operation mode, its weights are fixed and do not change any more. Most of the networks are of the off-line learning type.

- **On-line**

  In on-line or real time learning, when the system is in operating mode (recall), it continues to learn while being used as a decision tool. This type of learning has a more complex design structure.

2.3.2 Learning laws

There are a variety of learning laws that are in common use. These laws are mathematical algorithms used to update the connection weights. Most of these laws are some sorts of variation of the best-known and oldest learning law, Herb' s Rule. Man' s understanding of how neural processing actually works is very limited. Learning is certainly more complex than the simplification represented by the learning laws currently developed. Research into different learning functions continues as new ideas routinely show up in trade publications etc. A few of the major laws are given as an example below.

- **Hebb' s Rule**

The first and the best known learning rule were introduced by Donald Hebb. The description appeared in his book *The organization of Behavior* in 1949. This basic rule is: If a neuron receives an input from another neuron, and if

both are highly active (mathematically have the same sign), the weight between the neurons should be strengthened.

- **Hopfield Law**

This law is similar to Hebb' s Rule with the exception that it specifies the magnitude of the strengthening or weakening. It states, "if the desired output and the input are both active or both inactive, increment the connection weight by the learning rate, otherwise decrement the weight by the learning rate." (Most learning functions have some provision for a learning rate, or a learning constant. Usually this term is positive and between zero and one.)

- **The Delta Rule**

The Delta Rule is a further variation of Hebb' s Rule, and it is one of the most commonly used. This rule is based on the idea of continuously modifying the strengths of the input connections to reduce the difference (the delta) between the desired output value and the actual output of a neuron. This rule changes the connection weights in the way that minimizes the mean squared error of the network. The error is back propagated into previous layers one layer at a time. The process of back-propagating the network errors continues until the first layer is reached. The network type called Feed forward, Back-propagation derives its name from this method of computing the error term.

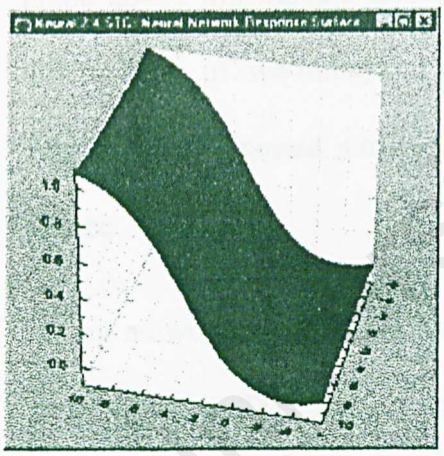
This rule is also referred to as the Windrow-Hoff Learning Rule and the Least Mean Square Learning Rule.

- **Kohonen' s Learning Law**

This procedure, developed by Teuvo Kohonen, was inspired by learning in biological systems. In this procedure, the neurons compete for the opportunity to learn, or to update their weights. The processing neuron with the largest output is declared the winner and has the capability of inhibiting its competitors as well as exciting its neighbors. Only the winner is permitted output, and only the winner plus its neighbors are allowed to update their connection weights. The Kohonen rule does not require desired output. Therefore it is implemented in the unsupervised methods of learning. Kohonen has used this rule combined with the on-center/off-surround intra-layer connection (discussed earlier under 2.2.2.2) to create the self-organizing neural network, which has an unsupervised learning method.

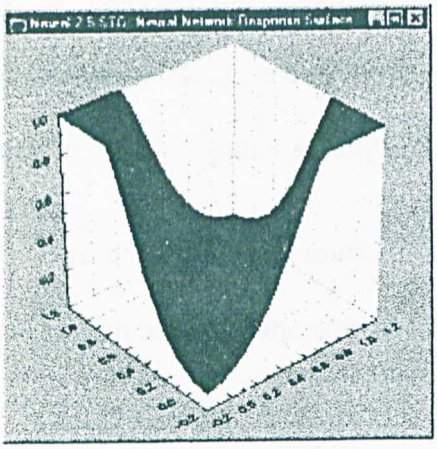
2.4 Insights into MLP training

More key insights into MLP behavior and training can be gained by considering the type of functions they model. Recall that the activation level of a unit is the weighted sum of the inputs, plus a threshold value. This implies that the activation level is actually a simple linear function of the inputs. The activation is then passed through a sigmoid (S-shaped) curve.



The combination of the multi-dimensional linear function and the one-dimensional sigmoid function gives the characteristic "sigmoid cliff" response of a first hidden layer MLP unit (the graph above illustrates the shape plotted across two inputs. An MLP unit with more inputs has a higher-dimensional version of this functional shape). Altering the weights and thresholds alters this response surface. In particular, both the orientation of the surface, and the steepness of the

sloped section, can be altered. A steep slope corresponds to large weight values: doubling all weight values gives the same orientation but a different slope.



A multi-layered network combines a number of these response surfaces together, through repeated linear combination and non-linear activation functions. The graph above illustrates a typical response surface for a network with only one hidden layer, of two units, and a single output unit, on the classic XOR problem. Two separate sigmoid surfaces have been combined into a single "U-shaped" surface.

During network training, the weights and thresholds are first initialized to small, random values. This implies that the units' response surfaces are each aligned randomly with low slope: they are effectively "uncommitted." As training progresses, the units' response surfaces are rotated and shifted into appropriate positions, and the magnitudes of the weights grow as they commit to modeling particular parts of the target response surface.

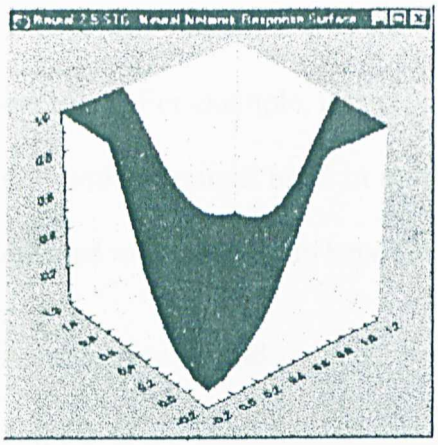
In a classification problem, an output unit's task is to output a strong signal if a case belongs to its class, and a weak signal if it does not. In other words, it is

attempting to model a function, which has magnitude one for parts of the pattern-space, which contain its cases, and magnitude zero for other parts. This is known as a *discriminant function* in pattern recognition problems. An "ideal" discriminant function could be said to have a plateau structure, where all points on the function are either at height zero or height one.

If there are no hidden units, then the output can only model a single "sigmoid-cliff" with areas to one side at low height and areas to the other high. There will always be a region in the middle (on the cliff) where the height is in-between, but as weight magnitudes are increased, this area shrinks.

A sigmoid-cliff like this is effectively a linear discriminant. Points to one side of the cliff are classified as belonging to the class, points to the other as not belonging to it. This implies that a network with no hidden layers can only classify linearly-separable problems (those where a line - or, more generally in higher dimensions, a hyper plane - can be drawn which separates the points in pattern space).

A network with a single hidden layer has a number of sigmoid-cliffs (one per hidden unit) represented in that hidden layer, and these are in turn combined into a plateau in the output layer. The plateau has a convex hull (i.e. there are no dents in it, and no holes inside it). Although the plateau is convex, it may extend to infinity in some directions (like an extended peninsular). Such a network is in practice capable of modeling adequately most real-world classification problems.



As shown above the plateau response surface developed by an MLP to solve the XOR problem: as can be seen, this neatly sections the space along a diagonal.

A network with two hidden layers has a number of plateaus combined together - the number of plateaus corresponds to the number of units in the second layer, and the number of sides on each plateau corresponds to the number of units in the first hidden layer. A little thought shows that you can represent any shape (including concavities and holes) using a sufficiently large number of such plateaus.

A consequence of these observations is that an MLP with two hidden layers is theoretically sufficient to model any problem (there is a more formal proof, the Kolmogorov Theorem). This does not necessarily imply that a network with more layers might not more conveniently or easily model a particular problem. In practice, however, most problems seem to yield to a single hidden layer, with two an occasional resort and three practically unknown.

A key question in classification is how to interpret points "on or near" the cliff. The standard practice is to adopt some confidence levels (the accept and reject

thresholds), which must be exceeded before the unit, is deemed to have "made a decision." For example, if accept/reject thresholds of 0.95/0.05 are used, an output unit with an output level in excess of 0.95 is deemed to be on, below 0.05 it is deemed to be off, and in between it is deemed to be "undecided."

A more subtle (and perhaps more useful) interpretation is to treat the network outputs as probabilities. In this case, the network gives more information than simply a decision: it tells us how sure (in a formal sense) it is of that decision. There are modifications to MLPs (supported by *ST Neural Networks*) which allow the neural network outputs to be interpreted as probabilities, which means that the network effectively learns to model the probability density function of the class. However, the probabilistic interpretation is only valid under certain assumptions about the distribution of the data (specifically, that it is drawn from the family of exponential distributions; see Bishop, 1995). Ultimately, a classification decision must still be made, but a probabilistic interpretation allows a more formal concept of "minimum cost" decision making to be evolved.

2.4 Other MLP Training Algorithms

Earlier in this section, we discussed how the *back propagation* algorithm performs gradient descent on the error surface. Speaking loosely, it calculates the direction of steepest descent on the surface, then jumps down the surface a distance proportional to the learning rate and the slope, picking up momentum as it maintains a consistent direction. As an analogy, it behaves like a blind-folded kangaroo hopping in the most obvious direction. Actually, the descent is calculated independently on the error surface for each training case, and in random order, but this is actually a good approximation to descent on the composite error surface. Other MLP training algorithms work differently, but all use a strategy designed to travel towards a minimum as quickly as possible.

More sophisticated techniques for non-linear function optimization have been in use for some time. *ST Neural Networks* includes two of these: *conjugate gradient descent* and *Livener-Marquardt* (see Bishop, 1995; Shepherd, 1997), which are very successful forms of two types of algorithm: line search and model-trust region approaches.

A line search algorithm works as follows: pick a sensible direction to move in the multi-dimensional landscape. Then project a line in that direction, locate the minimum along that line (it is relatively trivial to locate a minimum along a line, by using some form of bisection algorithm), and repeat. What is a "sensible direction" in this context? An obvious choice is the direction of steepest descent (the same direction which would be chosen by *back propagation*). Actually, this

intuitively-obvious choice proves to be rather poor. Having minimized along one direction, the next line of steepest descent may "spoil" the minimization along the initial direction (even on a simple surface like a parabola a large number of line searches may be necessary). A better approach is to select conjugate or "non-interfering" directions - hence *conjugate gradient descent* (Bishop, 1995).

The idea here is that, once the algorithm has minimized along a particular direction, the second derivative along that direction should be kept at zero. Conjugate directions are selected to maintain this zero second derivative on the assumption that the surface is parabolic (speaking roughly, a "nice smooth surface"). If this condition holds, *N* epochs are sufficient to reach a minimum. In reality, on a complex error surface the conjugacy deteriorates, but the algorithm still typically requires far less epochs than *back propagation*, and also converges to a better minimum (to settle down thoroughly, *back propagation* must be run with an extremely low learning rate).

A model-trust region approach works as follows: instead of following a search direction, assume the surface is a simple shape such that the minimum can be located (and jumped to) directly - if the assumption is true. Try the model out and see how good the suggested point is. The model typically assumes that the surface is a nice well-behaved shape (e.g. a parabola), which will be true if sufficiently close to a minimum. Elsewhere, the assumption may be grossly violated, and the model could choose wildly-inappropriate points to move to. The model can only be trusted within a region of the current point, and the size of this region is not known. Therefore, choose new points to test as a compromise between that

suggested by the model and that suggested by a standard gradient-descent jump. If the new point is good, move to it, and strengthen the role of the model in selecting a new point; if it is bad, don't move, and strengthen the role of the gradien descent step in selecting a new point (and make the step smaller). *Levenberg-Marquardt* uses a model which assumes that the underlying function is locally linear (and therefore has a parabolic error surface).

Levenberg-Marquardt is typically the fastest of the training algorithms supported in *ST Neural Networks*, although unfortunately it has some important limitations, specifically: it can only be used on single output networks, can only be used with the *sum squared error* function, and has memory requirements proportional to *W**2* (where *W* is the number of weights in the network; this makes it impractical for reasonably big networks). *Conjugate gradient descent* is nearly as good, and doesn't suffer from these restrictions.

Back propagation can still be useful, not least in providing a quick (if not overwhelmingly accurate) solution. It is also a good choice if the data set is very large, and contains a great deal of redundant data. *Back propagation*'s case-by-case error adjustment means that data redundancy does it no harm (for example, if you double the data set size by replicating every case, each epoch will take twice as long, but have the same effect as two of the old epochs, so there is no loss). In contrast, *Levenberg-Marquardt* and *SIZE=4* both perform calculations using the entire data set, so increasing the number of cases may significantly slow each epoch, but does not necessarily improve performance on that epoch (not if data is redundant; if data is sparse, then adding data will make each epoch better). *Back*

propagation may also be equally good if the data set is very small, for there is then insufficient information to make a highly fine-tuned solution appropriate (a more advanced algorithm may achieve a lower training error, but the verification error is unlikely to improve in the same way).

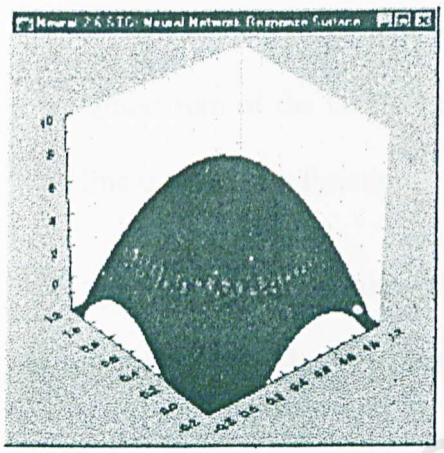
ST Neural Networks also includes two variations on *back propagation* (*quick propagation*, Fahlman, 1988, and *Delta-bar-Delta*, Jacobs, 1988) which are designed to deal with some of the limitations on this technique. In most cases, they are not significantly better than *back propagation*, and sometimes they are worse (relative performance is application-dependent). They also require more control parameters than any of the other algorithms, which make them more difficult to use, so they are not described in further detail in this section.

2.5 Other type of neural network

Radial Basis Function Networks

We have seen in the last section how an MLP models the response function using the composition of "sigmoid-cliff" functions - in the case of classification problems, this corresponds to dividing the pattern space up using hyper planes. The use of hyper planes to divide up space is a natural approach - intuitively appealing, and based on the fundamental simplicity of lines.

An equally appealing and intuitive approach is to divide up space using circles or (more generally) hyper spheres. A hyper sphere is characterized by its center and radius. More generally, just as an MLP unit responds (non-linearly) to the distance of points from the line of the "sigmoid-cliff", in a radial basis function network (Broomhead and Lowe, 1988; Moody and Darkin, 1989; Haykin, 1994) units respond (non-linearly) to the distance of points from the "center" represented by the radial unit. The response surface of a single radial unit is therefore a Gaussian (bell-shaped) function, peaked at the center, and descending outwards. Just as the steepness of the MLP's sigmoid curves can be altered, so can the slope of the radial unit's Gaussian. See the graph below.



MLP units are defined by their weights and threshold, which together give the equation of the defining line, and the rate of fall-off of the function from that line. Before application of the sigmoid activation function, the activation level of the unit is a hyper plane, and in *ST Neural Networks* these units are therefore referred to as *linear units* (although the activation function is typically non-linear). In contrast, a *radial unit* is defined by its center point and a "radius." A point in *N* dimensional space is defined using *N* numbers, which exactly corresponds to the number of weights in a linear unit, so the center of a radial unit is stored in *ST Neural Networks* as "weights." The radius (or

deviation) value is stored as the "threshold." It is worth emphasizing that the "weights" and "thresholds" in a radial unit are actually entirely different to those in a linear unit, and the terminology is dangerous if you don't remember this: Radial weights really form a point, and a radial threshold is really a deviation.

A *radial basis function* network (RBF), therefore, has a hidden layer of radial units, each actually modeling a Gaussian response surface. Since these functions are non-linear, it is not actually necessary to have more than one hidden layer to model any shape of function: sufficient radial units will always be enough to model any function. The remaining question is how to combine the hidden radial unit outputs into the network outputs? It turns out to be quite sufficient to use a linear combination of these outputs (i.e. a weighted sum of the Gaussians). The RBF has an output layer containing linear units with linear activation function.

RBF networks have a number of advantages over MLPs. First, as previously stated, they can model any non-linear function using a single hidden layer, which removes some design-decisions about numbers of layers. Second, the simple linear transformation in the output layer can be optimized fully using traditional linear modeling techniques, which are fast and do not suffer from problems such as local minimum which plague MLP training techniques. RBF networks can therefore be trained extremely quickly (i.e. orders of magnitude faster than MLPs).

On the other hand, before linear optimization can be applied to the output layer of an RBF network, the number of radial units must be decided, and then their centers and deviations must be set. Although much faster than MLP training, the algorithms to do this

are no less prone to discovering sub-optimal combinations. In compensation, *ST Neural Network ' s Automatic Network Designer* can perform the inevitable experimental stage for you.

Other features which distinguish RBF performance from MLPs are due to the differing approaches to modeling space, with RBFs "clumpy" and MLPs "planey."

RBFs are also more sensitive to the "curse of dimensionality," and have greater difficulties if the number of input units is large: this problem is discussed further in a later section.

As mentioned earlier, training of RBFs takes place in distinct stages. First, the centers and deviations of the radial units must be set; then the linear output layer is optimized.

Centers should be assigned to reflect the natural clustering of the data. The two most common methods are:

Sub-sampling. Randomly-chosen training points are copied to the radial units. Since they are randomly selected, they will "represent" the distribution of the training data in a statistical sense. However, if the number of radial units is not large, the radial units may actually be a poor representation (Haykin, 1994).

K-Means algorithm. This algorithm (Bishop, 1995) tries to select an optimal set of points which are placed at the centroids of clusters of training data.

Explicit. Choose the deviation yourself.

Isotropic. The deviation (same for all units) is selected heuristically to reflect the number of centers and the volume of space they occupy (Haykin, 1994).

K-Nearest Neighbor. Each unit's deviation is individually set to the mean distance to its *K* nearest neighbors (Bishop, 1995). Hence, deviations are smaller in tightly packed areas of space, preserving detail, and higher in sparse areas of space (interpolating where necessary).

Radial basis functions can also be hybridized in a number of ways. The output layer can be altered to contain non-linear activation functions, in which case any of the multiplayer perceptron training algorithms such as *back propagation* can be used to train it. It is also possible to train the radial layer (the hidden layer) using the Kohonen network training algorithm, which is another method of assigning centers to reflect the spread of data.

Probabilistic Neural Networks

In an earlier section, we briefly mentioned that, in the context of classification problems, a useful interpretation of network outputs was as estimates of probability of class membership, in which case the network was actually learning to estimate a probability density function. A similar useful interpretation can be made in regression problems, if the output of the network is regarded as the expected value of the model at a given point in input-space. This expected value is related to the joint probability density function of the output and inputs.

Estimating probability density functions (p.d.f.) from data has a long statistical history (Parzen, 1962), and in this context fits into the area of Bayesian statistics. Conventional statistics can, given a known model, inform us what the chances of certain outcomes are (e.g. we know that a unbiased die has a 1/6th chance of

coming up with a six). Bayesian statistics turns this situation on its head, by estimating the validity of a model given certain data. More generally, Bayesian statistics can estimate the probability density of model parameters given the available data. To minimize error, the model is then selected whose parameters maximize this p.d.f.

In the context of a classification problem, if we can construct estimates of the p.d.f.s of the possible classes, we can compare the probabilities of the various classes, and select the most-probable. This is effectively what we ask a neural network to do when it learns a classification problem - the network attempts to learn (an approximation to) the p.d.f.

A more traditional approach is to construct an estimate of the p.d.f. from the data. The most traditional technique is to assume a certain form for the p.d.f. (typically, that it is a normal distribution), and then to estimate the model parameters. The normal distribution is commonly used as the model parameters (mean and standard deviation) can be estimated using analytical techniques. The problem is that the assumption of normality is often not justified.

An alternative approach to p.d.f. estimation is *kernel-based approximation* (see Parzen, 1962; Speckt, 1990; Speckt, 1991; Bishop, 1995; Patterson, 1996). We can reason loosely that the presence of particular cases indicates some probability density at that point: a cluster of cases close together indicates an area of high probability density. Close to a case, we can have high confidence in some probability density, with a lesser and diminishing level as we move away. In

kernel-based estimation, simple functions are located at each available case, and added together to estimate the overall p.d.f. Typically, the kernel functions are each Gaussians (bell-shapes). If sufficient training points are available, this will indeed yield an arbitrarily good approximation to the true p.d.f.

This kernel-based approach to p.d.f. approximation is very similar to radial basis function networks, and motivates the probabilistic neural network (PNN) and generalize regression neural network (GRNN), both devised by Speckt (1990 and 1991). PNNs are designed for classification tasks, and GRNNs for regression. These two types of network are really kernel-based approximation methods cast in the form of neural networks.

In the PNN, there are at least three layers: input, radial, and output layers. The radial units are copied directly from the training data, one per case. Each models a Gaussian function centered at the training case. There is one output unit per class. Each is connected to all the radial units belonging to its class, with zero connections from all other radial units. Hence, the output units simply add up the responses of the units belonging to their own class. The outputs are each proportional to the kernel-based estimates of the p.d.f.s of the various classes, and by normalizing these to sum to 1.0 estimates of class probability are produced.

The basic PNN can be modified in two ways.

First, the basic approach assumes that the proportional representation of classes in the training data matches the actual representation in the population being modeled (the so-called prior probabilities). For example, in a disease-diagnosis

network, if 2% of the populations have the disease, then 2% of the training cases should be positives. If the prior probability is different from the level of representation in the training cases, then the network's estimate will be invalid. To compensate for this, prior probabilities can be given (if known), and the class weightings are adjusted to compensate.

Second, any network making estimates based on a noisy function will inevitably produce some misclassifications (there may be disease victims whose tests come out normal, for example). However, some forms of misclassification may be regarded as "more expensive mistakes" than others (for example, diagnosing somebody healthy as having a disease, which simply leads to exploratory surgery may be inconvenient but not life-threatening; whereas failing to spot somebody who is suffering from disease may lead to premature death). In such cases, the raw probabilities generated by the network need to be weighted by loss factors, which reflect the costs of misclassification. In *ST Neural Networks*, a fourth layer may be specified in PNNs which includes a loss matrix. This is multiplied by the probability estimates in the third layer, and the class with lowest estimated cost is selected. (Loss matrices may also be attached to other types of classification network).

The only control factor which needs to be selected for probabilistic neural network training is the smoothing factor (i.e. the radial deviation of the Gaussian functions). As with RBF networks, this factor needs to be selected to cause "a reasonable amount of overlap" - too small deviations cause a very spiky approximation which cannot generalize, too large deviations smooth out detail.

An appropriate figure is easily chosen by experiment, by selecting a number which produces a low verification error, and fortunately PNNs are not too sensitive to the precise choice of smoothing factor.

The greatest advantages of PNNs are the fact that the output is probabilistic (which makes interpretation of output easy), and the training speed. Training a PNN actually consists mostly of copying training cases into the network, and so is as close to instantaneous as can be expected.

The greatest disadvantage is network size: a PNN network actually contains the entire set of training cases, and is therefore space-consuming and slow to execute.

PNNs are particularly useful for prototyping experiments (for example, when deciding which input parameters to use), as the short training time allows a great number of tests to be conducted in a short period of time. *ST Neural Networks* itself uses PNNs in its *Neuro-Genetic Input Selection* algorithm, which automatically searches for useful inputs (discussed later in this document).

Generalized Regression Neural Networks

Generalized regression neural networks (GRNNs) work in a similar fashion to PNNs, but perform regression rather than classification tasks. As with the PNN, Gaussian kernel functions are located at each training case. Each case can be regarded as evidence that the response surface is a given height at that point in input space, with progressively decaying evidence in the immediate vicinity. The

GRNN copies the training cases into the network to be used to estimate the response on new points. The output is estimated using a weighted average of the outputs of the training cases, where the weighting is related to the distance of the point from the point being estimated (so that points nearby contribute most heavily to the estimate).

The first hidden layer in the GRNN contains the radial units. A second hidden layer contains units which help to estimate the weighted average. This is a specialized procedure. Each output has a special unit assigned in this layer which forms the weighted sum for the corresponding output. To get the weighted average from the weighted sum, the weighted sum must be divided through by the sum of the weighting factors. A single special unit in the second layer calculates the latter value. The output layer then performs the actual divisions (using special "division" units). Hence, the second hidden layer always has exactly one more unit than the output layer. In regression problems, typically only a single output is estimated, and so the second hidden layer usually has two units.

The GRNN can be modified by assigning radial units which represent clusters rather than each individual training case: this reduces the size of the network and increases execution speed. Centers can be assigned using any appropriate algorithm (i.e. sub-sampling, *K*-means or Kohonen), and *ST Neural Networks* adjusts the internal weightings to take account.

GRNNs have advantages and disadvantages broadly similar to PNNs - the difference being that GRNNs can only be used for regression problems, whereas

PNNs are used for classification problems. A GRNN trains almost instantly, but tends to be large and slow (although, unlike PNNs, it is not necessary to have one radial unit for each training case, the number still needs to be large). Like an RBF network, a GRNN does not extrapolate.

Linear Networks

A general scientific principal is that a simple model should always be chosen in preference to a complex model, if the latter does not fit the data better. In terms of function approximation, the simplest model is the linear model, where the fitted function is a hyper plane. In classification, the hyper plane is positioned to divide the two classes (a linear discriminant function); in regression, it is positioned to pass through the data. A linear model is typically represented using an *NxN* matrix and an *Nx1* bias vector.

In neural network terms, a linear model is represented by a network having no hidden layers, but an output layer with fully linear units (that is, linear units with linear activation function). The weights correspond to the matrix, and the thresholds to the bias vector. When the network is executed, it effectively multiplies the input by the weights matrix, then adds the bias vector.

In *ST Neural Networks*, you can define a linear network, and train it using the standard pseudo-inverse (SVD) linear optimization algorithm (Golub and Kahan, 1965). Of course, linear optimization is available in *STATISTICA* Multiple

Regression (see Multiple Regression); however, *ST Neural Network'* s linear network has the advantage of allowing you to compare performance with "real" neural networks within a single environment.

The linear network provides a good benchmark against which to compare the performance of your neural networks. It is quite possible that problems which are thought to be highly complex can actually be solved as well by linear techniques as neural networks. If you have only a small number of training cases, you are probably anyway not justified in using a more complex model.

Kohonen Networks

Kohonen networks are used quite differently to the other networks in *ST Neural Networks*. Whereas all the other networks are designed for supervised learning tasks, Kohonen networks are designed primarily for *unsupervised learning* (see Kohonen, 1982; Haykin, 1994; Patterson, 1996; Fausett, 1994).

Whereas in supervised learning the training data contains cases featuring input variables together with the associated outputs (and the network must infer a mapping from the inputs to the outputs), in unsupervised learning the training data set contains only input variables.

At first glance this may seem strange. Without outputs, what can the network learn? The answer is that the Kohonen network attempts to learn the structure of the data.

One possible use is therefore in exploratory data analysis. The Kohonen network can learn to recognize clusters of data, and can also relate similar classes to each other. The user can build up an understanding of the data, which is used to refine the network. As classes of data are recognized, they can be labeled, so that the network becomes capable of classification tasks. Kohonen networks can also be used for classification when output classes are immediately available - the advantage in this case is their ability to highlight similarities between classes.

A second possible use is in novelty detection. Kohonen networks can learn to recognize clusters in the training data, and respond to it. If new data, unlike previous cases, is encountered, the network fails to recognize it and this indicates novelty.

A Kohonen network has only two layers: the input layer, and an output layer of radial units (also known as the *topological map* layer). The units in the topological map layer are laid out in space - typically in two dimensions (although *ST Neural Networks* also supports one-dimensional Kohonen networks).

Kohonen networks are trained using an iterative algorithm. Starting with an initially-random set of radial centers, the algorithm gradually adjusts them to reflect the clustering of the training data. At one level, this compares with the sub-sampling and *K*-Means algorithms used to assign centers in RBF and GRNN networks, and indeed the Kohonen algorithm can be used to assign centers for these types of networks. However, the algorithm also acts on a different level.

The iterative training procedure also arranges the network so that units representing centers close together in the input space are also situated close together on the topological map. You can think of the network's topological layer as a crude two-dimensional grid, which must be folded and distorted into the N-dimensional input space, so as to preserve as far as possible the original structure. Clearly any attempt to represent an N-dimensional space in two dimensions will result in loss of detail; however, the technique can be worthwhile in allowing the user to visualize data which might otherwise be impossible to understand.

The basic iterative Kohonen algorithm simply runs through a number of epochs, on each epoch executing each training case and applying the following algorithm:

- Select the winning neuron (the one who's center is nearest to the input case);
- Adjust the winning neuron to be more like the input case (a weighted sum of the old neuron center and the training case).

The algorithm uses a time-decaying learning rate, which is used to perform the weighted sum and ensures that the alterations become more subtle as the epochs pass. This ensures that the centers settle down to a compromise representation of the cases, which cause that neuron to win.

The topological ordering property is achieved by adding the concept of a neighborhood to the algorithm. The neighborhood is a set of neurons surrounding the winning neuron. The neighborhood, like the learning rate, decays over time, so that initially quite a large number of neurons belong to the neighborhood (perhaps almost the entire topology map); in the latter stages the neighborhood

will be zero (i.e. consists solely of the winning neuron itself). In the Kohonen algorithm, the adjustment of neurons is actually applied not just to the winning neuron, but to all the members of the current neighborhood.

The effect of this neighborhood update is that initially quite large areas of the network are "dragged towards" training cases - and dragged quite substantially. The network develops a crude topological ordering, with similar cases activating clumps of neurons in the topological map. As epochs pass the learning rate and neighborhood both decrease, so that finer distinctions within areas of the map can be drawn, ultimately resulting in fine-tuning of individual neurons. Often, training is deliberately conducted in two distinct phases: a relatively short phase with high learning rates and neighborhood, and a long phase with low learning rate and zero or near-zero neighborhood.

Once the network has been trained to recognize structure in the data, it can be used as a visualization tool to examine the data. The *Win Frequencies Datasheet* (counts of the number of times each neuron wins when training cases are executed) can be examined to see if distinct clusters have formed on the map. Individual cases are executed and the topological map observed, to see if some meaning can be assigned to the clusters (this usually involves referring back to the original application area, so that the relationship between clustered cases can be established). Once clusters are identified, neurons in the topological map are labeled to indicate their meaning (sometimes individual cases may be labeled, too). Once the topological map has been built up in this way, new cases can be submitted to the network. If the winning neuron has been labeled with a class

name, the network can perform classification. If not, the network is regarded as undecided.

Kohonen networks also make use of the accept threshold, when performing classification. Since the activation level of a neuron in a Kohonen network is the distance of the neuron from the input case, the accept threshold acts as a maximum recognized distance. If the activation of the winning neuron is greater than this distance, the Kohonen network is regarded as undecided. Thus, by labeling all neurons and setting the accept threshold appropriately, a Kohonen network can act as a novelty detector (it reports undecided only if the input case is sufficiently dissimilar to all radial units).

Kohonen networks are inspired by some known properties of the brain. The cerebral cortex is actually a large flat sheet (about 0.5m squared; it is folded up into the familiar convoluted shape only for convenience in fitting into the skull!) with known topological properties (for example, the area corresponding to the hand is next to the arm, and a distorted human frame can be topologically mapped out in two dimensions on its surface).

Where are Neural Networks being used?

Neural networks are performing successfully where other methods do not, recognizing and matching complicated, vague, or incomplete patterns. Neural networks have been applied in solving a wide variety of problems.

The most common use for neural networks is to project what will most likely happen. There are many areas where prediction can help in setting priorities. For example, the emergency room at a hospital can be a hectic place, to know who needs the most critical help can enable a more successful operation. Basically, all organizations must establish priorities, which govern the allocation of their resources. Neural networks have been used as a mechanism of knowledge acquisition for expert system in stock market forecasting with astonishingly accurate results. Neural networks have also been used for bankruptcy prediction for credit card institutions.

Although one may apply neural network systems for interpretation, prediction, diagnosis, planning, monitoring, debugging, repair, instruction, and control, the most successful applications of neural networks are in categorization and pattern recognition. Such a system classifies the object under investigation (e.g. an illness, a pattern, a picture, a chemical compound, a word, the financial profile of a customer) as one of numerous possible categories that, in return, may trigger the recommendation of an action (such as a treatment plan or a financial plan.

A company called Nestor, have used neural network for financial risk assessment for mortgage insurance decisions, categorizing the risk of loans as good or bad. Neural networks has also been applied to convert text to speech, NETtalk is one of the systems developed for this purpose. Image processing and pattern recognition form an important area of neural networks, probably one of the most actively research areas of neural networks.

An other of research for application of neural networks is character recognition and handwriting recognition. This area has use in banking, credit card processing and other financial services, where reading and correctly recognizing handwriting on documents is of crucial significance. The pattern recognition capability of neural networks has been used to read handwriting in processing checks, the amount must normally be entered into the system by a human. A system that could automate this task would expedite check processing and reduce errors. One such system has been developed by HNC (Hecht-Nielsen Co.) for BankTec.

One of the best known applications is the bomb detector installed in some U.S. airports. This device called SNOOPE, determine the presence of certain compounds from the chemical configurations of their components.

In a document from International Joint conference, one can find reports on using neural networks in areas ranging from robotics, speech, signal processing, vision, character recognition to musical composition, detection

of heart malfunction and epilepsy, fish detection and classification, optimization, and scheduling. One may take under consideration that most of the reported applications are still in research stage.

Basically, most applications of neural networks fall into the following five categories:

- **Prediction**

  Uses input values to predict some output. E.g. pick the best stocks in the market, predict weather, identify people with cancer risk.

- **Classification**

  Use input values to determine the classification. e.g. is the input the letter A, is the blob of the video data a plane and what kind of plane is it.

- **Data association**

Like classification but it also recognizes data that contains errors. e.g. not only identify the characters that were scanned but identify when the scanner is not working properly.

- **Data Conceptualization**

Analyze the inputs so that grouping relationships can be inferred. E.g. extract from a database the names of those most likely to by a particular product.

- **Data Filtering**

Smooth an input signal. e.g. take the noise out of a telephone signal.

CHAPTER 3

Methodology and System Analysis

Chapter 3 Methodology and System Analysis

The methodology taken to develop this project is the Multilayer (MLP) neural network that incorporated the Back propagation learning method within it.

3.1 Multilayer Perceptrons

This is the type of network discussed briefly in previous sections: the units each performed a biased weighted sum of their inputs and pass this activation level through a transfer function to produce their output, and the units are arranged in a layered feed forward topology. The network thus has a simple interpretation as a form of input-output model, with the weights and thresholds (biases) the free parameters of the model. Such networks can model functions of almost arbitrary complexity, with the number of layers, and the number of units in each layer, determining the function complexity. Important issues in MLP design include specification of the number of hidden layers and the number of units in these layers.

The number of input and output units is defined by the problem (there may be some uncertainty about precisely which inputs to use, a point to which we will return later. However, for the moment we will assume that the input variables are intuitively selected and are all meaningful). The number of hidden layers and units to use is far from clear. As good a starting point as any is to use one hidden layer, with the number of units equal to half the sum of the number of input and output units. Again, we will discuss how to choose a sensible number later.

3.1.1 Training Multilayer Perceptrons

Once the number of layers, and number of units in each layer, has been selected, the network's weights and thresholds must be set so as to minimize the prediction error made by the network. This is the role of the *training algorithms*. The historical cases, which you have gathered, are used to automatically adjust the weights and thresholds in order to minimize this error. This process is equivalent to fitting the model represented by the network to the training data available. The error of a particular configuration of the network can be determined by running all the training cases through the network, comparing the actual output generated with the desired or target outputs. The differences are combined together by an *error function* to give the network error. The most common error function is the *sum squared error*, where the individual errors of output units on each case are squared and summed together. *ST Neural Networks* reports the RMS (the above normalized for the number of cases and variables, then square-rooted), which neatly summarizes the error over the entire training set and set of output units.

In traditional modeling approaches (e.g. linear modeling) it is possible to algorithmically determine the model configuration, which absolutely minimizes this error. The price paid for the greater (non-linear) modeling power of neural networks is that although we can adjust a network to lower its error, we can never be sure that the error could not be lower still.

A helpful concept here is the error surface. Each of the *N* weights and thresholds of the network (i.e. the free parameters of the model) is taken to be a dimension in

space. The $N+1$th dimension is the network error. For any possible configuration of weights the error can be plotted in the $N+1$th dimension, forming an *error surface*. The objective of network training is to find the lowest point in this many-dimensional surface.

In a linear model with *sum squared error* function, this error surface is a parabola (a quadratic), which means that it is a smooth bowl-shape with a single minimum. It is therefore "easy" to locate the minimum.

Neural network error surfaces are much more complex, and are characterized by a number of unhelpful features, such as local minima (which are lower than the surrounding terrain, but above the global minimum), flat-spots and plateaus, saddle-points, and long narrow ravines.

It is not possible to analytically determine where the global minimum of the error surface is, and so neural network training is essentially an exploration of the error surface. From an initially random configuration of weights and thresholds (i.e. a random point on the error surface), the training algorithms incrementally seek for the global minimum. Typically, this is done by calculating the gradient (slope) of the error surface at the current point, and then using that information to make a downhill move. Eventually, the algorithm stops in a low point, which may be a local minimum (but hopefully is the global minimum).

3.2 The Back Propagation Algorithm

The best-known example of a neural network training algorithm is *back propagation*. Modern second-order algorithms such as *conjugate gradient descent* and *Levenberg-Marquardt* (both included in *ST Neural Networks*) are substantially faster (e.g. an order of magnitude faster) for many problems, but *back propagation* still has advantages in some circumstances, and is the easiest algorithm to understand. We will introduce this now, and discuss the more advanced algorithms later. There are also heuristic modifications of *back propagation* which work well for some problem domains, such as *quick propagation* (Fahlman, 1988) and *Delta-Bar-Delta* (Jacobs, 1988) and are also included in *ST Neural Networks*.

In *back propagation*, the gradient vector of the error surface is calculated. This vector points along the line of steepest descent from the current point, so we know that if we move along it a "short" distance, we will decrease the error. A sequence of such moves (slowing as we near the bottom) will eventually find a minimum of some sort. The difficult part is to decide how large the steps should be.

Large steps may converge more quickly, but may also overstep the solution or (if the error surface is very eccentric) go off in the wrong direction. A classic example of this in neural network training is where the algorithm progresses very slowly along a steep, narrow, valley, bouncing from one side across to the other. In contrast, very small steps may go in the correct direction, but they also require a large number of iterations. In practice, the step size is proportional to the slope

(so that the algorithms settles down in a minimum) and to a special constant: the *learning rate*. The correct setting for the learning rate is application-dependent, and is typically chosen by experiment; it may also be time-varying, getting smaller as the algorithm progresses.

The algorithm is also usually modified by inclusion of a momentum term: this encourages movement in a fixed direction, so that if several steps are taken in the same direction, the algorithm "picks up speed", which gives it the ability to (sometimes) escape local minimum, and also to move rapidly over flat spots and plateaus.

The algorithm therefore progresses iteratively, through a number of *epochs*. On each epoch, the training cases are each submitted in turn to the network, and target and actual outputs compared and the error calculated. This error, together with the error surface gradient, is used to adjust the weights, and then the process repeats. The initial network configuration is random, and training stops when a given number of epochs elapses, or when the error reaches an acceptable level, or when the error stops improving (you can select which of these stopping conditions to use).

3.2.1 Over-learning and Generalization

One major problem with the approach outlined above is that it doesn't actually minimize the error, which we are really interested in - which is the expected error the network will make when *new* cases are submitted to it. In other words, the most desirable property of a network is its ability to *generalize* to new cases. In

reality, the network is trained to minimize the error on the training set, and short of having a perfect and infinitely large training set, this is not the same thing as minimizing the error on the "real" error surface - the error surface of the underlying and unknown model (see Bishop, 1995).

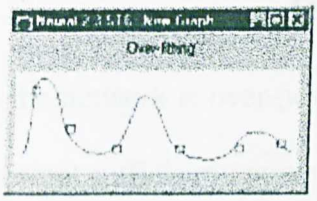
The most important manifestation of this distinction is the problem of over-learning, or over-fitting. It is easiest to demonstrate this concept using polynomial curve fitting rather than neural networks, but the concept is precisely the same.

A polynomial is an equation with terms containing only constants and powers of the variables. For example:

$y=2x+3$

$y=3x^2+4x+1$

Different polynomials have different shapes, with larger powers (and therefore larger numbers of terms) having steadily more eccentric shapes. Given a set of data, we may wish to fit a polynomial curve (i.e. a model) to explain the data. The data is probably noisy, so we don't necessarily expect the best model to pass exactly through all the points. A low-order polynomial may not be sufficiently flexible to fit close to the points, whereas a high-order polynomial is actually too flexible, fitting the data exactly by adopting a highly eccentric shape which is actually unrelated to the underlying function (as shown in the figure below).



Neural networks have precisely the same problem. A network with more weights models a more complex function, and is therefore prone to over-fitting. A network with less weights may not be sufficiently powerful to model the underlying function. For example, a network with no hidden layers actually models a simple linear function.

How then can we select the "right" complexity of network? A larger network will almost invariably achieve a lower error eventually, but this may indicate over-fitting rather than good modeling.

The answer is to use *cross verification*. Some of the training cases are reserved, and not actually used for training in the *back propagation* algorithm. Instead, they are used to keep an independent check on the progress of the algorithm. It is invariably the case that the initial performance of the network on training and verification sets is the same (if it is not at least approximately the same, the division of cases between the two sets is probably biased). As training progresses, the training error naturally drops, and providing training is minimizing the true error function, the verification error drops too. However, if the verification error stops dropping, or indeed starts to rise, this indicates that the network is starting to overfit the data, and training should cease (you can configure *ST Neural Networks* to stop automatically once over-learning starts to occur). When over-fitting occurs during the training process like this it is called over-learning. In this case, it is usually advisable to decrease the number of hidden units and/or hidden layers, as the network is over-powerful for the problem at hand. In contrast, if the network is not sufficiently powerful to model the underlying function, over-learning is not

likely to occur, and neither training nor verification errors will drop to a satisfactory level.

The problems associated with local minimum, and decisions over the size of network to use, imply that using a neural network typically involves experimenting with a large number of different networks, probably training each one a number of times (to avoid being fooled by local minimum), and observing individual performances. The key guide to performance here is the verification error. However, following the standard scientific precept that, all else being equal, a simple model is always preferable to a complex model, you may also select a smaller network in preference to a larger one with a negligible improvement in verification error.

A problem with this approach of repeated experimentation is that the verification set does actually play a key role in selecting the model, which means that it is actually part of the training process. Its reliability as an independent guide to performance of the model is therefore compromised - with sufficient experiments, you may just hit upon a "lucky" network which happens to perform well on the verification set. To add confidence in the performance of the final model, it is therefore normal practice (at least where the volume of training data allows it) to reserve a third set of cases - the test set. The final model is tested with the test set data, to ensure that the results on the verification and training set are real, and not artifacts of the training process. Of course, to fulfill this role properly the test set should be used only once - if it is in turn used to adjust and reiterate the training process, it effectively becomes verification data!

To summarize, network design (once the input variables have been selected) follows a number of stages:

- Select an initial configuration (typically, one hidden layer with the number of hidden units set to half the sum of the number of input and output units (*ST Neural Network'* s Network Advisor will automatically suggest this configuration).

- Iteratively, conduct a number of experiments with each configuration, retaining the best network (in terms of verification error) found. *ST Neural Networks* automatically retains the best network for you as you experiment. A number of experiments are required with each configuration to avoid being fooled if training locates a local minimum.

- On each experiment, if under-learning occurs (the network doesn't achieve an acceptable performance level) try adding more neurons to the hidden layer(s). If this doesn't help, try adding an extra hidden layer.

- If over-learning occurs (verification error starts to rise) try removing hidden units (and possibly layers).

Since repeated heuristic experimentation is at best tedious, *ST Neural Networks* includes an automatic search algorithm to perform this process for you. The *Automatic Network Designer* experiments with different numbers of hidden units, performs a number of training runs with each network architecture tested, selecting the best network on the basis of verification error biased by network size. The *Automatic Network Designer*'s sophisticated search algorithms,

including *simulated annealing* (Kirkpatrick et. al., 1983), can test hundreds of combinations of networks, concentrating on particularly promising networks, and can also find a "rough-and-ready " solution quite quickly.

3.3 Data Selection

All the above stages rely on a key assumption. Specifically, the training, verification and test data must be representative of the underlying model (and, further, the three sets must be independently representative). The old computer science adage "garbage in, garbage out" could not apply more strongly than in neural modeling. If training data is not representative, than the models worth is at best compromised. At worst, it may be useless. It is worth spelling out the kind of problems which can corrupt a training set:

The future is not the past. Training data is typically historical. If circumstances have changed, relationships which held in the past may no longer hold.

All eventualities must be covered. A neural network can only learn from cases that are present. If people with incomes over $100,000 per year are a bad credit risk, and your training data includes nobody over $40,000 per year, you cannot expect it to make a correct decision when it encounters one of the previously-unseen cases.

A network learns the easiest features it can. A classic (possibly apocryphal) illustration of this is a vision project designed to automatically recognize tanks. A network is trained on a hundred pictures including tanks, and a hundred not. It achieves a perfect 100% score. When tested on new data, it proves hopeless. The reason? The pictures of tanks are taken on dark, rainy days; the pictures without on sunny days. The network learns to distinguish the (trivial matter of) differences in overall light intensity. To work, the network would need training cases including all weather and lighting conditions under which it is expected to operate - not to mention all types of terrain, angles of shot, distances...

Unbalanced data sets. Since a network minimizes an overall error, the proportion of types of data in the set is critical. A network trained on a data set with 900 good cases and 100 bad will bias its decision towards good cases, as this allows the algorithm to lower the overall error (which is much more heavily influenced by the good cases). If the representation of good and bad cases is different in the real population, the network's decisions may be wrong. A good example would be disease diagnosis. Perhaps 90% of patients routinely tested are clear of a disease. A network is trained on an available data set with a 90/10 split. It is then used in diagnosis on patients complaining of specific problems, where the likelihood of disease is 50/50. The network will react over-cautiously and fail to recognize disease in some unhealthy patients. In contrast, if trained on the "complainants" data, and then tested on "routine" data, the network may raise a high number of false positives. In such circumstances, the data set may need to be crafted to take account of the distribution of data (e.g. you could replicate the less

numerous cases, or remove some of the numerous cases), or the network's decisions modified by the inclusion of a *loss matrix*(Bishop, 1995). Often, the best approach is to ensure even representation of different cases, then to interpret the network's decisions accordingly.

3.4 Why BackPROP is Necessary ?

Simple neural networks have a severe limitation: they cannot compute certain classes of functions. So, a more general architecture is a set of neurons with two or more modifiable sets of weights.

Where from this multiplayer architecture, we still have an input and an output set of units. The middle groups of units are called the hidden layers since they are neither input nor output, and their activities are not accessible from outside the network. It is not hard to show that if the network contains nonlinearities, it can have more computing power than one without a hidden layer. For example, it solves problems such as Exclusive OR.

The network must be nonlinear because a many layer linear network reduces to a single layer network. Consider a three layer linear network. Assume the input pattern is **f.** The connection matrix, **A**, connects input pattern, f, with hidden layer pattern, g. If this is a linear network, then: Af=g. A connection matrix, B,connects the hidden layer pattern, g, with output layer pattern, h.

Therefore: Bg = h

But since the network is linear,

$$Bg = h = B(Af)=h$$

$$Baf = h.$$

We can replace a linear network with network with a hidden layer with a network that a single layer of weights with connection strengths given by the matrix produce, BA. Therefore we gain no processing power.

The major technical problem in making an adaptive multilayer network is proving a rule for modifying the weights at the hidden layers. If we have only a single layer of weights, we have two effective error-correcting algorithms, the preceptron and the Widrow-Hoff rule. They both depend on being able to measure the error and the fact that only the weights on a single unit contribute to the error for that unit. But consider a more complex architecture. The weights on a unit contribute to the error of the unit. But so do the weights from the input layer to all the hidden layer units. If there is an error, which of these many possible contributors caused it? If the network gets the answer right, who did the right thing? This is called the credit-assignment problem and it is a difficulty for all complex systems, including neural networks, governments, armies, and business organizations.

Backpropagation gives a rule that solves the credit-assignment problem for a feed forward network using nonlinear units. The network is called feed forward because information goes only from input to output; there are no recurrent or backward connections. The need for nonlinearity requires us to use the general version of the generic neural network neuron. The nonlinear second stage is almost always considered to contain a monotonic sigmond function.

3.5 Systems development software

They' re a lot of platform software that can use to develop the project:

1. **Windows and win32 programming: c, c++**
2. **Visual BASIC**
3. **MATLAB – Neural Network Toolbox**

   -- Image processing Toolbox

In this project, the MATLAB~Neural network ToolBox and Image processing are use to develop the system.

About Neural network ToolBox

The neural network Toolbox extends the MATLAB technical computing environment to facilitate the design and simulation of many kinds of artificial neural networks. It is easily combined with other MATLAB toolboxes and simulink nonlinear simulation software to support research and practical applications in diverse fields of science, engineering, and business.

Why Using MATLAB- neural network Toolbox

Inspired by the biological nervous system, artificial neural network technology is being used to solve a wide variety of complex scientific, engineering and business problems.

Neural networks are ideally suited for such problems because like their biological counterparts, an artificial neural network can learn, and therefore can be trained to find solutions, recognize patterns, classify data, and forecast future events.

In contrast to classical approaches in fields such as statistics and control theory, neural nets require no explicit model or limiting assumptions of normality or linearity. Neural networks are a uniquely powerful tool in applications where formal analysis would be extremely difficult or impossible, such as pattern recognition and nonlinear system identification and control.

The behavior of a neural network is defined by the way its individual computing elements are connected and by the strength of those connections or weights. The weights are automatically adjusted by training the network according to a specified learning rule until it performs with the desired error rate.

Because neural networks require intensive matrix computations, Matlab provides a natural framework for rapidly implementing them and for studying their behavior and application.

The neural network Toolbox provides comprehensive support for the design, implementation, and simulation of many proven network paradigms, and is completely extensible and customizable. Its consistent methodology and modular organization facilitate research, provide a flexible framework for experimentation, and simplify customization.

A single-layer feed-forward network. The unique notation simplifies understanding of network architectures.

The neural network toolbox imposes no limitations on the number of layers or input sets. Virtually any combination of neurons and layers, with delays if required, can be trained in a number of ways.

About Image Processing Toolbox

The Image Processing ToolBox is a comprehensive set of image processing and analysis tools designed to support a broad range of applications requiring images. The image Processing Toolbox builds upon the MATLAB foundation to provide a unique solution that combines high-level images processing and analysis functions with a powerful and extensible technical computing environment. MATLAB and the image processing ToolBox provide a completely interactive, feature-rich environment that helps you speed the development of applications involving image processing and analysis. The Image processing ToolBox features a robust set of image processing algorithm and provides direct access to MATLAB' s comprehensive foundation of numeric computation, data analysis, algorithm development, and visualization capabilities. Together, MATLAB and the Image Processing ToolBox deliver a broad, flexible set of tools that can be used to address a wide range of multidisciplinary problems.

3.5.1 Advantages of MATLAB

Many image solution currently available to engineers and scientists lack the breadth and depth of capabilities required for these multidisciplinary problems. Image processing software programs such as GUI-base application packages suffer from fixed functionality and cross-platform incompatibility, while C/C++ libraries require a substantial programming effort on the part of the user.

The image Processing Toolbox, in contrast, combines high-level, easy to use functions for images processing and analysis with functions for image processing and analysis with MATLAB' s open and extensible technical computing

environment. This solution eliminates the functional limitations of GUI-based packages and the time-consuming coding and debugging start-up work associated with c/c++ libraries – while offering the convenience and flexibility of both. With the image Processing Toolbox you can focus on the "big picture" ---extracting reliable answer from your image data.

Programming tools in MATLAB 5 include an interactive visual debugger for algorithm development and refinement, a performance profiler to optimize runtime performance of algorithms in applications, and an interactive graphical user interface builder to rapidly develop custom graphical front ends for your applications.

Because neural network require intensive matrix computation, MATLAB provides a natural framework for rapidly implementing them and for studying their behavior and application. The neural Network Toolbox provides comprehensive support for the design, implementation, and simulation of many proven network paradigms, and is completely extensible and customizable.

CHAPTER 4

The Chinese character systems design

CHAPTER 4 The Chinese Character Recognition Systems design

To do Chinese character recognition using an MLP, I assume the input layer of the network to be a set image pixels, which can take on analogue (or gray scale) values between 0 and 1. The two-dimensional set of pixels is mapped onto the set of input neurons in a fairly arbitrary way: For an N x N images, the top row of N pixels is associated with the first N neurons, the next row of N pixels is associated with the next N neurons, and so forth. At the start of the training process, the network has no knowledge of underlying two-dimensional structure of the problem (that is, if a pixel is on, nearby pixels in the two-dimensional nature of the problem during the learning process.)

I will teach the network the alphabet of 5 Chinese characters. To encourage generalization, we show the net many different hand-written version of each character. The 20- image training set is shown in Figure 4.1. There are only 5 outputs we want the net to give, so we use 5 output neurons and map the output pattern: first neuron on, rest off, to the character "妮" second neuron on, rest off, to "武" and so on. Such an encoding scheme works well in a small scale, but is clearly unworkable for mapping with large output sets. In such cases, one

would prefer a more compact output encoding, with possibly an additional layer of hidden units to produce the more complex outputs.

The MLP is used only for the part of the algorithm where one matches to templates. Given any fixed set of exemplars, a neural network will usually learn this set perfectly, but the performance under generalization can be very poor. In fact, the more weights there are, the faster the learning (in the sense of number of iterations, not of CPU time), and the worse the ability to generalize.

Figure 4.1 The training set of 20 handwritten Characters

4.1 Requirement for the system design

The system design will based on the 3-basis requirement. There are:

a.) A three- layer Back propagation neural network is to be simulated for character recognition purposes.

b.) Characters are to be captured into image files using a paint utility.

c.) The network is to be trained using the data from the image files.

The Chinese character recognition system that will develop for this project is going to fulfill the entire above requirement.

4.1.1 Image files format in used

The image inputs are from the user handwriting on the drawing box or previously save image. Input files of neural networks also need to be loaded into the program if Supervised learning is to be carried out. For the unsupervised learning, on existing neural network can be loaded if the user wants to add on a new item to the list. The images are fed into the imaging system, and then they are converted into an internal format so that they can be manipulated expediently and uniformly. The internal format used is the Windows device-independent bitmap (DIB) format.

Image files in used is BITMAP(BMP) format. A bitmap is a data structure that represents a rectangular image. An image is made up of many pixels (picture elements), 256 in the case of image 16 pixels wide and 16 high. Each pixel is in turn represented by one or more bits in the bitmap (which is literally a map of bits). As an example consider a monochrome (2 color) image in which a black pixel may be represented by an 'on' bit (1) while a white pixel is represented by an "off" bit (0); in this case the image has 1-bit-per-pixel (1bpp) because only 1 bit is required to hold all the color information for a single pixel. If the image had contained 4 colors then we would require 2bpp (00=white, 01=black, 10=red, 11=blue) and 4bpp for 16 colors.

The file format for a windows style device-independent bitmap is shown as below. It should be noted that the module developed would only load Windows bitmaps; alternative formats have been ignored for simplicity.

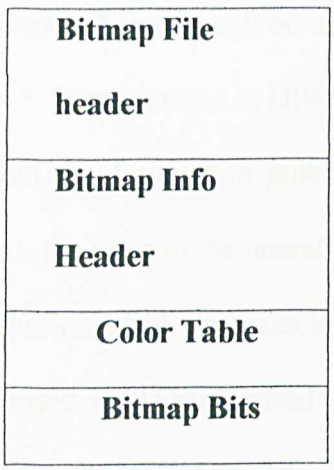


Figure 4.2 Bitmap file Format

The file format header is a data structure that holds information on the size of the file and the type of the file ('BM' in the case of a bitmap); it also contains a pointer to the bitmap bits. Immediately following the file header is the bitmap info header, this provides details on the width and height of the bitmap, number of bits per pixel and compression mode used. The color table describes what colors the bit patterns in the image relate to and the bitmap bits store the actual image. It should be pointed out that the bitmap bits segment consists of a single steam of bits; this means that in order to access the 4^{th} pixel in line 6 of the image you have to use the offset (6*width)+4 from the beginning of the bits. This offset system is complicated by the fact that the width of a bitmap is rounded up to fit on a 16-bit

boundary, so if your image was 76 pixels wide it will be stored in a bitmap 80 pixels wide.

4.1.2 Character Digitization and Recognition

The drawn character will be digitized into certain resolution such as 6 x 8, 9 x12, or 5 x 5. Using images in DIB enables them to be processed easily and efficiently. Without the digitization process, recognition process is not possible. The pixel Grids is the input to the neural network.

Recognition enables a character to be recognized under neural network with either Supervised or Unsupervised learning. For the Supervised learning, a neural network must exist before it can be performed. The algorithms for both digitization and recognition will be discussed later.

4.2 Modules

Based on the design, three main modules needs to be developed:

a.) Back propagation Network Modules

-Simulates a three-layer back propagation neural network.

b.) Image Processing Module

- Includes reading decoding and processing Bitmap image files.

c.) A front end tool for the system

- Provides a graphical interface to the whole system.

Summary

The design of the Chinese character recognition system has been decided upon that meet all the requirement of the thesis. The characters will be captured into BITMAP image files. These are the files that will read, decoded and process into input data for the network. The whole system will consist of three subsystems, namely the back propagation Network Module, the Image processing Module and the front-end whole system.

CHAPTER 5

Implementation and Strategy

Chapter 5 Implementation & Strategy

5.1 Problem Statement

A network is to be designed and trained to recognize the 10 Chinese Characters. For this thesis purpose, an image processing system that digitizes each character centered in the system's field of vision was also designed. Each character is represented as 8X10 grid of Boolean.

The image the have been utilize may be no perfect and the characters may suffer from noise and also rotation. The 10 80-element input vectors are defined as a matrix of input vectors. The target vectors are also defined. Each targets vector is a 10-element vector with a 1 in the position of the letter it represents, and 0's everywhere else.

Examples,

田 1000000000

十 0010000000

口 0001000000

王 0000100000

方 0000010000

水 0000001000

右 0000000010

5.2 Implementation & Strategy

This section will look into how the system is implemented. The development Environment will also be described.

5.2.1 Development environment

This system has been developed in the following environment:

Operating System

Microsoft Windows 98/ME/XP/2000 or any platform that support the MATLAB.

Software

1. ***Matlab neural network toolbox***
2. ***Matlab image processing toolbox***
3. ***Adobe Photoshop***

5.2.2 Implementation

The implementation can divide to 4 sections:

1. *Convert the Image to binary code*
2. *Neural network training section*
3. *Image processing section (add noise)*
4. *Image processing section (Rotation)*
5. *Front end for the system*

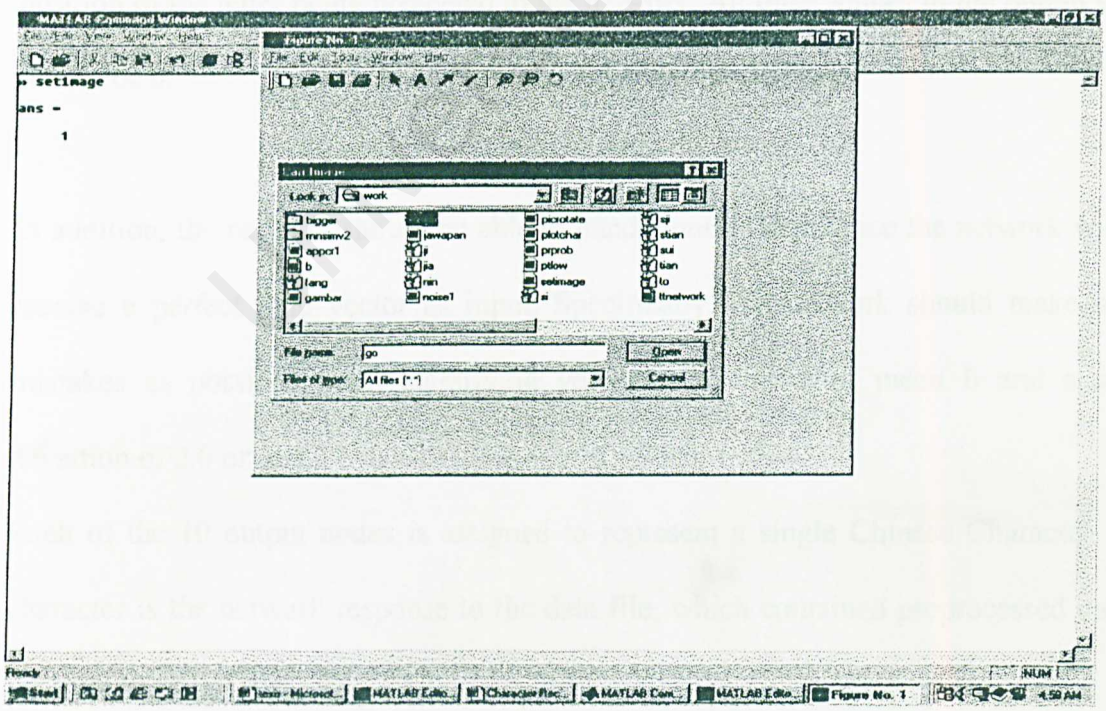
For the neural network training, Matlab neural network toolbox was used to development the system. Image processing module was implemented use Matlab image processing Toolbox with help of Adobe Photoshop 6 and also Corel Draw 10.

Convert the Image to Binary code.

To make the work easy in converting the image, a piece of code has been created in a module called **setimage.m** using MATLAB M-script. When the M-script has been run, a dialog box will pop up to select the image. The system was automatically converted to binary code in selected bit map matrix. It was make the process of converting to binary code more accurate and time effective.

The coding shows as below.

```
gcf
[fn,pn]=uigetfile('*.*','Cari Image');
imshow(fn);
A=imread(fn);
B=imresize(A,[8,10]);
C=im2bw(B)
```



After the binary code has been capture, it was save at the file called **PRPROB.M.**

Neural network training section

Neural network was used to train the system to recognize the character. The system use the binary image form **PRPROB.m** File to train the network to recognize the character. After the training was complete and the performance goal has been meet, then save in a file called **ptnetwork.mat**. Each time, when we want run the system, we just need to load the **.MAT** file.

Network setup

The neural network being used is a feed-forward and fully inter-connected three-layer neural network, the weighs of which are modified by back propagation algorithm. It consist 10 80-element input vectors and 10-element output vector. The 26-element letter represents a letter. To operate correctly the network should respond with a 1 in the position of the letter being presented to the network. All other values in the output vector should be 0.

In addition, the network should be able to handle noise. In practice the network will not receive a perfect bole vector as input. Specifically, the network should make a few mistakes as possible when classifying vectors with noise of mean 0 and standard deviation of 2.0 or less.

Each of the 10 output nodes is assigned to represent a single Chinese Character. This character is the network response to the data file, which contained preprocessed data of the bitmap image file.

Image processing

In this project, the entire image was process by the MATLAB image processing toolbox. The noises are rotation is some of the image manipulation that had used.

Training Data set preparation

Before data file can be created(PRPROB.M), the BITMAP image files need to prepare. In this thesis, the utilized BITMAP file consists of 10 Chinese characters.

Front end for the system

Another module was developed as the Chinese Character recognition system Interface. This module also can be use to test the accuracy system in recognition.

Different module was created with various files to utilize the system. The following is a list of the files used:

A. Start up file **--- run.m**

B. Load file **--- ptnetwork.mat**

C. Training & Network setup file **---appcr1.m**

D. Image processing file **---setimage.m, picrotate.m, noise1.m**

E. Display file **--- ptlow.m, ptlowrot.m**

CHAPTER 6 *Testing*

Chapter 6 Testing

The entire system was dividing into 6 main modules. Each of the modules was separately developed and test. After testing and the verification of the accuracy and correctness of each module' s functionalities, the 3 modules was brought together and integrated to form the entire system.

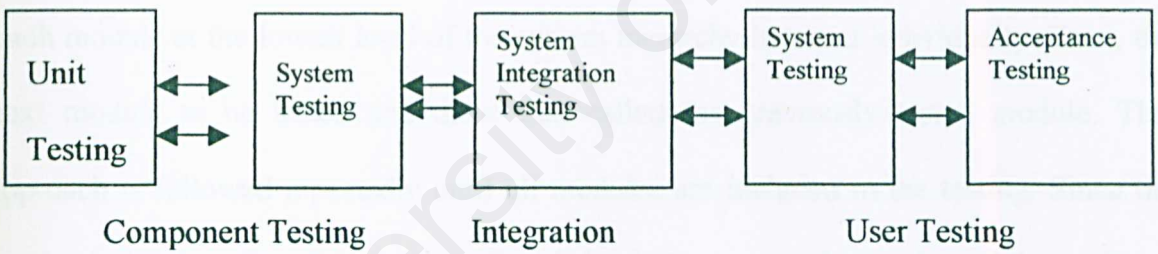
The testing procedure can be dividing to 3 sections:

I. Unit Testing

II. System Testing

III. System Integration Testing

The Process of testing can be explained as below.



6.1 Unit Testing

This test is carried out on each isolated module. This is to ensure that individual module perform what is required. This section of testing is to ensure the coding is implemented correctly.

6.2. Integration Testing

The main objective for this section is to ensure the whole system can be combine and running in correct mode. This also to make sure the modules communicate in the right way with each other.

When the individual component are working correctly and meet the objective and requirement, these components are combined into working system. In other word, integration testing is the process of verifying that system components work together as described in the system and program design specification. Each module is linked together to form a complete system.

For this Chinese Character recognition system, a bottom up approach has been used. Each module at the lowest level of the system hierarchy is tested individually. Then, the next module to be tested and those that called the previously tested module. This approach is followed repeatedly until all modules are included in the testing. Since the system is developed modularly, error found should be corrected in each module easily.

6.3 System Testing

The last testing procedure done is system testing. Testing system is very different from the unit testing and also integration testing. The objective of the unit testing and integration testing is to ensure that the code implemented the design properly. In system

testing, a very different objective is to be achieved to ensure that system does what to customer want it to do. In other word, to test the system to ensure it to fulfill the requirement. We collect enough samples from the users let say 10 sets. We test each set separately, and each set of sample result is collected. Calculate the average percentage of accuracy of the system, if it is over 90%, then we can concluded that the system was success.

CHAPTER 7 *Problem*

Chapter 7 Problem

Similar to most theses undertaken, this thesis is not without problems. This chapter serves to explain the hardship encountered in the process of developing this process.

Lack of materials on the character recognition using neural network.

The material about character recognition using neural network was difficult to find out. And because of unfamiliar in this sort of system, the actual idea about how the system should be design and implanted was unclear. The lack of examples using Matlab to design the Character recognition system was also cause the system cannot complete in time.

Lack of knowledge on Neural network.

This weakness causes the system that have been designed early cannot achieve the objective of this project. The lack of the knowledge on neural network had led to misunderstanding how the neural network work to train the recognition system.

Unfamiliar with Matlab software package

With the lack knowledge in neural network and absolutely no idea about Matlab software package especially Matlab Neural Network and Matlab Image Processing Toolbox, this

proved to be an even more daunting task in design and implemented the system. Much time has been allocated to understand these programming methodologies.

Personally, the MATLAB development tool was very difficult to master. It looks very confusing. The material about the Matlab was very hard to get, the only way can depend on was Internet. The company that developed the MATLAB tool have a portal website that packed with the MATLAB information. In the process to learn the skill, the website was help a lot.

The complexity of Chinese Character.

The Chinese character was very complexity. This is adding the difficulty in processing the character image. Some of the character has so many strokes that very difficult to recognize at some bit map matrix. Then, some of the Chinese characters are almost the same. Example as below:

已巳

于干

料斜

CHAPTER 8

Limitation And Future Enhancement

Chapter 8 Limitation and future enhancement

8.1 Limitation

1. The form of image must in Windows bitmap format.
2. The character must at 10X8 bit map for each character.
3. The system only can recognize the character that has been adding noise. For the character that have been rotated, the system always recognize incorrectly.
4. Currently, the system can only recognize 10 Chinese characters that had trained.
5. The result of recognition was not perfect.
6. The system Can only run at MATLAB platform
7. The system was not user friendly.

8.2 FUTURE ENHAMCEMENTS

Overall, this system has fulfilled all the requirements of this thesis. However, that is nothing perfect in this world. Room for improvement still available. Some of the enhancement could be made as below:

I. Support more character.

II. Provide higher percentage of accuracy by using better algorithm or more training set.

III. Support multiple kinds of forms. It would no just limit to redesigned form by the developer.

IV. Enhance speed of recognition.

V. Ore friendly and easier to use.

VI. Include all error checking of the system, system recovery and backup recovery.

VII. Train the network to be robust. Due to time constraint, the training data set prepared are limited whereby only noisy and rotation has been taken into consideration. This resulted a network that is not robust. The network could only recognize those with which it has been trained. Other free-form Chinese Character may no be recognized.

VIII. The network performance depends on the training data set with which it has been trained. A comprehensive training data set should include of different rotation, level of noise, translation and scaling. A network trained with such a comprehensive training data set would be robust enough to recognize characters regardless of their position, size and orientation.

8.3 Future research

In this thesis, I just use one algorithm to implement the project. Actually, there are still a lot of algorithm or method which according to IEEE paper.

Anyway, there is quite impossible to try all method in just a few months because it is too time consuming. Due to this, the algorithm used already proofed but still not perfect. I hope that the combination of few algorithms will give more accuracy of recognition.

References

Data & Analysis Center for software, *Artificial Neural networks Technology*, 1992 (http://www.dacs.dtis.mil/techs/neural.title.html, printed November 1998)

Elaine Rich and Kelvin Knight : Artificial Intelligence. McGraw Hill 1991, Second Edition.

Freeman, J.A, & Skapura, D.M (1991). *Neural Networks: Algorithms, applications, and programming techniques*. Reading, MA: Addison-Wesley.

Geva,shlomo;tte, Joaquin : Adaptive Nearest Neighbour Pattern Classification, IEEE *Transactions on Neural Networks*, 1991, Vol.2, No.2.

Haykin simon, *Neural Network*, 1994 Macmillan College Publishing Company Inc.

Patrick K. Simpson,pp. 112-123, Artificial Neural Systems, 1990, Pergamon Press, New York, NY

Stallings, William: *Approached to Chinese character Recognition. Pattern recognition* 1976, Vol. 8, 87-98.

Stephen I.gallant, Chapter 11 &12; *Neural NetworkLearning and expert systems*, 1993, The MIT Press, Cambridge, MA

Suchenwirth, Richard; Guo, Jun: Hartmann,IrmFried; Hincha, Georg; Krause, Manfred;Zhang, Zheng : Optical Recognition of Chinese characters. *Advances in control systems and signal Processing* 1989, Vol. 8.

www. Mathworks.com

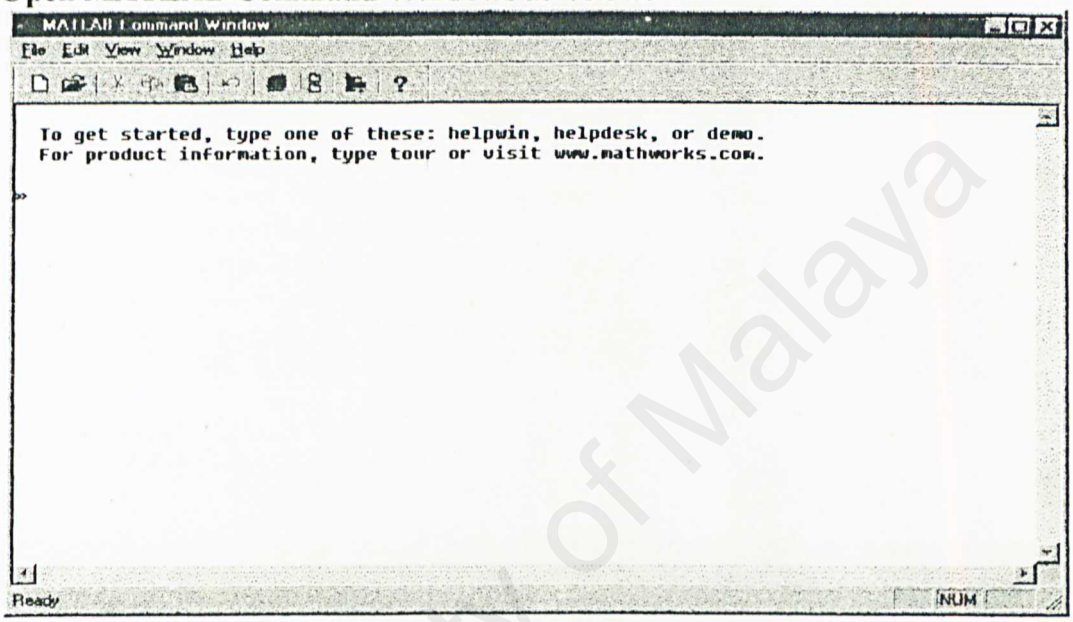
CHINESE CHARACTER RECOGNITION SYSTEM

USER MANUAL

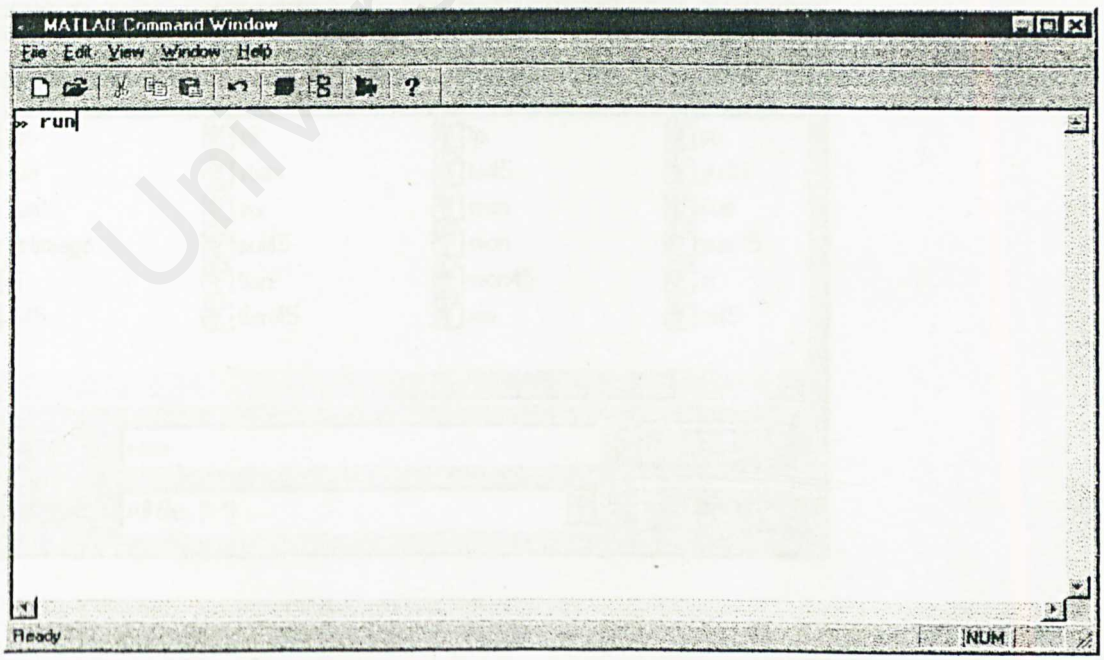
USER MANUAL FOR CHINESE CHARACTER RECOGNITION SYSTEM

1. **Unzip the *ptnetwork.zip***

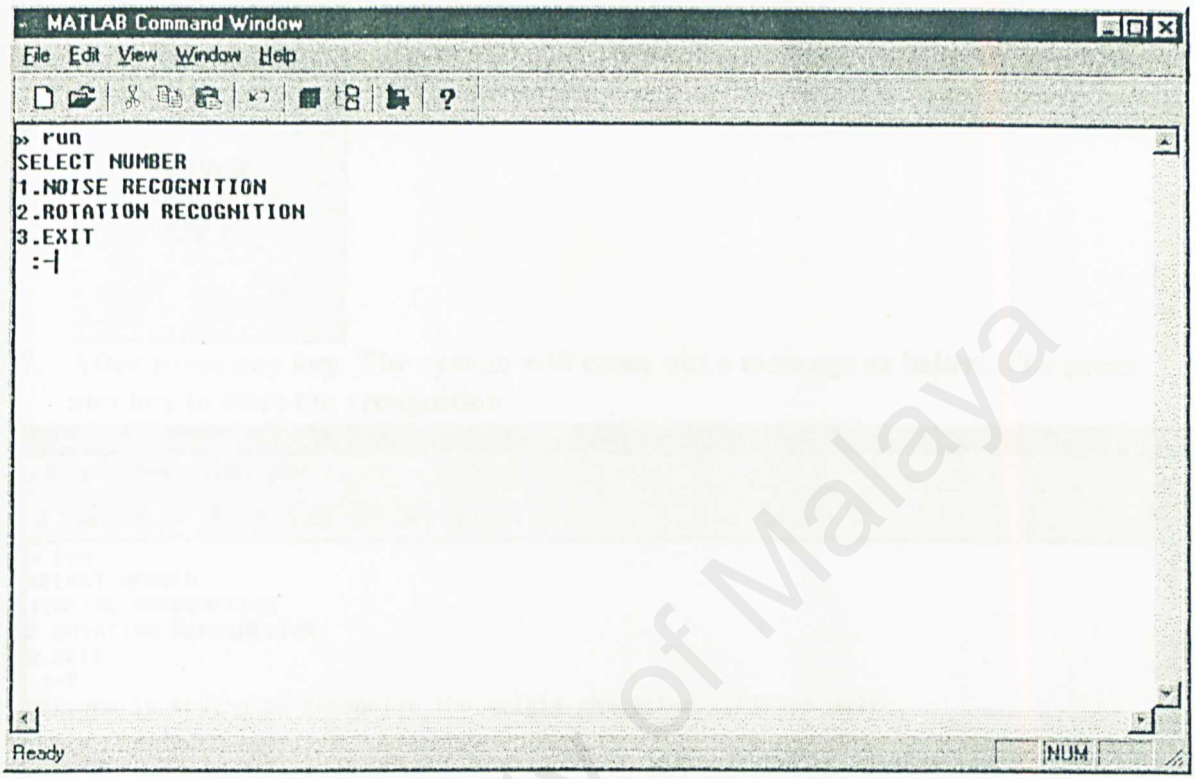
2. **Open MATLAB Command Windows as below.**



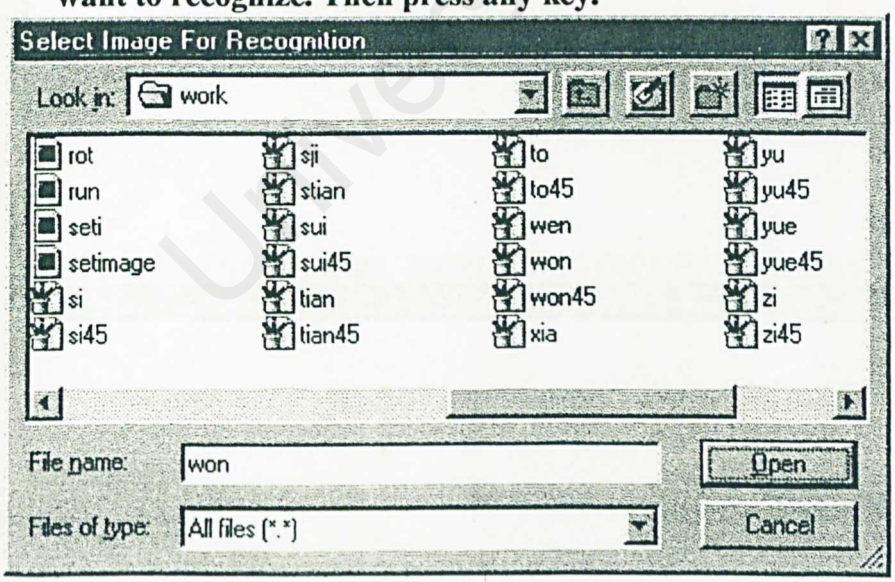
3. **The type load *run* in the command prompt.**



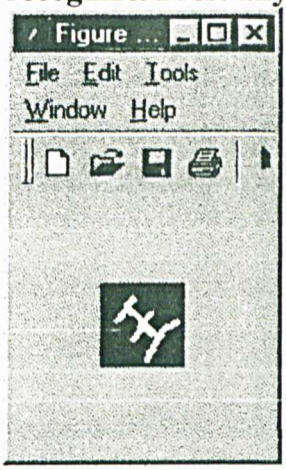
4. Type the number. 1 -for recognize the character that have been noise added.
   2 -for recognize the character that have been rotated 45°.
   3 –Exit the program.



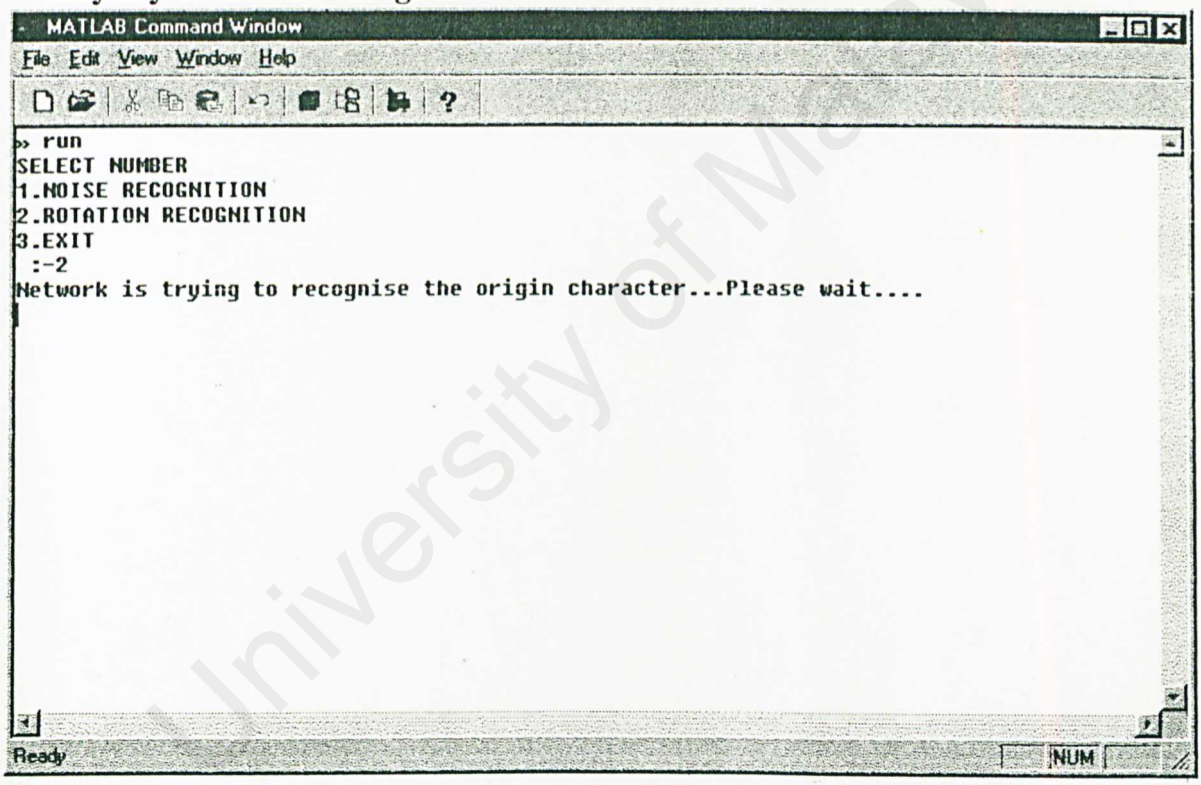
5. For selection 1 or 2, a dialog box with pop up. Select the image file that user want to recognize. Then press any key.



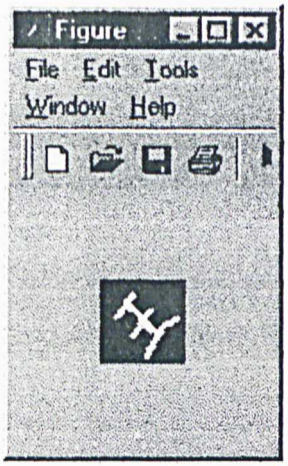
6. **The character recognition system will show the character that need to recognize. Press any key to continue.**



7. **After press any key. The system will come out a message as below. The press any key to start the recognition.**



8. **Then system will show the result of recognition. Example as blow.**



9. **If user need to run the program for another recognition work, just type in *run*.**

APPENDIX – PROGRAM LISTING

APPENDIX --- PROGRAM LISTING

APPENDIX -- PROGRAM LISTING

This system using Matlab to developed the coding. The overall system was dividing onto several functions and modules.

1. ***run.m***
   --- Startup file which automatically load the training file (***ptnetwork8.mat***). User can to choose to run rotation recognition and noisy recognition.

2. ***appcr1.m***
   ---- create the network to train the character. The training algorithm have been use is Traingdx.

3. ***prprob.m***
   ---- Defines a matrix Huruf which contains the bit maps of the 10 letters of the character. This file also defines target vectors TARGETS for each letter. Each target vector has 10 elements with all zeros, except for a single 1. A has a 1 in the first element, B in the second, etc.

4. ***picrotate.m***
   ---- This file processing the image and rotate . Then, the rotation output will show out.

5. ***rot.m***
   ---The rotated image that have been convert to binary image.It defines the bit maps of the 10 character which have been rotated.

6. ***noise1.m***
   --- Match the binary image with the Bitmap format iamge to show the noisy image.

7. ***jawapan.m***
   --- Show the result of the recognition in the Bitmap formats.

8. ***gambar.m***
   --- Convert the binary image to the actual resized Bitmap format image.

9. ***ptlowrot.m***
   --- Recognise the rotated image and present the result.

10. ***ptlow.m***
    --- Recognise the noisy vector and present the result.

```
%run.m

load ptnetwork8

TTT =input('SELECT NUMBER \n1.NOISE RECOGNITION \n2.ROTATION
RECOGNITION\n3.EXIT\n :-');

if TTT ==1
      ptlow

elseif TTT==2
   ptlowrot
end
```

```
% APPCR1.m

Chinese Character recognition.

clf;
figure(gcf)

echo on


%    NEWFF   - Inititializes feed-forward networks.
%    TRAINGDX - Trains a feed-forward network with faster
backpropagation.
%    SIM   - Simulates feed-forward networks.

pause % Strike any key to continue...

%    DEFINING THE MODEL PROBLEM
%    ==========================

%    The script file PRPROB defines a matrix HURUF
%    which contains the bit maps of the 10 letters of the
%    alphabet.

%    This file also defines target vectors TARGETS for
%    each letter.  Each target vector has 10 elements with
%    all zeros, except for a single 1.  A has a 1 in the
%    first element, B in the second, etc.

[huruf,targets] = prprob;
[rot45] = rot;
[RR1] =size(rot45);
%[SS2] =size(obj);
[R,Q] = size(huruf);
[S2,Q] = size(targets);

pause % Strike any key to define the network...

%    DEFINING THE NETWORK
%    ====================

%    The character recognition network will have 25 TANSIG
%    neurons in its hidden layer.

S1 = 10;
net = newff(minmax(huruf),[S1 S2],{'logsig' 'logsig'},'traingdx');
net.LW{2,1} = net.LW{2,1}*0.01;
net.b{2} = net.b{2}*0.01;

pause % Strike any key to train the network...

%    TRAINING THE NETWORK WITHOUT NOISE
%    ==================================

net.performFcn = 'sse';          % Sum-Squared Error performance function
net.trainParam.goal = 0.1;       % Sum-squared error goal.
```

```
net.trainParam.show = 20;       % Frequency of progress displays (in
epochs).
net.trainParam.epochs = 5000;   % Maximum number of epochs to train.
net.trainParam.mc = 0.95;       % Momentum constant.

%    Training begins...please wait...

P = huruf;
T = targets;

[net,tr] = train(net,P,T);

%    ...and finally finishes.

pause % Strike any key to train the network with noise...

%    TRAINING THE NETWORK WITH NOISE
%    ===============================

%    A copy of the network will now be made.  This copy will
%    be trained with noisy examples of letters of the alphabet.

netn = net;

netn.trainParam.goal = 0.6;     % Mean-squared error goal.
netn.trainParam.epochs = 300;   % Maximum number of epochs to train.

%    The network will be trained on 10 sets of noisy data.

pause % Strike any key to begin training...

%    Training begins...please wait...

T = [targets targets targets targets];
for pass = 1:10
  fprintf('Pass = %.0f\n',pass);
  P = [huruf, huruf, ...
       (huruf + randn(R,Q)*0.1), ...
       (huruf + randn(R,Q)*0.2)];


    [netn,tr] = train(netn,P,T);
  echo off
end
echo on

%    ...and finally finishes.

pause % Strike any key to finish training the network...

%    TRAINING THE SECOND NETWORK WITHOUT NOISE
%    ========================================

%    The second network is now retrained without noise to
%    insure that it correctly categorizes non-noizy letters.

netn.trainParam.goal = 0.1;     % Mean-squared error goal.
```

```
netn.trainParam.epochs = 500;  % Maximum number of epochs to train.
net.trainParam.show = 5;       % Frequency of progress displays (in
epochs).

%    Training begins...please wait...

P = huruf;
T = targets;

[netn,tr] = train(netn,P,T);

%    ...and finally finishes.

pause % Strike any key to test the networks...

%    TRAINING THE NETWORK WITHOUT ROTATION
%    ====================================

net.performFcn = 'sse';        % Sum-Squared Error performance function
net.trainParam.goal = 0.1;     % Sum-squared error goal.
net.trainParam.show = 20;      % Frequency of progress displays (in
epochs).
net.trainParam.epochs = 5000;  % Maximum number of epochs to train.
net.trainParam.mc = 0.95;      % Momentum constant.

%    Training begins...please wait...

P = huruf;
T = targets;

[net,tr] = train(net,P,T);

%    ...and finally finishes.

pause % Strike any key to train the network with noise...



%    TRAINING THE NETWORK WITH Rotation
%    ==================================

%    A copy of the network will now be made.  This copy will
%    be trained with noisy examples of letters of the alphabet.

netm = net;

netm.trainParam.goal = 0.6;    % Mean-squared error goal.
netm.trainParam.epochs = 300;  % Maximum number of epochs to train.

%    The network will be trained on 10 sets of Rotation data.

pause % Strike any key to begin training...

%    Training begins...please wait...

%T = [targets targets targets targets];
```

```
P = rot45;
T = targets;
for pass = 1:10
 fprintf('Pass = %.0f\n',pass);

  [netm,tr] = train(netm,P,T);
  echo off
end
echo on

%    ...and finally finishes.

pause % Strike any key to finish training the network...

%    TRAINING THE SECOND NETWORK WITHOUT ROTATION
%    ===========================================

%    The second network is now retrained without rotation to
%    insure that it correctly categorizes non-noizy letters.

netm.trainParam.goal = 0.1;     % Mean-squared error goal.
netm.trainParam.epochs = 500;   % Maximum number of epochs to train.
net.trainParam.show = 5;        % Frequency of progress displays (in
epochs).

%    Training begins...please wait...

P = huruf;
T = targets;

[netm,tr] = train(netm,P,T);

%    ...and finally finishes.

pause % Strike any key to test the networks...



%    TRAINING THE NETWORK
%    ====================

% SET TESTING PARAMETERS
noise_range = 0:.05:.5;
max_test = 100;
network1 = [];
network2 = [];
network3 = [];
network4 = [];
T = targets;

% PERFORM THE TEST
for noiselevel = noise_range
  fprintf('Testing networks with noise level of %.2f.\n',noiselevel);
  errors1 = 0;
  errors2 = 0;
```

```
  for i=1:max_test
    P = huruf + randn(240,10)*noiselevel;
      % TEST NETWORK 1
    A = sim(net,P);
    AA = compet(A);
    errors1 = errors1 + sum(sum(abs(AA-T)))/2;

    % TEST NETWORK 2
    An = sim(netn,P);
    AAn = compet(An);
    errors2 = errors2 + sum(sum(abs(AAn-T)))/2;
    echo off
 end
  % AVERAGE ERRORS FOR 100 SETS OF 26 TARGET VECTORS.
   network1 = [network1 errors1/10/100];
   network2 = [network2 errors2/10/100];
end
echo on

errors3 = 0;
errors4 =0;
for i=1:max_test
P = rot45;
% TEST NETWORK 3
 B = sim(net,P);
BB = compet(B);
errors3 = errors3+ sum(sum(abs(BB-T)))/2;
 %TEST NETWORK 4

   Bn = sim(netm,P);
   BBn = compet(Bn);
   errors2 = errors4 + sum(sum(abs(BBn-T)))/2;
   echo off
end

  % AVERAGE ERRORS FOR 100 SETS OF 26 TARGET VECTORS.
    network3 = [network3 errors3/10/100];
   network4 = [network4 errors4/10/100];
end
echo on
pause % Strike any key to display the test results...

%    DISPLAY RESULTS
%    ===============

%    Here is a plot showing the percentage of errors for
%    the two networks for varying levels of noise.

clf
plot(noise_range,network1*100,'--',noise_range,network2*100);
title('Percentage of Recognition Errors');
xlabel('Noise Level');
ylabel('Network 1 - -   Network 2 ---');

%    Network 1, trained without noise, has more errors due
%    to noise than does Network 2, which was trained with noise.
```

```
clf
plot(noise_range,network3*100,'--',noise_range,network4*100);
title('Percentage of Recognition Errors');
xlabel('Noise Level');
ylabel('Network 3 - -   Network 4 ---');


echo off

disp('End of APPCR1')
```

Prprob.m

```
function [huruf,targets] = prprob()

   letter1 = [ 0     0     0     0     0     0     1     1     1     1     0     0
               0     0     0     1     1     1     1     1     0     1     0     0
               0     1     1     1     1     1     0     0     0     1     0     0
               0     1     1     0     0     1     0     0     0     1     0     0
               0     1     0     0     0     1     0     0     0     1     0     0
               0     1     0     0     0     1     0     0     0     1     0     0
               0     1     0     0     0     1     1     1     0     1     0     0
               0     1     0     0     1     1     1     1     0     1     0     0
               0     1     0     1     1     1     1     0     0     1     0     0
               0     1     0     1     1     1     0     0     1     1     0     0
               0     1     0     0     0     1     0     0     1     1     0     0
               0     1     0     0     0     1     0     0     1     0     0     0
               0     1     0     0     0     1     0     0     1     0     0     0
               0     1     1     0     0     1     0     0     1     0     0     0
               0     1     1     0     1     1     1     1     1     0     0     0
               0     1     1     1     1     0     0     1     1     0     0     0
               0     0     1     1     0     0     0     0     0     0     0     0
               0     0     0     0     0     0     0     0     0     0     0     0
               0     0     0     0     0     0     0     0     0     0     0     0
               0     0     0     0     0     0     0     0     0     0     0     0]'



letter2 =[     0     0     0     0     0     0     0     0     0     0     0     0
               0     0     0     0     0     0     0     0     0     0     0     0
               0     0     0     0     0     0     0     0     0     0     0     0
               0     0     1     0     0     1     0     0     0     0     0     0
               0     0     1     1     0     1     0     0     0     0     0     0
               0     0     0     1     0     1     0     0     0     0     0     0
               0     0     0     0     0     1     0     0     0     0     0     0
               0     0     1     0     0     1     0     1     1     0     0     0
               0     0     1     0     0     1     1     1     1     0     0     0
               0     0     0     0     1     1     1     0     0     0     0     0
               0     0     1     1     1     1     0     0     0     0     0     0
               1     1     1     0     1     1     0     0     0     0     0     0
               0     1     0     0     1     0     1     0     0     0     0     0
               0     0     0     1     1     0     1     1     0     0     0     0
               0     0     0     1     0     0     0     1     1     0     0     0
               0     0     1     1     0     0     0     0     1     0     0     0
               0     0     1     1     0     0     0     0     0     1     0     0
               0     0     0     0     0     0     0     0     0     0     1     0
               0     0     0     0     0     0     0     0     0     0     0     0
               0     0     0     0     0     0     0     0     0     0     0     0]'
```

```
letter3 = [  0     0     0     0     0     0     0     0     0     0     0     0
             0     0     0     1     0     0     0     0     0     0     0     0
             0     0     0     1     0     0     1     0     1     0     0     0
             0     0     0     1     0     1     1     0     1     0     0     0
             0     0     0     1     0     1     1     0     1     0     0     0
             0     0     0     1     0     0     0     1     0     0     0     0
             0     1     0     1     0     0     0     1     1     1     0     0
             0     1     0     0     1     1     1     0     0     0     1     0
             0     1     0     1     1     0     0     0     0     1     0     0
             0     1     0     0     0     0     0     0     0     1     0     0
             0     1     0     0     1     1     1     0     1     1     0     0
             0     0     0     0     0     1     1     0     0     0     0     0
             0     0     0     0     0     1     1     0     0     0     0     0
             0     0     0     0     0     0     1     1     1     0     0     0
             0     0     0     0     1     1     1     1     1     1     0     0
             0     0     0     1     1     1     1     0     0     0     0     0
             0     0     0     0     0     0     1     0     0     0     0     0
             0     0     0     0     0     0     1     0     0     0     0     0
             0     0     0     0     0     1     1     0     0     0     0     0
             0     0     0     0     0     0     0     0     0     0     0     0]'
```

```
letter4 = [  0     0     0     0     0     0     0     0     0     0     0     0
             0     0     0     0     0     0     0     0     0     0     0     0
             0     0     0     0     0     0     1     1     1     0     0     0
             0     0     0     0     1     1     1     1     1     1     0     0
             0     0     0     1     1     1     1     0     1     1     0     0
             0     0     1     1     1     0     0     0     1     1     0     0
             0     0     1     1     0     0     0     0     1     1     0     0
             0     0     1     1     0     0     0     0     1     1     0     0
             0     0     1     1     0     0     0     0     1     1     0     0
             0     0     1     1     0     0     0     0     1     1     0     0
             0     0     1     1     0     0     0     0     1     1     0     0
             0     0     1     1     0     0     0     0     1     0     0     0
             0     0     1     1     0     0     0     0     1     0     0     0
             0     0     1     1     0     0     0     0     1     0     0     0
             0     0     0     1     1     1     1     1     1     0     0     0
             0     0     0     1     1     1     1     1     0     0     0     0
             0     0     0     0     1     1     0     0     0     0     0     0
             0     0     0     0     0     0     0     0     0     0     0     0
             0     0     0     0     0     0     0     0     0     0     0     0
             0     0     0     0     0     0     0     0     0     0     0     0]'
```

```
letter5 = [0     0     0     0     0     0     0     0     0     0     0     0
           0     0     0     0     0     0     0     0     0     0     0     0
           0     0     0     0     0     0     0     0     0     0     0     0
           0     0     0     0     0     0     1     1     0     0     0     0
           0     0     0     0     1     1     1     1     0     0     0     0
           0     0     0     0     1     0     1     0     0     0     0     0
           0     0     0     0     0     0     1     0     0     0     0     0
           0     0     0     0     0     0     1     0     0     0     0     0
           0     0     0     0     0     0     1     0     0     0     0     0
           0     0     0     0     0     1     1     1     0     0     0     0
           0     0     0     1     1     1     1     0     0     0     0     0
           0     0     0     0     1     0     1     0     0     0     0     0
           0     0     0     0     0     0     1     0     0     0     0     0
           0     0     0     0     0     0     1     0     0     0     0     0
           0     0     0     0     0     1     1     1     1     1     0     0
           0     0     1     1     1     0     0     0     0     0     0     0
           0     0     0     0     0     0     0     0     0     0     0     0
           0     0     0     0     0     0     0     0     0     0     0     0
           0     0     0     0     0     0     0     0     0     0     0     0]'
```

```
letter6 = [ 0     0     0     0     0     0     0     0     0     0     0     0
            0     0     0     0     0     1     0     0     0     0     0     0
            0     0     0     0     0     1     0     0     0     0     0     0
            0     0     0     0     0     1     0     1     1     0     0     0
            0     0     0     0     0     0     1     1     1     0     0     0
            0     0     0     0     1     1     1     0     0     0     0     0
            0     0     1     1     1     1     1     0     0     0     0     0
            0     0     1     1     0     1     0     0     0     0     0     0
            0     0     0     0     0     1     1     1     0     0     0     0
            0     0     0     0     1     1     1     0     1     0     0     0
            0     0     0     0     1     0     0     0     1     0     0     0
            0     0     0     1     0     0     0     0     1     0     0     0
            0     0     0     1     0     0     0     0     1     0     0     0
            0     0     1     1     0     0     0     0     1     0     0     0
            0     0     1     0     0     0     1     0     1     0     0     0
            0     0     0     0     0     1     1     1     1     0     0     0
            0     0     0     0     0     0     0     1     1     0     0     0
            0     0     0     0     0     0     0     0     0     0     0     0
            0     0     0     0     0     0     0     0     0     0     0     0]'
```

```
letter7 = [0     0     0     0     0     0     0     0     0     0     0     0
           0     0     0     0     0     0     0     0     0     0     0     0
           0     0     0     0     0     1     0     0     0     0     0     0
           0     0     0     0     0     1     0     0     0     0     0     0
           0     0     0     0     0     1     0     0     0     0     0     0
           0     0     0     0     0     1     0     1     0     0     0     0
           0     0     0     1     1     1     1     1     0     0     0     0
           0     0     1     1     1     1     1     0     0     0     0     0
           0     0     0     1     0     1     1     0     0     0     0     0
           0     0     0     1     0     1     1     0     0     0     0     0
           0     0     1     1     1     1     1     0     0     0     0     0
           0     0     1     0     1     0     1     1     0     0     0     0
           0     1     1     0     1     0     0     1     0     0     0     0
           0     1     0     0     1     0     0     1     1     0     0     0
           0     0     0     0     1     0     0     0     1     1     0     0
           0     0     0     0     1     0     0     0     0     1     1     0
           0     0     0     0     0     0     0     0     0     0     0     0
           0     0     0     0     0     0     0     0     0     0     0     0
           0     0     0     0     0     0     0     0     0     0     0     0
           0     0     0     0     0     0     0     0     0     0     0     0]'

letter8 = [ 0     0     0     0     0     0     0     0     0     0     0     0
            0     0     0     0     0     0     0     0     0     0     0     0
            0     0     0     0     0     0     0     0     0     0     0     0
            0     0     0     0     0     0     1     0     0     0     0     0
            0     0     0     0     0     0     1     1     1     0     0     0
            0     0     0     0     0     0     1     1     1     0     0     0
            0     0     0     0     0     1     1     1     0     0     0     0
            0     0     0     1     1     1     0     0     0     0     0     0
            0     1     1     1     1     1     0     0     0     0     0     0
            0     1     1     0     1     1     0     1     1     1     0     0
            0     0     0     1     1     1     1     1     1     1     0     0
            0     0     0     1     1     1     1     0     0     1     0     0
            0     0     0     1     1     0     0     0     0     1     0     0
            0     0     1     0     1     0     0     0     1     0     0     0
            0     0     1     0     1     0     0     0     1     0     0     0
            0     0     1     0     1     1     1     1     1     0     0     0
            0     0     0     0     0     1     1     0     0     0     0     0
            0     0     0     0     0     0     0     0     0     0     0     0
            0     0     0     0     0     0     0     0     0     0     0     0
            0     0     0     0     0     0     0     0     0     0     0     0 ]'
```

```
letter9 =[  0  0  0  0  0  0  0  0  0  0  0  0
            0  0  0  0  0  0  0  0  0  0  0  0
            0  0  0  0  1  1  0  0  0  0  0  0
            0  0  0  1  1  0  0  0  0  0  0  0
            0  0  0  1  0  0  0  0  0  0  0  0
            0  0  1  1  0  1  0  0  0  0  0  0
            0  1  1  0  0  1  0  0  0  0  0  0
            0  1  1  0  0  1  0  1  0  0  0  0
            0  1  1  0  0  1  1  1  0  0  0  0
            0  1  1  1  1  1  0  0  0  0  0  0
            0  0  1  1  0  1  0  0  0  0  0  0
            0  0  0  0  0  1  0  1  1  0  0  0
            0  0  0  1  0  1  0  0  1  1  0  0
            0  0  1  1  0  1  0  0  0  1  1  0
            0  1  1  0  1  1  1  0  0  0  1  0
            0  0  0  0  0  0  0  0  0  0  0  0
            0  0  0  0  0  0  0  0  0  0  0  0
            0  0  0  0  0  0  0  0  0  0  0  0
            0  0  0  0  0  0  0  0  0  0  0  0
            0  0  0  0  0  0  0  0  0  0  0  0]'

letter10 = [ 0  0  0  0  0  0  0  0  0  0  0  0
             0  0  0  0  0  0  0  0  0  0  0  0
             0  0  0  0  0  0  0  0  0  0  0  0
             0  0  0  0  0  0  0  0  0  0  0  0
             0  0  0  0  0  0  1  1  1  0  0  0
             0  0  0  0  1  1  1  0  1  1  0  0
             0  0  0  1  1  1  0  0  0  1  0  0
             0  0  0  1  1  0  0  0  1  1  0  0
             0  0  0  1  0  0  0  0  1  1  0  0
             0  0  0  1  0  0  1  1  1  1  0  0
             0  0  0  1  1  1  1  0  1  1  0  0
             0  0  0  1  0  0  0  0  1  1  0  0
             0  0  0  1  0  0  0  0  1  0  0  0
             0  0  0  1  0  0  0  0  1  0  0  0
             0  0  0  1  0  0  1  1  1  0  0  0
             0  0  0  1  1  1  1  1  1  0  0  0
             0  0  0  0  0  0  0  0  0  0  0  0
             0  0  0  0  0  0  0  0  0  0  0  0
             0  0  0  0  0  0  0  0  0  0  0  0
             0  0  0  0  0  0  0  0  0  0  0  0]'

huruf = [letter1,letter2,letter3,letter4,letter5,letter6,letter7,letter8,...
        letter9,letter10];

targets = eye(10);
```

```
%picrorate.m

function picrotate(TT)

if TT==1
   A=imread('tian.bmp');
   AA=imrotate(A,45,'crop');
   imshow(AA);
end

if TT ==2
   A=imread('to.bmp');
   AA=imrotate(A,45,'crop');
    imshow(AA);
end

if TT ==3
   A=imread('xia.bmp');
   AA=imrotate(A,45,'crop');
    imshow(AA);
end

if TT==4
   A=imread('go.bmp');
   AA=imrotate(A,45,'crop');
    imshow(AA);
end


if TT ==5
   A=imread('won.bmp');
   AA=imrotate(A,45,'crop');
    imshow(AA);
end

if TT ==6
   A=imread('fang.bmp');
   AA=imrotate(A,45,'crop');
    imshow(AA);
end

if TT==7
   A=imread('sui.bmp');
   AA=imrotate(A,45,'crop');
   imshow(AA);
end

if TT==8
   A=imread('yu.bmp');
   AA=imrotate(A,45,'crop');
    imshow(AA);
end

if TT==9
   A=imread('yue.bmp');
   AA=imrotate(A,45,'crop');
```

```
    imshow(AA);
end


if TT==10
  A=imread('zi.bmp');
    AA=imrotate(A,45,'crop');
   imshow(AA);
end
```

Rot.m

```
function [rot45] = rot()

rot1 = [0     1     0     0     0     0     0     0     0     0     0     0
        1     1     1     0     0     0     0     0     0     0     0     0
        1     0     0     1     0     0     0     0     0     0     0     0
        1     0     0     0     1     0     0     0     0     0     0     0
        1     0     0     0     0     1     0     0     0     0     0     0
        1     0     0     0     0     0     1     0     0     0     0     0
        0     1     0     0     1     0     0     1     0     0     0     0
        0     0     1     1     1     1     0     1     1     0     0     0
        0     0     0     1     1     1     0     1     1     0     0     0
        0     0     0     1     1     0     0     0     1     1     0     0
        1     0     0     1     1     1     0     0     0     1     1     0
        1     1     0     1     0     0     1     0     0     0     1     1
        0     1     1     0     0     0     0     1     0     1     1     1
        0     0     1     1     0     0     0     0     1     1     0     0
        0     0     0     1     1     0     0     0     1     0     0     0
        0     0     0     0     1     1     1     0     1     0     0     0
        0     0     0     0     0     1     1     1     1     0     0     0
        0     0     0     0     0     0     1     0     0     0     0     0
        0     0     0     0     0     0     0     0     0     0     0     0
        0     0     0     0     0     0     0     0     0     0     0     0]'

rot2 = [0     0     0     0     0     0     0     0     0     0     0     0
        0     0     0     0     0     0     0     0     0     0     0     0
        0     0     0     0     0     0     0     0     0     0     0     0
        0     0     0     0     0     0     0     0     0     0     0     0
        0     0     0     0     0     0     0     0     0     0     0     0
        0     0     0     0     0     0     0     0     0     0     0     0
        0     1     0     0     0     1     1     0     0     0     0     0
        1     0     1     0     0     1     1     0     0     0     0     0
        1     1     0     1     0     1     0     0     0     0     0     0
        0     0     0     0     1     1     0     0     0     0     0     0
        0     1     0     0     1     1     0     0     0     0     0     0
        0     0     1     0     1     1     1     1     1     1     1     1
        0     0     0     1     1     0     1     0     1     0     1     1
        0     0     0     0     1     0     0     1     0     0     0     0
        0     0     0     1     1     0     1     1     0     0     0     0
        0     0     0     0     1     0     0     1     1     0     0     0
        0     0     0     0     0     0     0     1     1     0     0     0
        0     0     0     0     0     0     0     0     0     0     0     0
        0     0     0     0     0     0     0     0     0     0     0     0
        0     0     0     0     0     0     0     0     0     0     0     0]'
```

```
rot3=[ 0   0   0   0   0   0   0   0   0   0   0   0
       0   0   0   0   0   0   0   0   0   0   0   0
       0   0   1   0   0   0   0   0   0   0   0   0
       0   0   1   1   0   0   0   0   0   0   0   0.
       0   1   0   1   0   0   0   1   0   0   0   0
       0   1   1   1   0   1   0   1   0   0   0   0
       0   1   0   0   1   0   0   1   0   0   0   0
       1   0   0   0   1   0   0   0   1   0   0   0
       1   1   0   1   1   0   0   1   1   0   0   0
       0   1   0   1   0   0   1   0   0   0   0   0
       1   0   0   1   0   1   1   1   0   0   1   1
       1   1   0   0   0   1   1   1   1   1   1   1
       0   1   1   0   0   0   0   1   1   1   1   0
       0   0   1   1   0   0   0   0   1   1   1   0
       0   0   0   0   0   0   0   0   1   1   1   1
       0   0   0   0   0   0   0   0   1   0   0   1
       0   0   0   0   0   0   0   0   0   0   0   1
       0   0   0   0   0   0   0   0   0   0   0   0
       0   0   0   0   0   0   0   0   0   0   0   0
       0   0   0   0   0   0   0   0   0   0   0   0]'

rot4=[ 0   0   0   0   0   0   0   0   0   0   0   0
       0   0   0   0   0   0   0   0   0   0   0   0
       0   0   1   0   0   0   0   0   0   0   0   0
       0   1   1   1   1   0   0   0   0   0   0   0
       0   1   1   1   1   1   0   0   0   0   0   0
       0   1   1   0   1   1   1   0   0   0   0   0
       1   1   0   0   0   1   1   1   0   0   0   0
       1   1   0   0   0   0   1   1   1   0   0   0
       1   1   0   0   0   0   0   1   1   0   0   0
       1   1   1   0   0   0   0   0   1   1   0   0
       0   1   1   1   0   0   0   0   0   1   1   0
       0   0   1   1   1   0   0   0   0   0   1   1
       0   0   0   1   1   1   0   0   0   1   1   0
       0   0   0   0   1   1   1   0   1   1   1   0
       0   0   0   0   0   1   1   1   1   1   0   0
       0   0   0   0   0   0   0   1   1   1   0   0
       0   0   0   0   0   0   0   0   0   0   0   0
       0   0   0   0   0   0   0   0   0   0   0   0
       0   0   0   0   0   0   0   0   0   0   0   0
       0   0   0   0   0   0   0   0   0   0   0   0]'
```

```
rot5=[ 0     0     0     0     0     0     0     0     0     0     0     0
       0     0     0     0     0     0     0     0     0     0     0     0
       0     0     0     0     0     0     0     0     0     0     0     0
       0     0     0     0     0     0     0     0     0     0     0     0
       0     0     1     0     0     0     0     0     0     0     0     0
       0     1     1     0     0     0     0     0     0     0     0     0
       0     1     1     1     0     0     0     0     0     0     0     0
       0     1     0     1     1     0     0     0     0     0     0     0
       0     0     0     0     1     1     1     0     0     0     0     0
       0     0     0     0     0     1     1     0     0     0     0     0
       0     0     0     0     0     1     1     1     0     0     0     1
       0     0     0     0     1     1     0     1     1     0     1     1
       0     0     0     0     1     0     0     0     1     1     1     1
       0     0     0     0     0     0     0     0     1     1     1     0
       0     0     0     0     0     0     0     0     0     1     0     0
       0     0     0     0     0     0     0     0     0     1     0     0
       0     0     0     0     0     0     0     0     1     0     0     0
       0     0     0     0     0     0     0     0     0     0     0     0
       0     0     0     0     0     0     0     0     0     0     0     0
       0     0     0     0     0     0     0     0     0     0     0     0]'

rot6=[0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     1     0     0     0     0     0     0     0     0
      1     0     1     1     0     0     0     0     0     0     0     0
      1     0     1     0     0     0     0     0     0     0     0     0
      0     0     1     1     0     0     0     0     0     0     0     0
      0     1     1     1     0     0     0     1     0     0     0     0
      0     0     1     1     0     1     0     1     1     0     0     0
      0     1     1     0     1     1     0     0     1     1     0     0
      0     1     1     0     1     0     0     0     0     1     1     0
      0     0     0     0     0     1     0     0     0     0     1     1
      0     0     0     0     0     1     0     0     0     1     0     1
      0     0     0     0     0     1     1     0     0     1     1     1
      0     0     0     0     0     0     1     0     0     1     0     0
      0     0     0     0     0     0     1     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0     0]'
```

```
rot7=[0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0
      1   0   0   0   0   0   0   0   0   0   0   0
      1   0   0   1   0   0   0   0   0   0   0   0
      0   1   0   0   1   0   0   0   0   0   0   0
      0   0   1   1   1   0   0   0   0   0   0   0
      0   0   1   1   1   1   0   0   0   0   0   0
      0   1   1   1   1   1   1   0   0   0   0   0
      0   1   1   1   0   1   1   1   1   0   1   1
      0   0   0   1   1   1   0   0   0   1   1   1.
      0   0   0   1   1   0   1   0   0   0   0   0
      0   0   0   0   1   0   0   1   0   0   0   0
      0   0   0   0   1   1   0   0   1   0   0   0
      0   0   0   0   1   1   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0]'

rot8 = [0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   1   1   0   0   0   0   0   0   0
      0   1   1   1   1   0   0   0   0   0   0   0
      0   0   1   1   1   0   0   0   0   0   0   0
      0   0   0   1   0   0   0   1   1   0   0   0
      0   0   0   1   0   0   1   1   1   1   0   0
      0   0   1   1   1   0   1   1   0   0   1   0
      0   0   1   1   1   1   1   1   0   0   1   0
      0   0   1   0   1   1   1   0   0   0   1   1
      0   1   1   0   1   1   1   0   0   0   0   1
      0   0   0   0   0   1   1   1   0   0   1   0
      0   0   0   0   0   0   0   0   1   1   1   0
      0   0   0   0   0   0   1   0   0   0   0   0
      0   0   0   0   0   0   0   1   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0   0]'
```

```
rot9 = [0     0     0     0     0     0     0     0     0     0     0     0
        0     0     0     0     0     0     0     0     0     0     0     0
        0     0     0     0     0     0     0     0     0     0     0     0
        0     0     0     0     0     0     0     0     0     0     0     0
        1     0     0     0     0     0     0     0     0     0     0     0
        1     0     0     0     0     0     0     0     0     0     0     0
        1     0     0     0     0     0     0     0     0     0     0     0
        1     0     1     0     0     1     0     0     0     0     0     0
        1     1     0     1     0     1     0     0     0     0     0     0
        1     0     0     0     1     0     0     0     1     1     1     1
        1     1     0     0     1     1     0     0     0     1     0     0
        1     1     1     1     1     0     1     0     0     0     0     0
        0     1     1     1     1     0     0     1     0     1     0     0
        0     0     0     0     0     1     1     0     1     1     0     0
        0     0     0     0     0     0     1     0     1     0     0     0
        0     0     0     0     0     0     1     0     0     0     0     0
        0     0     0     0     0     1     1     0     0     0     0     0
        0     0     0     0     0     0     0     0     0     0     0     0
        0     0     0     0     0     0     0     0     0     0     0     0
        0     0     0     0     0     0     0     0     0     0     0     0]'

 rot10 =[0     0     0     0     0     0     0     0     0     0     0     0
         0     0     0     0     0     0     0     0     0     0     0     0
         0     0     0     0     0     0     0     0     0     0     0     0
         0     0     0     0     0     0     0     0     0     0     0     0
         0     0     0     1     1     1     0     0     0     0     0     0
         0     0     1     0     1     0     1     0     0     0     0     0
         0     0     1     1     0     1     1     1     0     0     0     0
         0     1     1     0     0     0     1     1     1     0     0     0
         0     0     1     0     0     0     1     1     1     1     0     0
         0     1     1     1     0     1     1     0     1     1     0     0
         0     0     1     1     0     1     1     0     0     1     1     0
         0     0     0     1     1     1     0     0     0     0     1     1
         0     0     0     0     1     1     0     0     0     1     1     1
         0     0     0     0     0     1     1     0     0     1     1     0
         0     0     0     0     0     0     1     1     0     1     0     0
         0     0     0     0     0     0     0     1     1     0     0     0
         0     0     0     0     0     0     0     0     0     0     0     0
         0     0     0     0     0     0     0     0     0     0     0     0
         0     0     0     0     0     0     0     0     0     0     0     0]'

rot45 = [rot1,rot2,rot3,rot4,rot5,rot6,rot7,rot8,...
         rot9,rot10];

obj =eye(10);
```

```
%gambar.m


A1=imread('tian.bmp');
AA1=imshow('tian.bmp');

A2=imread('to.bmp');
AA2=imshow('to.bmp');

A3=imread('si.bmp');
AA3=imshow('si.bmp');

A4=imread('go.bmp');
AA4=imshow('go.bmp');

A5=imread('won.bmp');
AA5=imshow('won.bmp');

A6=imread('fang.bmp');
AA6=imshow('fang.bmp');

A7=imread('sui.bmp');
AA7=imshow('sui.bmp');

A8=imread('yu.bmp');
AA8=imshow('yu.bmp');

A9=imread('yue.bmp');
AA9=imshow('yue.bmp');

A10=imread('zi.bmp');
AA10=imshow('zi.bmp');


if j==1
   AA1;
else
   if j==2
      AA2;
   else
      if j==3
         AA3;
      else
         if j==4
            AA4;
             else
         if j==5
          AA5;
           else
           if j==6
           AA6;
             else
           if j==7
              AA7;
              else
                 if j==8
```

```
                    AA8;
                    else
                      if j==9
                        AA9;
                        else
                        if j==10
                           AA10;
                        end
                      end
                  end
              end
          end
      end
   end
end
end
```

```
%jawapan.m

function jawapan(KK)

if KK ==1
   A=imread('tian.bmp');
   imshow(A);
end

if KK ==2
   A=imread('to.bmp');
    imshow(A);
end

if KK ==3
   A=imread('xia.bmp');
    imshow(A);
end

if KK ==4

A=imread('go.bmp');
 imshow(A);
end


if KK ==5
   A=imread('won.bmp');
   imshow(A);
end

if KK ==6
   A=imread('fang.bmp');
    imshow(A);
end

if KK ==7
   A=imread('sui.bmp');
    imshow(A);
end

if KK ==8
   A=imread('yu.bmp');
    imshow(A);
end

if KK ==9

A=imread('yue.bmp');
 imshow(A);
end


if KK ==10
   A=imread('zi.bmp');
    imshow(A);
end
```

```
%Noise1.m

function noise1(HH)

if HH==1
   A=imread('tian.bmp');
   AA=imnoise(A,'salt & pepper');
   imshow(AA);
end

if HH ==2
   A=imread('to.bmp');
    AA=imnoise(A,'salt & pepper');
   imshow(AA);

   end

if HH ==3
   A=imread('xia.bmp');
    AA=imnoise(A,'salt & pepper');
   imshow(AA);

end

if HH==4
   A=imread('go.bmp');
   AA=imnoise(A,'salt & pepper');
   imshow(AA);

end


if HH ==5
   A=imread('won.bmp');
   AA=imnoise(A,'salt & pepper');
   imshow(AA);

end

if HH==6
   A=imread('fang.bmp');
   AA=imnoise(A,'salt & pepper');
  imshow(AA);

end

if HH==7
   A=imread('sui.bmp');
   AA=imnoise(A,'salt & pepper');
   imshow(AA);

end

if HH==8
   A=imread('yu.bmp');
   AA=imnoise(A,'salt & pepper');
```

```
    imshow(AA);

end

if HH==9
    A=imread('yue.bmp');
    AA=imnoise(A,'salt & pepper');
    imshow(AA);

end

if HH==10
  A=imread('zi.bmp');
  AA=imnoise(A,'salt & pepper');
    imshow(AA);

end
```

Ptlowrot.m -select image and rotate the image and present the output.

```
[HH] = ambilpic;

PP=rot45(:,HH);
picrotate(HH);    %capture image

%picrotate(TT);
%pu(KK);
%B=imrotate(A,Q,'bilinear','crop');

pause  %Tekan  key untuk pemprosesan seterusnya.....


A3=sim(net,PP);
A3=compet(A3);
answer=find(compet(A3)==1);
disp('Network is trying to recognise the origin character...Please
wait....');

pause  %Enter any Key
%plotchar(huruf(:,answer));
figure
jawapan(answer);
```

```
Ptlow.m - Image processing system. Add noise to the character and
present the output.

[HH] = ambilpic;

JJ=input('Select the level of noise want to be recognise(0.00--
1.00):');


figure(gcf)
%gambar;

noise=huruf(:,HH)+randn(240,1)*JJ;
%plotchar(noise);
%figure

noise1(HH);
pause
A2=sim(net,noise);
A2=compet(A2);
answer=find(compet(A2) ==1);
%plotchar(huruf(:,answer));
figure
jawapan(answer);

figure(gcf)
%gambar;
```

```
%Seti.m

[fn,pn]=uigetfile('*.*','Cari Image');
imshow(fn);
A=imread(fn);
B=imresize(A,[20,12]);
C=im2bw(B)
```

%setimage.m

```
gcf
[fn,pn]=uigetfile('*.*','Cari Image');
imshow(fn);
A=imread(fn);
B=imresize(A,[20,12]);
B1=imrotate(B,45,'crop');
imshow(B1);
C=im2bw(B1)
```

```
%ambilpic.m

function [HH] = ambilpic();

L1 =[0     0     0     0     0     0     1     1     1     1     0     0;
     0     0     0     1     1     1     1     1     0     1     0     0;
     0     1     1     1     1     1     0     0     0     1     0     0;
     0     1     1     0     0     1     0     0     0     1     0     0;
     0     1     0     0     0     1     0     0     0     1     0     0;
     0     1     0     0     0     1     0     0     0     1     0     0;
     0     1     0     0     0     1     1     1     0     1     0     0;
     0     1     0     0     1     1     1     1     0     1     0     0;
     0     1     0     1     1     1     1     0     0     1     0     0;
     0     1     0     1     1     1     0     0     1     1     0     0;
     0     1     0     0     0     1     0     0     1     1     0     0;
     0     1     0     0     0     1     0     0     1     0     0     0;
     0     1     0     0     0     1     0     0     1     0     0     0;
     0     1     1     0     0     1     0     0     1     0     0     0;
     0     1     1     0     1     1     1     1     1     0     0     0;
     0     1     1     1     1     0     0     1     1     0     0     0;
     0     0     1     1     0     0     0     0     0     0     0     0;
     0     0     0     0     0     0     0     0     0     0     0     0;
     0     0     0     0     0     0     0     0     0     0     0     0;
     0     0     0     0     0     0     0     0     0     0     0     0];

L2 =[0     0     0     0     0     0     0     0     0     0     0     0;
     0     0     0     0     0     0     0     0     0     0     0     0;
     0     0     0     0     0     0     0     0     0     0     0     0;
     0     0     1     0     0     1     0     0     0     0     0     0;
     0     0     1     1     0     1     0     0     0     0     0     0;
     0     0     0     1     0     1     0     0     0     0     0     0;
     0     0     0     0     0     1     0     0     0     0     0     0;
     0     0     1     0     0     1     0     1     1     0     0     0;
     0     0     1     0     0     1     1     1     1     0     0     0;
     0     0     0     0     1     1     1     0     0     0     0     0;
     0     0     1     1     1     1     0     0     0     0     0     0;
     1     1     1     0     1     1     0     0     0     0     0     0;
     0     1     0     0     1     0     1     0     0     0     0     0;
     0     0     0     1     1     0     1     1     0     0     0     0;
     0     0     0     1     0     0     0     1     1     0     0     0;
     0     0     1     1     0     0     0     0     1     0     0     0;
     0     0     1     1     0     0     0     0     0     1     0     0;
     0     0     0     0     0     0     0     0     0     0     1     0;
     0     0     0     0     0     0     0     0     0     0     0     0;
     0     0     0     0     0     0     0     0     0     0     0     0];
```

```
L3 =[0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   0   1   0   0   0   0   0;
     0   0   0   0   0   0   1   0   0   0   0   0;
     0   0   0   0   0   0   1   0   0   0   0   0;
     0   0   0   0   0   1   1   0   0   0   0   0;
     0   0   0   0   0   1   1   0   0   0   0   0;
     0   0   0   0   0   1   1   0   0   0   0   0;
     0   0   0   0   0   1   1   1   1   0   0   0;
     0   0   0   0   0   1   1   1   1   1   0   0;
     0   0   1   1   1   1   1   0   0   0   0   0;
     0   0   1   1   1   1   1   0   0   0   0   0;
     0   0   0   0   0   1   1   0   0   0   0   0;
     0   0   0   0   0   1   1   0   0   0   0   0;
     0   0   0   0   0   1   1   0   0   0   0   0;
     0   0   0   0   0   1   1   0   0   0   0   0;
     0   0   0   0   0   1   1   0   0   0   0   0;
     0   0   0   0   0   1   1   0   0   0   0   0;
     0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   0   0   0   0   0   0   0];

L4 =[0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   0   1   1   1   0   0   0;
     0   0   0   0   1   1   1   1   1   1   0   0;
     0   0   0   1   1   1   1   0   1   1   0   0;
     0   0   1   1   1   0   0   0   1   1   0   0;
     0   0   1   1   0   0   0   0   1   1   0   0;
     0   0   1   1   0   0   0   0   1   1   0   0;
     0   0   1   1   0   0   0   0   1   1   0   0;
     0   0   1   1   0   0   0   0   1   1   0   0;
     0   0   1   1   0   0   0   0   1   1   0   0;
     0   0   1   1   0   0   0   0   1   0   0   0;
     0   0   1   1   0   0   0   0   1   0   0   0;
     0   0   1   1   0   0   0   0   1   0   0   0;
     0   0   0   1   1   1   1   1   1   0   0   0;
     0   0   0   1   1   1   1   1   0   0   0   0;
     0   0   0   0   1   1   0   0   0   0   0   0;
     0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   0   0   0   0   0   0   0];
```

```
L5 =[0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   0   1   1   0   0   0   0;
     0   0   0   0   1   1   1   1   0   0   0   0;
     0   0   0   0   1   0   1   0   0   0   0   0;
     0   0   0   0   0   0   1   0   0   0   0   0;
     0   0   0   0   0   0   1   0   0   0   0   0;
     0   0   0   0   0   0   1   0   0   0   0   0;
     0   0   0   0   0   1   1   1   0   0   0   0;
     0   0   0   1   1   1   1   0   0   0   0   0;
     0   0   0   0   1   0   1   0   0   0   0   0;
     0   0   0   0   0   0   1   0   0   0   0   0;
     0   0   0   0   0   0   1   0   0   0   0   0;
     0   0   0   0   0   1   1   1   1   1   0   0;
     0   0   0   0   1   1   1   0   1   1   0   0;
     0   0   1   1   1   0   0   0   0   0   0   0;
     0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   0   0   0   0   0   0   0];

L6=[0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   1   0   0   0   0   0   0;
     0   0   0   0   0   1   0   0   0   0   0   0;
     0   0   0   0   0   1   0   1   1   0   0   0;
     0   0   0   0   0   0   1   1   1   0   0   0;
     0   0   0   0   1   1   1   0   0   0   0   0;
     0   0   1   1   1   1   1   0   0   0   0   0;
     0   0   1   1   0   1   0   0   0   0   0   0;
     0   0   0   0   0   1   1   1   0   0   0   0;
     0   0   0   0   1   1   1   0   1   0   0   0;
     0   0   0   0   1   0   0   0   1   0   0   0;
     0   0   0   1   0   0   0   0   1   0   0   0;
     0   0   0   1   0   0   0   0   1   0   0   0;
     0   0   1   1   0   0   0   0   1   0   0   0;
     0   0   1   0   0   0   1   0   1   0   0   0;
     0   0   0   0   0   1   1   1   1   0   0   0;
     0   0   0   0   0   0   0   1   1   0   0   0;
     0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   0   0   0   0   0   0   0];
```

```
L7 =[0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   1   0   0   0   0   0   0;
     0   0   0   0   0   1   0   0   0   0   0   0;
     0   0   0   0   0   1   0   0   0   0   0   0;
     0   0   0   0   0   1   0   1   0   0   0   0;
     0   0   0   1   1   1   1   1   0   0   0   0;
     0   0   1   1   1   1   1   0   0   0   0   0;
     0   0   0   1   0   1   1   0   0   0   0   0;
     0   0   0   1   0   1   1   0   0   0   0   0;
     0   0   1   1   1   1   1   0   0   0   0   0;
     0   0   1   0   1   0   1   1   0   0   0   0;
     0   1   1   0   1   0   0   1   0   0   0   0;
     0   1   0   0   1   0   0   1   1   0   0   0;
     0   0   0   0   1   0   0   0   1   1   0   0;
     0   0   0   0   1   0   0   0   0   1   1   0;
     0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   0   0   0   0   0   0   0];

L8 =[0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   0   1   0   0   0   0   0;
     0   0   0   0   0   0   1   1   1   0   0   0;
     0   0   0   0   0   0   1   1   1   0   0   0;
     0   0   0   0   0   1   1   1   0   0   0   0;
     0   0   0   1   1   1   0   0   0   0   0   0;
     0   1   1   1   1   1   0   0   0   0   0   0;
     0   1   1   0   1   1   0   1   1   1   0   0;
     0   0   0   1   1   1   1   1   1   1   0   0;
     0   0   0   1   1   1   1   0   0   1   0   0;
     0   0   0   1   1   0   0   0   0   1   0   0;
     0   0   1   0   1   0   0   0   1   0   0   0;
     0   0   1   0   1   0   0   0   1   0   0   0;
     0   0   1   0   1   1   1   1   1   0   0   0;
     0   0   0   0   0   1   1   0   0   0   0   0;
     0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   0   0   0   0   0   0   0;
     0   0   0   0   0   0   0   0   0   0   0   0];
```

```
L9 =[0     0     0     0     0     0     0     0     0     0     0     0;
     0     0     0     0     0     0     0     0     0     0     0     0;
     0     0     0     0     1     1     0     0     0     0     0     0;
     0     0     0     1     1     0     0     0     0     0     0     0;
     0     0     0     1     0     0     0     0     0     0     0     0;
     0     0     1     1     0     1     0     0     0     0     0     0;
     0     1     1     0     0     1     0     0     0     0     0     0;
     0     1     1     0     0     1     0     1     0     0     0     0;
     0     1     1     0     0     1     1     1     0     0     0     0;
     0     1     1     1     1     1     0     0     0     0     0     0;
     0     0     1     1     0     1     0     0     0     0     0     0;
     0     0     0     0     0     1     0     1     1     0     0     0;
     0     0     0     1     0     1     0     0     1     1     0     0;
     0     0     1     1     0     1     0     0     0     1     1     0;
     0     1     1     0     1     1     1     0     0     0     1     0;
     0     0     0     0     0     0     0     0     0     0     0     0;
     0     0     0     0     0     0     0     0     0     0     0     0;
     0     0     0     0     0     0     0     0     0     0     0     0;
     0     0     0     0     0     0     0     0     0     0     0     0;
     0     0     0     0     0     0     0     0     0     0     0     0];

L10=[0     0     0     0     0     0     0     0     0     0     0     0;
     0     0     0     0     0     0     0     0     0     0     0     0;
     0     0     0     0     0     0     0     0     0     0     0     0;
     0     0     0     0     0     0     0     0     0     0     0     0;
     0     0     0     0     0     0     1     1     1     0     0     0;
     0     0     0     0     1     1     1     0     1     1     0     0;
     0     0     0     1     1     1     0     0     0     1     0     0;
     0     0     0     1     1     0     0     0     1     1     0     0;
     0     0     0     1     0     0     0     0     1     1     0     0;
     0     0     0     1     0     0     1     1     1     1     0     0;
     0     0     0     1     1     1     1     0     1     1     0     0;
     0     0     0     1     0     0     0     0     1     1     0     0;
     0     0     0     1     0     0     0     0     1     0     0     0;
     0     0     0     1     0     0     1     1     1     0     0     0;
     0     0     0     1     1     1     1     1     1     0     0     0;
     0     0     0     0     0     0     0     0     0     0     0     0;
     0     0     0     0     0     0     0     0     0     0     0     0;
     0     0     0     0     0     0     0     0     0     0     0     0;
     0     0     0     0     0     0     0     0     0     0     0     0];


[an,pn]=uigetfile('*.*','Select Image For Recognition ');

A = imread(an);
A1=imresize(A,[20,12]);
A2=im2bw(A1);
```

```
if A2 == L1
      HH = 1;
   elseif (A2==L2)
      HH = 2;
   elseif (A2==L3)
      HH = 3;
   elseif (A2==L4)
      HH = 4;
   elseif (A2==L5)
      HH = 5;
   elseif (A2==L6)
      HH = 6;
   elseif (A2==L7)
      HH = 7;
   elseif (A2==L8)
      HH = 8;
   elseif (A2==L9)
      HH = 9;
   elseif (A2==L10)
      HH = 10;
 end
```