Design and Development of Artificial Neural Network Based Tamil Unicode Symbols Identification System

A.B.Karthick Anand Babu[1]

[1] **Assistant Professor, Department of Software Engineering , Periyar Maniammai University, Vallam,Thanjavur, Tamilnadu,India**

Abstract

Design and Development of Unicode and its recognition especially for Indian script is an active area of research today. An attempt is made to identify Tamil- a vernacular of southern India, which is also the official language of Tamilnadu. Tamil language present great challenges to an OCR designer due to the large number (247 letters) in the alphabet, the sophisticated ways in which they combine, and the complicated graphemes they result in. The conventional programming methods of mapping symbol images into matrices, analyzing pixel and/or vector data and trying to decide which symbol corresponds to which character would yield little or no realistic results. Clearly the needed methodology will be one that can detect closeness of graphic representations to known symbols based on the character height, character width, the number of horizontal lines (long and short), the number of vertical lines (long and short), number of slope lines, special dots and based on that the glyphs are now set ready for classification. The extracted features are passed to neural network where the characters are classified by supervised learning of Back Propagation algorithm which compromises training, calculation of error, and modifying weights and then testing the given image.and make decisions based on this nearness. This proposed work has employed the MLP technique to identify the symbols, excellent results were obtained for a number of widely used Unicode Tamil font types.

Keywords:
Artifical Neural Network , MLP , Unicode , Weights

1. Introduction

1.1. Artificial Neural Networks

Artificial Neural networks have seen an explosion of interest over the last few years, and are being successfully applied across an extraordinary range of problem domains, in areas as diverse as finance, medicine, engineering, geology and physics. Indeed, anywhere that there are problems of prediction, classification or control, neural networks are being introduced. To capture the essence of biological neural systems, an artificial *neuron* is defined as follows:

- It receives a number of inputs (either from original data, or from the output of other neurons in the neural network). Each input comes via a connection that has a strength (or *weight*); these weights correspond to synaptic efficacy in a biological neuron. Each neuron also has a single threshold value. The weighted sum of the inputs is formed, and the threshold subtracted, to compose the *activation* of the neuron.
- The activation signal is passed through an activation function (also known as a transfer function) to produce the output of the neuron.

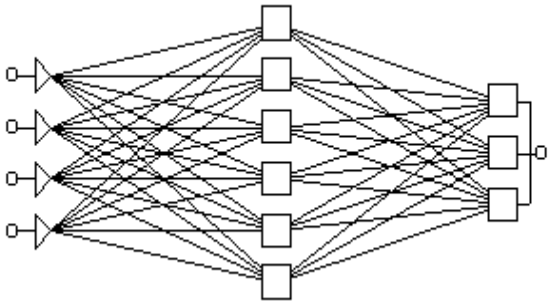

Fig.1 A Typical Feedforward Network

A typical feedforward network has neurons arranged in a distinct layered topology. The input layer is not really neural at all: these units simply serve to introduce the values of the input variables. The hidden and output layer neurons are each connected to all of the units in the preceding layer. Again, it is possible to define networks that are partially-connected to only some units in the preceding layer; however, for most applications fully-connected networks are better.

1.2. The Multi-Layer Perceptron Neural Network Model

The Multi-Layer Perceptron Neural Network is perhaps the most popular network architecture in use today. The units each perform a biased weighted sum of their inputs and pass this activation level through an activation 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 Multilayer Perceptrons (MLP) design include specification of the number of hidden layers and the number of units in each layer. Most common activation functions are the logistic and hyperbolic tangent sigmoid functions. This work uses the **hyperbolic tangent function:** $f(x) = \frac{2}{(1+e^{-x})} - 1$ (1)

and derivative: $f'(x) = f(x)(1 - f(x))$. (2)

1.3. Optical Language - Tamil Symbols

The Tamil script is written from left to right is characterized by having its own written symbolic representations, it has twelve vowels, eighteen consonants and one character, the āytam, which is classified in Tamil grammar as being neither a consonant nor a vowel. The script, however, is syllabic and not alphabetic. The complete script, therefore, consists of the thirty-one letters in their independent form, and an additional 216 combinant letters representing a total 247 combinations of a consonant and a vowel, a mute consonant, or a vowel alone. These combinant letters are formed by adding a vowel marker to the consonant. Some vowels require the basic shape of the consonant to be altered in a way that is specific to that vowel. Others are written by adding a vowel-specific suffix to the consonant, yet others a prefix, and finally some vowels require adding both a prefix and a suffix to the consonant. In every case the vowel marker is different from the standalone character for the vowel. Like other South Asian scripts in Unicode, the Tamil encoding was originally derived from the ISCII standard. Both ISCII and Unicode encode Tamil as an abugida. In an abugida, each basic character represents a consonant and default vowel. Consonants with a different vowel or bare consonants are represented by adding a modifier character to a base character. Each codepoint representing a similar phoneme is encoded in the same relative position in each South Asian script block in Unicode, including Tamil. Although Unicode represents Tamil as an abugida all the pure consonants (consonants with no associated vowel) and syllables in Tamil can be represented by combining multiple Unicode code points.

2. Technical Overview

2.1. Introduction

The operations of the network implementation in this project can be summarized by the following steps:

Training phase

- Analyze image for characters
- Convert symbols to pixel matrices
- Retrieve corresponding desired output character and convert to Unicode
- Lineraize matrix and feed to network
- Compute output , Compare output with desired output Unicode value and compute error
- Adjust weights accordingly and repeat process until preset number of iterations

Testing phase

- Analyze image for characters
- Convert symbols to pixel matrices
- Compute output
- Display character representation of the Unicode output

Essential components of the implementation are:

- Formation of the network and weight initialization routine
- Pixel analysis of images for symbol detection
- Loading routines for training input images and corresponding desired output characters in special files
- Loading and saving routines for trained network (weight values)
- Character to binary Unicode and vice versa conversion routines
- Error, output and weight calculation routines

2.2. Network Formation

The MLP Network implemented for the purpose of this project is composed of 3 layers, one input, one hidden and one output. The input layer constitutes of 150 neurons which receive pixel binary data from a 10x15 symbol pixel matrix. The size of this matrix was decided taking into consideration the average height and width of character image that can be mapped without introducing any significant pixel noise. The hidden layer constitutes of 250 neurons whose number is decided on the basis of optimal results on a trial and error basis.

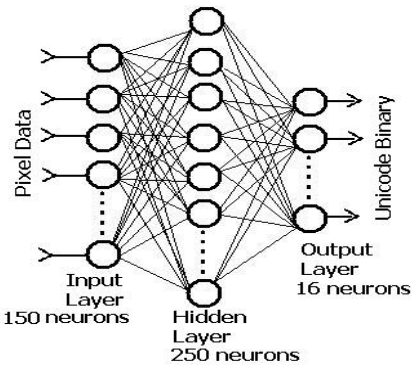

Fig. 2 The Project MLP Network

The output layer is composed of 16 neurons corresponding to the 16-bits of Unicode encoding. To initialize the weights a random function was used to assign an initial random number which lies between two preset integers named **weight_bias**. The weight bias is selected from trial and error observation to correspond to average weights for quick convergence.

2.3. Symbol image detection

The Process of Character /symbol Recognition of the document image mainly involves following phases:

- Acquisition and Digitization/Binarization of Grayscale Image
- Thinning and Edge Detection
- Feature Extraction
- Feed Forward Artificial Neural Network based Matching.
- Recognition of Character based on matching score.

A . Detection

The process of image analysis to detect character symbols by examining pixels is the core part of input set preparation in both the training and testing phase. Symbolic extents are recognized out of an input image file based on the color value of individual pixels, which for the limits of this project is assumed to be either black **RGB(255,0,0,0)** or white **RGB(255,255,255,255)**. The input images are assumed to be in bitmap form of any resolution which can be mapped to an internal bitmap. The procedure also assumes the input image is composed of only characters and any other type of bounding object like a border line is not taken into consideration. The procedure for analyzing images to detect characters is listed in the following algorithms:

i. Determining character lines

Enumeration of character lines in a character image is essential in delimiting the bounds within which the detection can proceed. Thus detecting the next character in an image does not necessarily involve scanning the whole image all over again.

Algorithm:

1. start at the first x and first y pixel of the image and lines to 0
2. scan up to the width of the image on the same y-component of the image
3. start at the top of the line found and first x-component pixel
4. scan up to the width of the image on the same y-component of the image
5. start below the bottom of the last line found and repeat steps 1-4 to detect subsequent lines
6. If bottom of image (image height) is reached stop.

ii. Detecting Individual symbols

Detection of individual symbols involves scanning character lines for orthogonally separable images composed of black pixels.

Algorithm:

1. start at the first character line top and first x-component
2. scan up to image width on the same y-component
3. start at the top of the character found and first x-component, pixel(0,character_top)
4. scan up to the line bottom on the same x-component
5. start at the left of the symbol found and top of the current line, pixel(character_left, line_top)
6. scan up to the width of the image on the same x-component
7. start at the bottom of the current line and left of the symbol, pixel(character_left,line_bottom)
8. scan up to the right of the character on the same y-component

Fig 3. Line and Character boundary detection

From the procedure followed and the above figure it is obvious that the detected character bound might not be the actual bound for the character in question. This is an issue that arises with the height and bottom alignment irregularity that exists with printed unicode symbols. Thus a line top does not necessarily mean top of all characters and a line bottom might not mean bottom of all characters as well. Hence a confirmation of top and bottom for the character is needed. An optional confirmation algorithm implemented in the project is:

A. start at the top of the current line and left of the character
B. scan up to the right of the character
    1. if a black pixels is detected register y as the confirmed top
    2. if not continue to the next pixel
    3. if no black pixels are found increment y and reset x to scan the next horizontal line

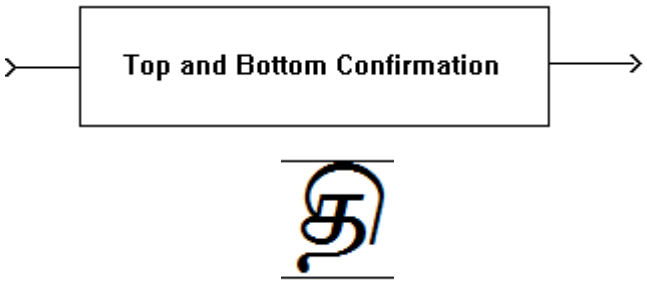

Fig 4. Confirmation of Character boundaries

iii. Symbol Image Matrix Mapping

The next step is to map the symbol image into a corresponding two dimensional binary matrix. An important issue to consider here will be deciding the size of the matrix. If all the pixels of the symbol are mapped into the matrix, one would definitely be able to acquire all the distinguishing pixel features of the symbol and minimize overlap with other symbols. However this strategy would imply maintaining and processing a very large matrix (up to 1500 elements for a 100x150 pixel image). Hence a reasonable tradeoff is needed in order to minimize processing time which will not significantly affect the separability of the patterns. The project employed a sampling strategy which would map the symbol image into a 10x15 binary matrix with only 150 elements. Since the height and width of individual images vary, an adaptive sampling algorithm was implemented.

B. Training

Once the network has been initialized and the training input space prepared the network is ready to be trained. Some issues that need to be addressed upon training the network are:

- How chaotic is the input space? A chaotic input varies randomly and in extreme range without any predictable flow among its members.
- How complex are the patterns for which we train the network? Complex patterns are usually characterized by feature overlap and high data size.
- What should be used for the values of:
  - Learning rate
  - Sigmoid slope
  - Weight bias
- How many Iterations are needed to train the network for a given number of input sets?
- What error threshold value must be used to compare against in order to prematurely stop iterations if the need arises?

The complexity of the individual pattern data is also another issue in character recognition. Each symbol has a large number of distinct features that need to be accounted for in order to correctly recognize it. Elimination of some features might result in pattern overlap and the minimum amount of data required makes it one of the most complex classes of input space in pattern recognition. Other than the known issues mentioned, the other numeric parameters of the network are determined in real time. They also vary greatly from one implementation to another according to the number of input symbols fed and the network topology.

For the purpose of this project the parameters use are:

- Learning rate = 170
- Sigmoid Slope = 0.017
- Weight bias = 25 (determined by trial and error)
- Number of Epochs = 200-700 (depends on font)
- Mean error threshold value = 0.0003 (determined by trial and error)

Algorithm:

The training routine implemented the following basic algorithm

1. Form network according to the specified topology parameters
2. Initialize weights with random values within the specified weight_bias value
3. load trainer set files (both input image and desired output text)
4. analyze input image and map all detected symbols into linear arrays
5. read desired output text from file and convert each character to a binary Unicode value to store separately
6. for each character :
   a. calculate the output of the feed forward network
   b. compare with the desired output corresponding to the symbol and compute error
   c. back propagate error across each link to adjust the weights
7. move to the next character and repeat step 6 until all characters are visited
8. compute the average error of all characters
9. repeat steps 6 and 8 until the specified number of epochs
   a. Is error threshold reached? If so abort iteration
   b. If not continue iteration

C. Testing

The testing phase of the implementation is simple and straightforward. Since the program is coded into modular parts the same routines that were used to load, analyze and compute network parameters of input vectors in the training phase can be reused in the testing phase as well. The basic steps in testing input images for characters can be summarized as follows:

Algorithm:

- load image file
- analyze image for character lines
- for each character line detect consecutive character symbols
  - analyze and process symbol image to map into an input vector
  - feed input vector to network and compute output
  - convert the Unicode binary output to the corresponding character and render to a text box

3 Results and Discussion

The network has been trained and tested for a number of widely used font type in Tamil Font style. The necessary steps are preparing the sequence of input symbol images in a single image file (*.bmp [bitmap] extension), typing the corresponding characters in a text file (*.cts [character trainer set] extension) and saving the two in the same folder (both must have the same file name except for their extensions). The application will provide a file opener dialog for the user to locate the *.cts text file and will load the corresponding image file by itself. Although the results listed in the subsequent tables are from a training/testing process of symbol images created with a 72pt. font size the use of any other size is also straight forward by preparing the input/desired output set as explained. The application can be operated with symbol images as small as 8pt font size.

A. Results for variation in number of Epochs
Number of characters=90, Learning rate=150, Sigmoid slope=0.014

Font Type	**300**		**600**		**800**	
	№ of wrong characters	% Error	№ of wrong characters	% Error	№ of wrong characters	% Error
Vijaya	4	4.44	3	3.33	1	1.11
Latha	1	1.11	0	0	0	0

Table 1

B. Results for variation in number of Input characters
Number of Epochs=100, Learning rate=150, Sigmoid slope=0.014

Font Type	**20**		**50**		**90**	
	№ of wrong characters	% Error	№ of wrong characters	% Error	№ of wrong characters	% Error
Vijaya	0	0	6	12	11	12.22
Latha	0	0	3	6	8	8.89

Table 2

C. Results for variation in Learning rate parameter
Number of characters=90, Number of Epochs=600, Sigmoid slope=0.014

Font Type	**50**		**100**		**120**	
	№ of wrong characters	% Error	№ of wrong characters	% Error	№ of wrong characters	% Error
Vijaya	82	91.11	18	20	3	3.33
Latha	56	62.22	11	12.22	1	1.11

Table 3

4. Performance Observation

1. Influence of parameter variation

i. Increasing the number of iterations has generally a positive proportionality relation to the performance of the network. However in certain cases further increasing the number of epochs has an adverse effect of introducing more number of wrong recognitions. This partially can be attributed to the high value of learning rate parameter as the network approaches its optimal limits and further weight updates result in bypassing the optimal state. With further iterations the network will try to swing back to the desired state and back again continuously, with a good chance of missing the optimal state at the final epoch. This phenomenon is known as over learning.

ii. The size of the input states is also another direct factor influencing the performance. It is natural that the more number of input symbol set the network is required to be trained for the more it is susceptible for error. Usually the complex and large sized input sets require a large topology network with more number of iterations. For the above maximum set number of 90 symbols the optimal topology reached was one hidden layer of 250 neurons.

iii. Learning rate parameter variation also affects the network performance for a given limit of iterations. The less the value of this parameter, the lower the value with which the network updates its weights. This intuitively implies that it will be less likely to face the over learning difficulty discussed above since it will be updating its links slowly and in a more refined manner. But unfortunately this would also imply more number of iterations is required to reach its optimal state. Thus a trade of is needed in order to optimize the overall network performance. The optimal value decided upon for the learning parameter is 150.

5. Reference

1. **Artificial Intelligence and cognitive science** 2006, Nils J. Nilsson Stanford AI Lab
2. **Off-line Handwriting Recognition Using Artificial Neural Networks** 2000, Andrew T. Wilson University of Minnesota, Morris
3. **Using Neural Networks to Create an Adaptive Character Recognition System** 2002, Alexander J. Faaborg Cornell University, Ithaca NY

4. **Hand-Printed Character Recognizer using Neural Network**
2000, Shahzad Malik
5. **Neural Networks and Fuzzy Logic**. 1995, Rao, V., Rao, H.MIS Press, New York
6. **"Character Recognition by Neural Network,"** in Proc. IEEE International Conference on Automatic Face and Gesture Recognition, 2000, pp. 196–201 G. Guodong, S. Li, and C. Kapluk
7. **"A Devnagari OCR and A Brief Overview of OCR for Indian Script"**, *PROC Symposium on Transaction support System (STRANS 2001)*, Feb. 15-17, 2001, Kanpur, India Veena Bansal and R.M.K. Sinha, "A Devnagari OCR and A Brief Overview of OCR for Indian Script", *PROC Symposium on Transaction support System (STRANS 2001)*,Feb. 15-17, 2001, Kanpur, India.
8. http://en.wikipedia.org/wiki/Tamil_alphabet

9. ABOUT THE AUTHOR

10.

A.B.Karthick Anand Babu working as Assistant Professor in the department of Software Engineerin, Periyar Maniammai University,Vallam,Thanjavur. He is currently working in the area of effective teaching and easy learning methodology.