A NEURAL NETWORK BASED CHARACTER RECOGNITION SYSTEM FOR SINHALA SCRIPT

Rohana K. Rajapakse, A. Ruvan Weerasinghe
and
E. Kevin Seneviratne
Department of Statistics and Computer Science, University of Colombo

{rkr/ruvan/eks@dscs.cmb.ac.lk}

ABSTRACT

Much effort has been extended in making a computer recognise both typed and handwritten characters automatically. Until quite recently, the focus of this endeavour has been on characters of English Language. As for Asian languages such as Sinhala and Tamil, little or no attention has been given. Methods currently widely used for character recognition for these languages are mainly those which involve pattern matching using image processing techniques. One limitation of such is their inability to respond to variations.

This paper reports the use of neural network techniques in developing for the first time, a system that can recognise hand-written Sinhala characters. In particular, it applies the widely used pattern classification neural network technique, the Backpropagation, in the recognition process. A program was developed for this purpose in C on IBM compatible personal computers to recognise basic consonant letters of the Sinhala Alphabet written on the computer screen with mouse.

1.0 Introduction

An input to a neural network usually consists of a collection of 1's and 0's arranged in an array. These arrays are called input vectors. The network is then trained to release the output patterns corresponding to these inputs. Each character is split into a number of segments (depending on the complexity of the alphabet involved) and each segment is handled by a set of purpose built neural network. The final output is unified via a lookup table.

Unless handled carefully, the various parameters involved in the architecture of the neural network may cause the training process (adjusting weights) to slow down considerably. Some of these parameters are: the number of layers, number of neurons in each layer, the initial values of weights, the training coefficient and the tolerance of the correctness. The optimal selection of parameters varies depending on the alphabet.

In order to train the weights, an initial set of weights is tested against each input vector. If an input vector is found for which the recognition fails, weights are adjusted to suit the particular input vector. However, this adjustment might also affect the recognition of other input vectors which have already been tested. So, the entire model needs to be tested all over again from the beginning.

Three-component architecture was developed to accomplish the objective of recognising Sinhala characters. These components concern the Pre-processing, Recognizing and Postprocessing tasks involved in the overall process.

The pre-processing component handles al the manipulation necessary for the preparation of the characters for forwarding to the recognition component. First, the required character or part of characters needs to be extracted from the pictorial representation. The splitting of characters into 9 segments, scaling the segments so split to a standard size and thinning the resultant character segments to obtain skeletal patterns for recognition process are also done by this component. It further involves the fitting of a grid on the segments so as to extract binary data. Since neural networks need their inputs to be in the form of binary (0's & 1's), a grid with some dimension is placed on top of the character in order to extract a binary pattern of the input character image. A 6x6 grid was found to be the optimal size of the grid as it optimises the recognition ability while taking a reasonable time for training.

The next component is that which recognises segments of the character. This is carried out by 9 neural networks each modelling a particular geometrical segment assigned to it. Modifiers in the Sinhala alphabet go through the recognition process unsplit and are handled by a separate neural network.

The post-processing component is the final stage of the entire process. Here segments split before are unified with the aid of a lookup table. Characters which are similar looking but distinct are also distinguished at this time.

Results obtained were satisfactory, especially when input characters were very close to printed letters. The network even responded positively for similar looking letters.

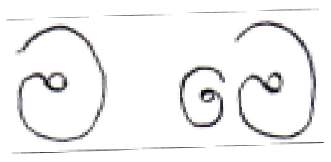
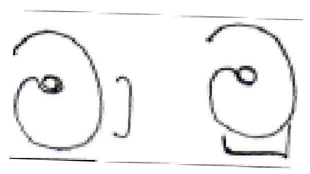
2.0 Background

2.1 Sinhala Language

The structure and feature of letters are characteristic to the language they belong. Sinhala characters are completely different from those based on Latin script and other languages for which recognition systems have been implemented.

Sinhala, the language spoken by the majority of Sri Lankans has one of the oldest systems of writing in South & East Asia. The earliest recorded Sinhala writing go back to the tird century BC, when Buddhism was introduced to the island by the Indian emperor Asoka. Sinhala writing traces its origin to the Brahmi script of ancient India. The Sinhala alphabet is also one of the largets, containing 58 letters. The letters of the Sinhala alphabet fall into two classes: vowel-letters and consonant-letters. The Sinhala character set consists of 16 vowels, 2 semi-consonants, 40 consonants and 13 consonant modifiers also known as strokes of character modifiers. These graphical signs always used in conjunction with consonants. Unlike English, consonant modifiers could be positioned at different location around the character. Although the basic shape of the characters is symmetrical and curved-shaped, some parts such as the upper or lower parts may not be in the same level.

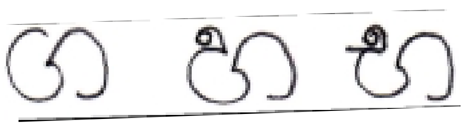
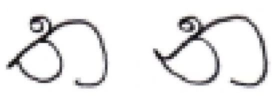
Eg.

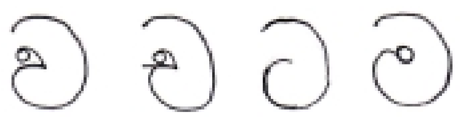
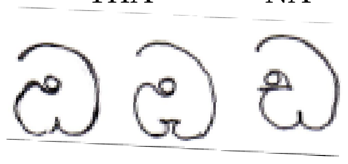
“MA” sound “ME” sound “MAA” sound “MU sound

In Sinhala, the combination of consonants and modifiers produce different phonetic sounds. These account for more than 800 different combined character sounds. This makes the recognition of Sinhala characters difficult. Because, most of the modifiers are linked to the basic character, it is difficult to isolate the basic characters when they are written in conjunction with modifiers. The fact that Sinhala characters are spread above and below the line adds to the complexity of recognition. Apart from the above, language symbol recognition also runs into difficulties due to the large size of the alphabet and the vast variations in the writing styles of different individuals. Shape discrimination between characters that look a like is also difficult for machines. Some Sinhala characters have similar shapes.

E.g.

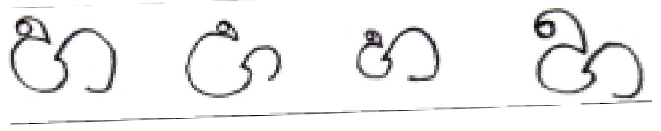
 

“GA” - “HA” - “BHA” “THA” - “NA”

“WA” - “CHA” - “TA” - “MA” ”O” - “MBA” - “DA”

In some hand-writing, the above are indistinguishable even to the human eye, and that they can only be distinguished by context. Even if only printed characters are considered, it is non-trivial for someone who is not familiar with Sinhala script to distinguish them, and this is even more so to a machine. In order to distinguish between such similar characters, the tiny differences that they have must be identified. One of the major problems of doing this for hand written characters is that they do not appear at the same relative location of the letter due to the different proportions in which characters are written by different writers of the language. Even the same person may not always write the same letter with the same proportions. This effect is shown in the following figure (for the consonant letter “HA”). Even the normalisation of the characters into a standard size does not completely eliminate this effect, though it does help to some extent.



2.2 Representation and Standards

The American Standard code for Information interchange (ASCII) is a standard used for characters of the Latin script. ASCII characters have numeric byte codes with values ranging from 00h through FFh (0 through 255 decimal). The characters are of two types. The first 128 characters, 00h through 7Fh (decimal 0 through 127) are the standard ASCII character set. Most computers handle standard characters in the same way (with the exception of the first 32 characters which are used for special purposes).

The last 128 characters, 80h through FFh (decimal 128 through 255) are special characters that make up the extended ASCII character set. Each computer manufacturer decides how to use these special characters.

The code representation of Singala characters is extended from the code table [Nandasara et. al, 1990] which was previously defined by the International Standards Organisation (ISO 646) to use only 7 bits. The last 128 characters, 80h through FFh will be used for Sinhala characters. Thus, the 8th bit will always be set when the corresponding byte represents a Sinhala character.

3.0 Neural Networks and Character Recognition

Neural networks have emerged in the last decade as a serious alternative to traditional AI techniques for implementing "intelligent systems". Their origins go way back to the 1940s, when the first mathematical model of a biological neuron was published by McCulloch and Pitts [1943]. This new paradigm gets its name "neural network" from the fact that it attempts to represent a network of interconnected neuron-like nodes. These nodes were inspired from studies of the biological nervous system. In other words, neural; networks are an attempt at creating machines that work in a similar way to the human brain by being constructed using components that behave like biological neurons. These elements are then organised in a way that may (or may not) ne related to the anatomy of the human brain. Despite this superficial resemblance, artificial neural networks in fact exhibit a surprising number of the brain's characteristics. For example, they learn from experience, generalise from previous examples to new ones, and abstract characteristics from inputs containing distorted data.

3.1 Learning

Artificial neural networks can modify their behaviour in response to their environment. This factor, more than any others, is responsible for the keen interest they have received. Shown a set of inputs (perhaps with desired outputs) they self adjust to produce consistent responses. A wide variety of training/learning algorithms have been developed, each with its own strengths and weaknesses to "teach" neural networks.

3.2 Generalisation

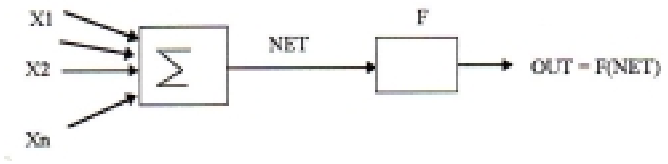
Once trained, a network's response can be, to a degree, insensitive to minor variations in its input. This ability to see through noise and distortion to the pattern that lies within is vital to pattern recognition in a real-world environment. The artificial neural network generalises automatically as a result of its own structure, and not by using human intelligence embedded in the form of ad-hoc computer programs.

3.3 Abstraction

Artificial neural networks are capable of abstracting the essence of a set of inputs. For example, a network can be trained on a sequence of distorted versions of a letter. After adequate training, application of such a distorted example will cause the network to produce a perfectly formed letter. Experimental results have revealed that training of more than 20 such distorted versions of the same letter produces correct results with a very high percentage of accuracy.

4.0 The Backpropagation Algorithm

For many years, there was no theoretically sound algorithm for training multilayer artificial neural networks. The invention of the Backpropagation algorithm has played a large part in the resurgence of interest in artificial neural networks. Backpropagation is a systematic method for training multilayer artificial neural networks (perceptrons). The following figure shows the basic model of the neuron used in backproagation networks.



Each input is multiplied by corresponding weights, analogous to a synaptic strength, and all the weighted inputs are then summed to determine the activation level of the neuron. These summed (NET) signals are further processed by an activation function (F) to produce the neuron’s output signal (OUT). In backpropagation, the function used for the activation is the logistic function or Sigmoid. This function is expressed mathematically as:

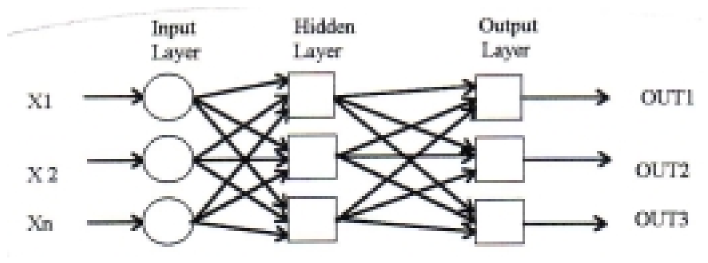
$$F(x) = \frac{1}{1+e^{-x}} \quad \text{thus } OUT = \frac{1}{1+e^{-NET}}$$

The Sigmoid compresses the range of NET so that OUT lies between zero and one. Since the backpropagation uses the derivative of the squashing function, it has to be everywhere differentiable. The Sigmoid has this property and the additional advantage of providing a form of automatic gain control (i.e. if the value of NET is large, the gain is small and if it is small the gain is large).

4.1 An overview of Training in Backpropagation

4.1.1 Training Algorithm

The objective of training the network is to adjust the weights so that the application of a set of inputs (input vectors) produces the desired outputs (output vectors). Training a backpropagation network involves each input vector being paired with a target vector representing the desired output; together they are called a training pair. The following figure shows the architecture of the multilayer backpropagation neural network.



Before starting the training process, all of the weights are initialised to small random numbers. Training the backpropagation network requires the following steps:

1. Select a training pair (next pair) from the training data set and apply the input vector to the network input
2. Calculate the output of the network, i.e. to each neuron NET=$\Sigma X_i W_i$ must be calculated and then the activation function must be applied on the result (F(NET)
3. Calculate the error between the network output and the desired putput (TARGET – OUT)
4. Adjust the weights of the network in a way that minimises the ERROR (described below)
5. Repeat step 1 through 4 for each vector in the training set until no training pair produces an ERROR larger than a pre-decided acceptance level.

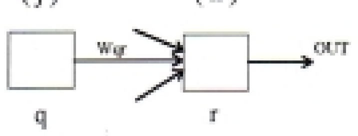
4.1.2 Adjusting Weights of the Network

Adjusting weights of the output layer

Adjusting the weights of the output layer is easier, as a target value is available for each neuron. The following shows the training process for a single weight from neuron “q” in the hidden layer “j” to neuron “r” in the output layer “k”. The output of a neuron in layer “k” is subtracted from its target values to produce an ERROR signal. This is multiplied by the derivative of a squashing function (OUT(1-OUT) calculated for that neuron (“r”) thereby producing a “δ” value.

$$\delta = \text{OUT}\,(1-\text{OUT})(\text{TARGET} - \text{OUT})$$

Then, δ value is multiplied by OUT from neuron “q” of layer “j” – the source neuron for the weight in question. This product is in turn multiplied by a training rate coefficient “η” (typically in the range 0.01 – 1.0), and the result is added to the weight.



$$\Delta_{Wqr,k} = \eta \delta_{r,k}\, \text{OUT}_{qj} \text{ ------------------------(1)}$$

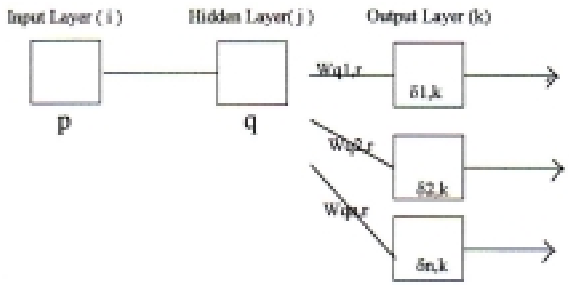
$$W_{qr,k}(n+1) = W_{qr,k}(n) + \Delta W_{qr,k} \text{ ------------(2)}$$

Where,

> $W_{qr,j}(n)$ = the value of weight from neuron in the hidden layer "j" to neuron "r" in the output layer "k" at step "n" (before adjustment)
>
> $W_{r,j}(n+1)$ = the value of weight from neuron in the hidden layer j to neuron "r" in the output layer "k" at step n+1 (after adjustment)
>
> $OUT_{q,j}$ = the value of OUT of neuron in the hidden layer "j"
>
> $\Delta W_{qr,k}$ = amount that $W_{qr,j}$ to be adjusted

Adjusting the weights of the hidden layer

Backpropagation trains the hidden layer by propagating the output ERROR back through the network layer by layer, adjusting weights at each layer. The same 2 equations (1) and (2) above are used for all layers, both ooutput & hidden except that, for hidden layers the δ values must be generated without the benefit of targets. The following figure explains how this is accomplished.



δs for hidden layer neurons are calculated according to equation (3) by using the δs calculated for output layer ($\delta_{y,k}$ s) and propagating them backward through the corresponding weights.

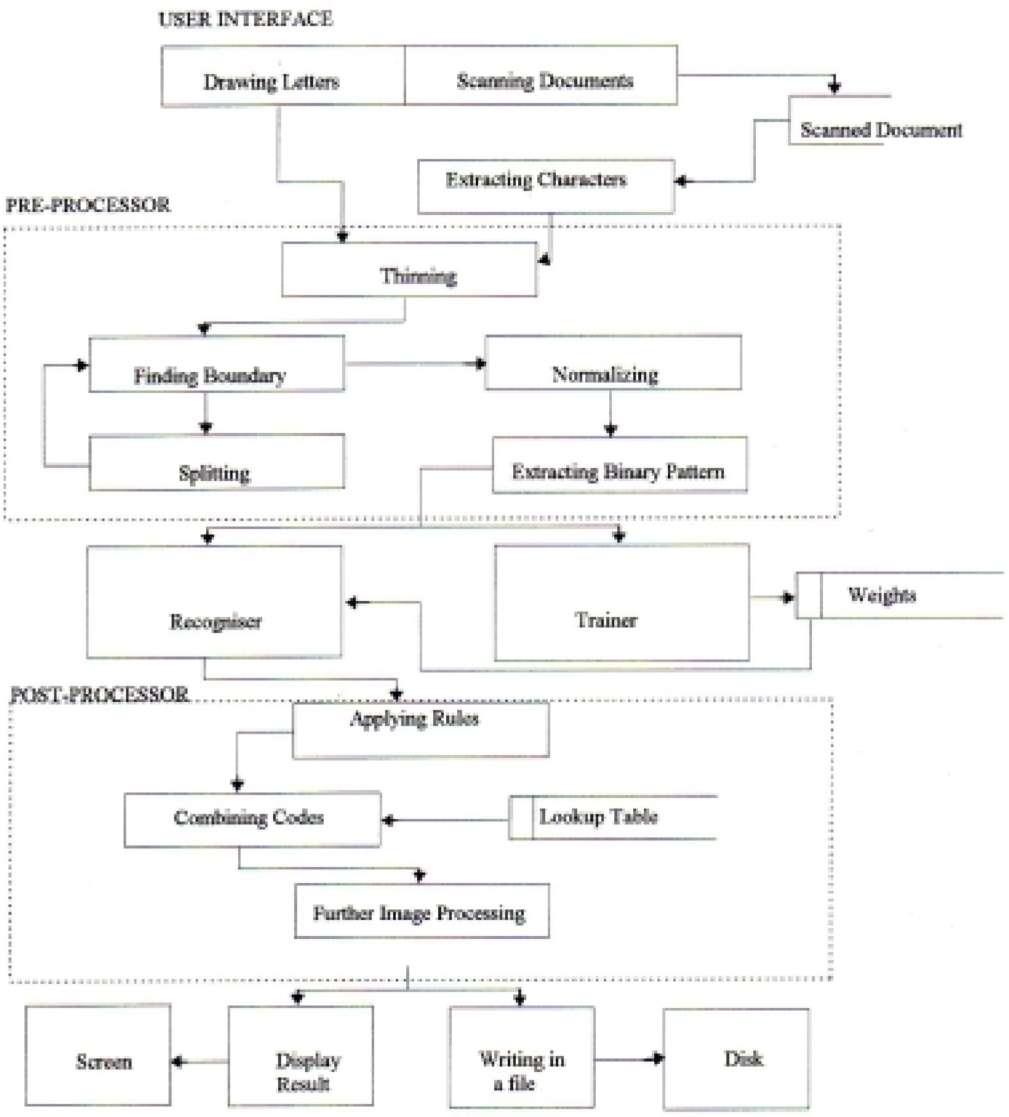
$$\delta_{q,j} = OUT_{q,j}\,(1-OUT_{q,j})\left(\sum_{r} \delta_{r,k} W_{qr,k}\right) \text{ -----------(3)}$$

Then, with δs in hand, hidden layer weights can be adjusted similar to the output layer weights as given below:

$$\Delta W_{qr,j} = \eta\delta_{qj}\, OUT_{qj}$$

$$W_{qr,j}(n+1) = W_{qr,j}(n) + \Delta W_{qr,j}$$

5.0 The Architecture of the Recognition System



The architecture of the present system consists of three major components:

1. The pre-processor
2. The Pattern Recogniser and
3. The Post-processor

The pre-processor handles all the manipulations necessary for preparing the input character before forwarding it for recognition (during pre-processing, input character is divided into 9 equal segments and directed to 9 neural networks to recognise the shapes of each segment).

The pattern recogniser analyses the input pattern of 0's and 1's and classifies it in to one of the output patterns which represents the code assigned to the shape in question.

The post-processor combines the results of each neural network to compose the character. Rules are imposed on the results of the 9 neural networks to make the recognition process more accurate [Rajapakse, 1996].

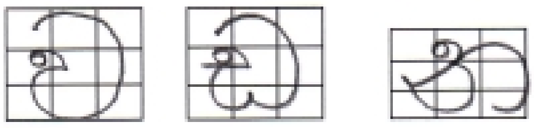
5.1 Pre-processing

Many attempts have been made to employ image processing techniques for character recognition. In almost all such attempts, image processing techniques have been used to improve the quality of the image and also to prepare the image beforehand, so that, it is suitable enough to go through the recognition process. A certain amount of such processing is needed by the algorithm used in recognition here too, while the rest is used to improve the performance of recognition.

5.1.1 Pre-processing Needed by the Recognition Algorithm

Segmentation of characters

Practical experience revealed that it is impossible to train a single neural net for a large and complicated character set such as that of Sinhala. Therefore, it was decided to split the characters in some way into a number of segments and to assign separate neural networks to recognise the shapes that belong to each such segment. An efficient and meaningful way of splitting the characters was required for this purpose. By careful analysis of the structure of Sinhala characters, it was found that many distinguishing shapes could easily be extracted, if they are split into 9 equal segments as shown below.

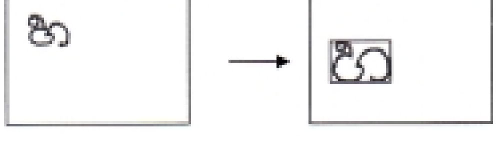


Since Sinhala letters are round in shape in general, most of them have the same common shape for a given segment. This reduces the number of different shapes that each network needed be trained to recognise. Owing to these facts, it was decided to split each character into 9 segments. This segmentation is one major function of the pre-processor.

Finding the Boundaries of Characters

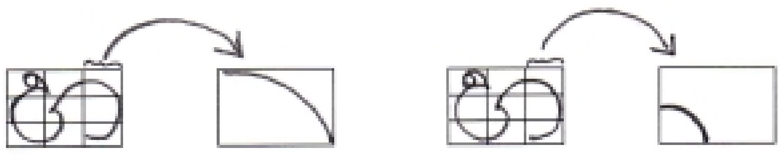
The input images may be in different sizes and could occupy different locations of the “written-line”. Therefore, boundaries of the input character must be found and the character should be normalised to a standard size before the segmentation described above.

Eg.



Further, the boundaries of each of the shapes of the separate segments should also be found and normalised, as otherwise some segments would contain tiny parts of the character, and their shapes should not be taken into consideration.

E.g.

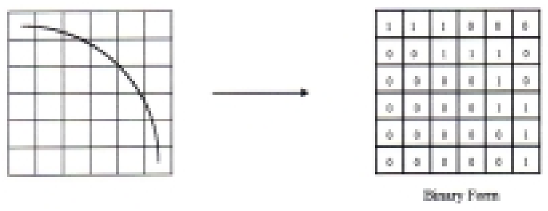


All of this work (of finding boundaries and normalising) is handled by the pre-processor. Standard image processing techniques were employed to accomplish these tasks.

Digitising Input Shapes

The backpropagation algorithm needs its input to be in binary form. Therefore, the image of each character must be converted to a binary form before forwarding it to the networks for recognition. This was achieved by placing a grid with dimension 6x6 on top of the frame in which the normalised character segment lies. This grid makes a 36 bit input pattern. If the character segment passes through any cell of the grid, then the bit that represents the cell in question is set to one, otherwise it is set to zero.

Eg.



1	1	1	0	0	0
0	0	1	1	1	0
0	0	0	0	1	0
0	0	0	0	1	1
0	0	0	0	0	1
0	0	0	0	0	1

Isolating individual characters of a word and extracting consonant letters

The final objective of this component is to scan convert a hand-written document into a soft text file. This involves isolating individual characters in a scanned image and then extracting the consonant letters and their modifiers separately. Although this part is not discusses in this paper, it is also a prerequisite of the second component and therefore needs to be performed by the pre-processor.

5.1.2 Further Processing for Better Performance at the Pre-processing stage

Although the following operations are not essential for the recognition process, the accuracy of recognition can be improved if they are performed.

1. The characters can be thinned and their skeletons obtained using well-known image processing techniques, before extracting their binary forms.

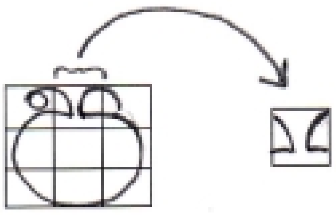
2. One of the most obvious measurable factors of a particular handwriting style is the angle between longer strokes in a word and the vertical direction referred to as the word slant. Poor handwriting styles often do not exhibit consistency in its stroke orientation, and in that sense, are not subject to valid slant measurement. However, a sizable portion of handwriting is. It will be much easier to recognise characters if they are in a standard format. Therefore, it is important to remove the slants (if any) of a given handwriting style, before trying to recognise characters.

3. The scanned documents can also be "cleaned" and "smoothed" with the help of image processing techniques for better performance.

5.2 Training Neural Networks for Character Recognision

The segmentation of input characters into 9 equal sized segments was discussed in section 5.1.1. Each such segment is assigned a separate neural network dedicated to recognise the possible shapes of that segment. All possible shapes were coded and the neural networks were trained to produce these codes. All 9 codes given by the 9 neural networks assigned to each segment are combined at the post-processing stage to identify the input character. The recognition of some characters required only a few of the 9 codes. Since codes given by all 9 segments (9 per character) are not essential for recognition and some complex shapes cause problems in training, such shapes were not trained on, and the codes given by the neural network for such shapes were ignored during training.

E.g.



5.3 Post-Processing

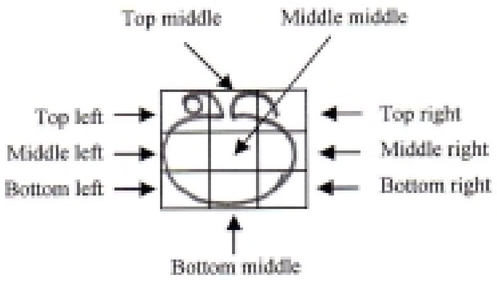
Post-processing involves processing of the output that comes from the 9 neural networks. It is also possible to use image processing techniques to process the input character images for further clarification even after the results have been produced by the neural networks. As in the case of pre-processing, there is some post-processing that are needed by the algorithm used for the recognition as well as some other that can be used for better performance.

5.3.1 Combining Results Given by the 9 Neural Networks

One of the most important processes of this component is the use of a lookup table to combine the resultant codes given by the 9 neural networks. This table is read sequentially from to bottom until the resultant codes given by the neural network match with the code sequences listed in the table. More than a single set of code sequence may be used to identify a given character.

It was found by careful analysis of the shapes of the segments of characters that only a few shapes of the segments are needed to recognise most of the characters. Therefore, a special code (50) was used in the lookup table entries to ignore matching the unnecessary codes.

E.g.



SLASCII code of the letter “PA”	Identification code of the shapes
192	1 50 1 50 50 50 1 1 1

Codes of the Top middle, Middle middle, Middle left, Middle right are not necessary to recognise this letter, and so are ignored by placing the code 50 at the corresponding places in the looup table. This ignoring of unnecessary shapes should be done with care to ensure that no such ignoring satisfies another set of codes leading to an incorrect result (misrecognition). For example, if another (say SLASCII code XXX of letter x) needs only the code number 1 at the Top right segment to recognise it, then the lookup table entry for that character is : XXX 50 50 1 50 50 50 50 50 50 50. Then during the recognition of the letter “PA”, it is misrecognised as “x” because the resultant codes for “PA” contains 1 at the 3rd position and it perfectly matches with the lookup table entry of “x”. This problem can be overcome by careful placement of lookup entries in the lookup table. For example, if the entries are placed in the lookup table in the following order so that the code for the letter “PA” appears before “x” as shown below, then the letter “PA” will not be recognised as “x”, as in the case of reversing this order.

```
192      50  1   1   50  50  50  1    1    1
XXX     50  50  1   50  50  50  50  50  50
```

5.3.2 Further Processing for Better Performance at the Post-processing Stage

Imposing Rules

A set of rules can be imposed on the resultant codes given by the neural networks to improve the accuracy of recognition. If the code for a particular segment can be determined by a set of codes of other segments, then a rule can be imposed to set that code.

6.0 Training the Neural Networks

Training is the most important and the most time consuming activity of neural network implementations. An efficient system should take the minimum training time possible. To minimise the training time, experiments should be carried out on the values of the parameters to choose a better set of values which reduces the training time. There are certain factors that affect training time and performance of the networks. Following are the parameters that could be adjusted to minimise the training time.

1. Initial values of the weights
2. Number of neurons in the hidden layer
3. Training coefficient (learning rate)
4. Tolerance
5. Grid size used to extract bit patterns from the input image

In addition to the above, the following factors may also affect training.

1. |Size of the training data set
2. Constituent characters in the training set
3. Form of the input (i.e. individual handwriting)
4. How representative the training set
5. How representative the test set for generalisation

6.1 Training Sets and Performances

6.1.1 Training Sets

The system was trained to respond to 36 of the consonant letters of the Sinhala language. Since the characters are divided into 9 segments and 9 neural networks are used to recognise the shapes belong to these 9 segments, 9 training sets, one for each segment/neural network, were needed. The training pairs of these training sets should contain shapes belong to these 9 segments and code to identify them.

It was not practical to input these shapes individually when creating training sets, because the shape of a particular segment of the actual character depends on handwriting. Therefore, this was automated so that the entire letter is input to the system, and then the shape of the segment needed is extracted from this full letter instead of drawing the shape of the segment itself.

Because of the fact that different letters may have the same shape for the same segment, the number of different shapes that any of the segment can have was less than 10. So, the least number of training pairs that any of the training sets of the corresponding neural networks require is 10. However, it was found at the early stages of the experimentation that at least 20 variations of the same shape should be included in the training set for better generalisation performance. Therefore, characters written by different people were taken to extract shapes when creating the training sets for the 9 neural nets.

In addition to these patters which were initially included in the training sets, the shapes which lead to incorrect results during recognition were also put back into corresponding training sets for subsequent training. After repeating this process for a range of different individual’s handwriting, the performance of the system could be improved to yield an acceptable level of accuracy. The final set of parameters used in all 9 neural networks is:

Training coefficient	= 0.09
Hidden layer neurons	= 20
Initial Weights	= Random [0,1]

The following table shows the number of training patterns in a sample of training sets and time taken for training (in a 486 DX2 processor PC).

Neural network	**No. of Training pairs**	**Training Time (in seconds)**
1. Top left	31	688
2. Top middle	31	823
3. Top right	42	750
4. Middle left	31	1675
5. Middle middle	32	556
6. Middle right	13	303
7. Bottom left	39	531
8. Bottom middle	56	1970
9 Bottom right	25	944

6.1.2 Performances

Three experiments were carried out to asses the performance of the system to:

i. recognise a particular individuals handwriting
ii. distinguish similar looking letters and
iii. recognise arbitrary handwriting styles

Firstly, a set of characters written by the same individual whose handwriting was trained to the system neural networks were input and the results obtained. The system could recognise 88% of the input letters correctly.

Secondly, two experiments were done to see the ability of the system to distinguish similar looking letters; one for the same individuals handwriting whish were used to train the networks and the other for different individuals handwriting which the system has never seen. A selected set of similar looking letters were chosen for these experiments and it was found that the system could recognise 68% and 60% correctly in the respective two experiments.

Finally, the system was trained with a chosen set of individual’s handwriting and tested with a collection of letters including the those whose handwriting were trained to the system and those whose handwriting were not trained. The system responded 75% correctly.

7.0 Conclusions

From the whole exercise of attempting to use neural network techniques for the recognition of characters in the Sinhala alphabet, it was discovered that a better approach could be developed employing neural network techniques together with image processing techniques. One of the major advantages of using neural networks is their inherent ability to respond to variations. This ability is important in particular where handwriting is concerned. The system responded positively for 75 of the 100 sample of input characters on average. This mark was taken by giving different individuals handwriting. It is obvious that the system will respond with near perfect accuracy for printed letters. Also, the system showed the ability to respond to different styles of handwriting, and the ability to distinguish similar characters effectively. Through this endeavour, the following conclusions/observations could be made:

a. The backpropagation algorithm can be used as a means of recognising Sinhala characters.

b. The segmentation method used to partition the letters into 9 segments was of great benefit in grouping the common features of the Sinhala characters and extracting important features to help distinguish similar characters.

c. The system has shown a significant accuracy for the recognition of a given individual's handwriting when only his handwriting was trained to the system.

d. Given a training set which represents the variety of possible characters suitably well, a relatively good performance can be obtained.

e. The system has the ability to distinguish between similar characters quite well.

REFERENCES

Aleksander, Igore & Morton, Helen (1991) : "An Introduction to Neural Computing", Chapman & Hal, ISBN 0 412 37780 2

Beale, R. and Jakson, T. (1990) : "Neural Computing – An Introduction", IOP publishing Ltd. ISBN 0 852774 262 2

Disanayake, J. B. (1993) : "Lets Read and Write Sinhala", Pioneer Lanka

Nandasara, S. T., Disanayake, J. B., Samaranayake, V. K., Seneviratne, E. K.,, and Koannantakool, T. (1990) : Draft Standards for the use of Sinhala in Computer Technology" by the Computer & Information Council of Sri Lanka (CINTEC)

Rajapakse, R. K. (1996) : A Neural Network based Character Recognition System for Sinhala Script", MSc dissertation, Department of Statistics and Computer Science, University of Colombo.

Valluru, R. and Hayagriva, R. (1996) : C++ Neural Networks and Fuzzy Logic, BPB Publications

This paper was presented at the South East Asian Regional Computer Confederation, Conference and Cyberexhibition (SEARCC'96) held in Bangkok , Thailand , July 4 - 7th 1996, and appeared in the conference proceedings.