An Evolutionary Neural Learning Algorithm for Offline Cursive Handwriting Words with Hamming Network Lexicon

Moumita Ghosh, Ranadhir Ghosh, John Yearwood

School of Information Technology and Mathematical Sciences
University of Ballarat, Victoria, Australia
{m.ghosh, r.ghosh, j.yearwood} @ballarat.edu.au

Abstract

In this paper we incorporate a hybrid evolutionary method, which uses a combination of genetic algorithm and matrix based solution method such as QR factorization. A heuristic segmentation algorithm is initially used to over segment each word. Then the segmentation points are passed through the rule-based module to discard the incorrect segmentation points and include any missing segmentation points. Following the segmentation the connected contour is extracted between two correct segmentation points. The contour is passed through the feature extraction module that extracts the angular features of the contour, after which the EALS-BT algorithm finds the architecture and the weights for the classifier network. These recognized characters are grouped into words and passed to a variable length lexicon that retrieves words that has highest confidence value. Hamming neural network is used as a lexicon that rectifies the word misrecognized by the classifier. We have used CEDAR benchmark dataset and UCI Machine Learning repository (Upper case) to test the train and test the system

Introduction

Different approaches have been utilized for segmentation and recognition in handwriting recognition tasks. Several have used ANNs to segment the cursive words (Eastwoord et al., 1997), (Blumenstein & Verma, 1999), (Blumenstein & Jones, 1999). Segmentation plays important roles in the overall process of handwriting recognition (Lu & Shridhar, 1999). Cursive word segmentation deserves particular attention since it has been acknowledged as the most difficult of all handwriting problems. In this proposed word recognition system, rule based segmentation methods are used for handwritten words. Following segmentation, a contour between two consecutive segmentation points is extracted. From this contour structural features are extracted after which the EALS-BT algorithm finds the architecture and the weights for a neural network classifier. The recognized characters are grouped together into words and passed to a variable length lexicon that retrieves words that have the highest confidence value. An overview of the recognition system is shown in Figure 1.

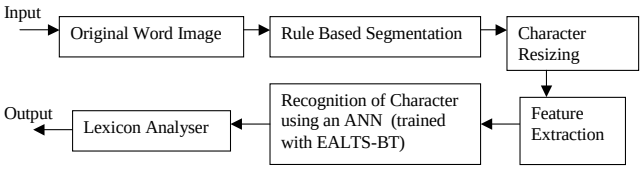

Figure 1: Handwriting recognition System

Methodology

The research methodology is briefly explained in this section.

Segmentation

The segmentation follows the following steps:
Step1: Compute Baselines.
Step2: Over-segment the word.
Step 3: Pass the segmentation points through the rule base to detect the incorrect segmentation point.
Step 4: Output the correct segmentation points.

Baseline Computation

Baseline computation is an important technique in handwriting recognition. Baselines are used for size normalization, correcting rotation, extracting features etc. In this approach we compute five lines for the segmentation task. These are the: upper baseline, lower baseline, middle baseline, ascender line, and descender line. All the baselines are computed with respect to the horizontal pixel density. The upper baseline is the line that goes through the top of the lower case characters. The lower baseline is the line that goes through the bottom of the lower case characters. The middle baseline corresponds to the writing line on which the word was written. The ascender line corresponds to the line that passes through the topmost point of the word.

Over Segmentation

This module is used to assign Candidate Segmentation Points (CSP) that could be validated through the rule-based system for further processing. A heuristic over segmentation algorithm is used that incorporates the vertical histogram change. A vertical histogram is drawn at each column point and the change in vertical density is noted. Where the change is drastic, a possible Candidate segmentation point is drawn.

Rule Based Validation

The over-segmented word is passed through the rule base where rules are written on the basis of contour characteristics of a character (such as a loop, a hat shape etc.) described below. Application of the rules leads to the removal of segmentation points from the character.

Rule 1. If a loop (closed area) is detected, remove the segmentation points within a loop. Add a segmentation point after the end of the loop as a Candidate Segmentation Point (CSP).

Rule 2. If the hat shape is detected remove the segmentation point within the hat shape contour. Add an extra segmentation point after the end point of the hat shape. The hat shape is described as '$\wedge$' or '$\vee$'.

Rule 3. Add missing segmentation points. The missing points are detected by comparing the distance between two segmentation points to a threshold. The average distance between two segmentation points (threshold) is calculated by taking the average of all segmentation points. If the distances cross the threshold value a Candidate Segmentation Point is added as a missing one.

Rule 4. Delete a few irrelevant segmentation points. The irrelevant points are detected by comparison with the average width between two segmentation points. The average distance between two segmentation points is calculated by taking the average of all segmentation points. If the distance is less than the average width a segmentation point is irrelevant and one deleted.

Contour Extraction

The contour between two consecutive segmentation points is extracted as follows. In the first step disconnect the pixels near the first segmentation point; disconnect the pixels near the second segmentation point. Find the smallest distance of the first black pixel from the first segmentation point and the three baselines. Follow the contour path across that baseline having minimum distance. Find the connecting contour. Mark it as visited once it is visited. If the contour is already visited then discard that, take the other part if any.

Resizing

The individual extracted contours are of varying size and hence in need of size normalization for use with the neural network. The contours are passed through a resizing algorithm that adjusts each contour to a normalized size.

Feature Extraction

A novel feature extraction technique is used to extract the features of the extracted contour. The features extracted in this methodology are structural features.

Slope. The slope of consecutive points is calculated. The rate of change of slope is used as the main feature. The input to the feature extraction module is the set of coordinates (x, y) of the contour extracted from the contour extraction phase. Slope (θ) between two coordinate (x_1, y_1) and (x_2, y_2) is as follows

If $(x_2 = x_1)$ then

$$\theta = 0$$

else

$$\theta = \tan^{-1}\left(\frac{y_2 - y_1}{x_2 - x_1}\right)$$

Direction Change (Up/Down). The point with respect to the main body of the contour where the direction is changing is also taken care of. The change of direction is classified by whether the contour is changing direction upwards to downward or vice versa.

EALS-BT Classifier Model

A hybrid evolutionary technique is used to find the neural network architecture and weights. The details of this algorithm are described in Ghosh [2003]. The training of the model is based on a hierarchical structure, which is evolved with its own architecture and weights. A method called 'Evolutionary Algorithm and Least Squares' is used to find the weights and a type of Binary Search is used for the architecture (EALS-BT). In Figure 2 the flowchart for the dynamics of the combination methodology for searching the architecture and calculating the weights is given. The two separate modules for the architecture and the weights are referred to as the findArchitecture and the findWeight modules respectively.

Architecture Details. A two-layer network architecture is considered. The input nodes for the ANN do the range compression for the input and transmit output to all the nodes in the hidden layer. The activation function used for all the nodes in hidden and output layer is sigmoidal. The first layer is composed of input nodes, which simply range compress the applied input (based on pre-specified range limits) so that it is in the open range interval (0, 1) and fan out the result to all nodes in the second layer. The hidden

nodes perform a weighted sum on its input, and then pass that through the sigmoidal activation function before sending the result to the next layer. The output layers also perform the same weighted sum operation on its input and then pass that through the sigmoidal activation function to give the final result.

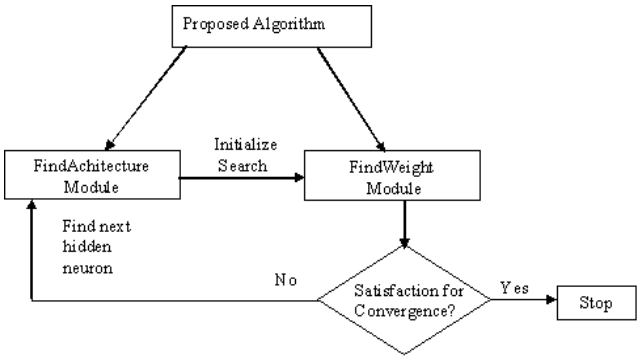

Figure 2 Hierarchical structure for weight and architecture module

FindWeight module. The weight variables for each layer are found using a hybrid method, which uses the genetic algorithm (GA) and a least square method. The architecture is shown in Figure 3. The genetic algorithm is applied for the first layer weight and the least square method is applied to find the weights for the output layer.

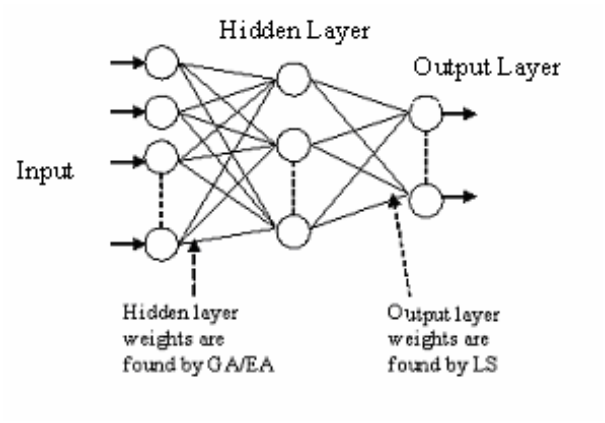

Figure 3: A two layer ANN architecture for the proposed hybrid learning method

We initialize the hidden layer weights with a uniform distribution with closed range interval [-1, +1]. How we can combine the evolutionary algorithm with the least square method (EALS) is again a very important issue as there are many possibilities joining the two independent modules. The LS method is called after the convergence of the evolutionary algorithm (EA) is over. After certain number of generations for the EA, the best fitness population is halved and the lower half is used as the weights for the hidden layer and those weights are used for the LS method. The Stopping criterion for EALS is also based on a few simple rules. All the rules are based on the current train and test output and the maximum number of generations for the evolution algorithm. In the following, we describe the stopping criterion for the convergence of the evolutionary algorithm.

If (best_RMS_error[1] < goal_RMS_error) then Stop

Else if (number_of_generation = total_number_of_generation[2]) then Stop

Else if (train_classification_error is increased in #m[3] consecutive generation) then Stop

Else continue

FindArchitecture module. We use a binary tree search type to find the optimal number of hidden neuron. The pseudo-code of the algorithm is given below:

Step 1: Find the percentage test classification error & train_classification_error (error_min) for the minimum number of hidden neurons, where error_min = (train_classification_error (%)+ test_classification_error (%)) / 2

Step 2: Find the percentage test classification error & train classification error (error_max) for the maximum number of hidden neurons, where error_max = (train_classification_error (%) + test_classification_error (%)) / 2

Step 3: Find the percentage test classification error & train classification error (error_mid) for the middle (mid = (min + max) / 2) number of hidden neurons, where error_mid = (train_classification_error (%) + test_classification_error (%)) / 2

Step 4: If (error_mid < error_min) and (error_mid > error_max) then
min = mid
mid = (min + max / 2)
else
max = mid
mid = (min + max / 2)
end if

Step 5: Go to Step1, if (mid > min) and (mid < max)
Else go to Step 6

[1] The best_RMS_error is the best of the RMS error from the population pool
[2] Total number of generations is considered as 30
[3] m is considered as 3

Step 6: Number of hidden neurons = mid

A simple rule base can describe the working of the stopping criterion for the combined algorithm.
Rule 1: If the current output is satisfactory, then stop the algorithm, else check rule 2.
Rule 2: If the stopping criterion for the weight search is met and the search is completely exhausted (in terms of the number of iterations) then stop, else check rule 3.
Rule 3: If the stopping criterion for the weight search is met then go to rule 4, else go to the next generation for the EALS.
Rule 4: If the stopping criterion for EALS is met then go to rule 1, else initialize for the next number of hidden neurons for EALS.

Lexicon Analyzer

A neural network based dictionary was used for recognizing words following the recognition of individual characters. The normalized ASCII value for each character is taken as the feature for the Neural network. The number of words in the dictionary represents the number of output of the Neural network. The hamming neural network is trained with the words in the dictionary. Hamming distance can be defined as being the number of bits in the input that do not match corresponding bits encoded in the weights of the network. The hamming network calculates the Hamming distance to the exemplar of each class and selects the class with the minimum hamming distance. When analyzing, the network fires the output neuron that matches the word in the dictionary. Figure 4 shows a representation of the lexicon analyzer.

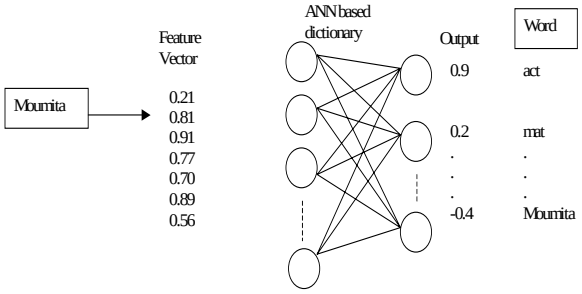

Figure 4: ANN based lexicon analyzer

Experimental Results

A number of experiments were conducted. Samples of handwritten words from the CEDAR benchmark dataset were used to test the segmentation module. The character dataset of the CEDAR and UCI Machine Learning repository (Upper case) were also used to train and test the neural network classifier. All algorithms were implemented in C++ on a UNIX platform. The number of characters used for training and testing respectively were 7000 and 2000 for CEDAR dataset. For UCI Machine Learning repository handwriting dataset 6000 and 1500 were used respectively for training and testing. The number of outputs was 26 representing uppercase characters (A-Z) and 26 representing lowercase characters (a-z).

Segmentation Results

To test the accuracy of the rule based novel segmentation algorithm three criteria were used for the segmentation results. These are the number of: 1) over segmentations, 2) missed segmentations and 3) bad segmentations. Over segmentation occurs when the character is segmented by more than two segmentation lines. Missed segmentation occurs when a correct segmentation point is not noted by the segmentation algorithm. Bad segmentation occurs when the segmentation point does not separate two characters properly. The error rates are shown in Table 1.

Table 1: Segmentation Results

Over segmentation (%)	Missed (%)	Bad (%)
20.02	0.2	8.7

As shown in Table 1, the segmentation algorithm performed reasonably well. The missed error rate was almost zero (0.2%). The over segmentation error was prominent but not excessive (20.02%). The bad segmentation error obtained was also modest (8.7%).

Character Recognition Results

The character recognition results obtained for CEDAR benchmark dataset are shown in Table 2. The character recognition results obtained for UCI Machine Learning repository dataset are shown in Table 3.The result is compared with the traditional Back Propagation algorithm in both the cases.

Table 2: Character Recognition Results for CEDAR

Lower/ Upper case	Dataset	Recognition Rate [%]	
		Back propagation	EALS-BT
Lower	Training	100	99.8
Lower	Testing	87.3	92.4
Upper	Training	99.7	100
Upper	Testing	86.4	95.6

Table 3: Character Recognition Results for UCI Machine Learning

Dataset	Recognition Rate [%]	
	Back propagation	EALS-BT
Training	100	99
Testing	95	96.8

Word Recognition Results

The word recognition result is shown in Table 3. The words are passed through the lexicon analyzer. The word recognition result we got before passing through the lexicon analyzer was 75%. The recognition rate after passing through the lexicon was 96%.

Table 3: Word Recognition Results

Length of lexicon	Word Recognition Result [%]	
	Before passing through the Lexicon analyzer	After passing through the Lexicon analyzer
50	75	96
100	75	95

Analysis and Discussion

The experiment was started with two neural classifiers for lower case and uppercase. The upper-case characters were giving higher classification results than the lower case. This is due to the increasing ambiguity of lower case characters. The shapes of the upper case characters are mostly straightforward and unambiguous. However, in the case of lower case characters are very ambiguous. Several lowercase characters (like i, l, j) sometimes follow overlapping shapes and this caused miss-recognition a number of times.

The following figure (Figure 5) shows the improvement of test classification accuracy in percentage over the standard EBP and the evolutionary algorithm (EA) in CEDAR benchmark dataset. From the Figure 5, it shows that in cases for the proposed algorithm the test classification accuracies were higher than the standard EBP and EA methods. Whereas in case of EBP the improvement was 5.8% for lowercase and 10.6% for Uppercase, the results improved a lot when compared with standard EA method. In later case the improvement was 9% for lowercase and 14% for Uppercase.

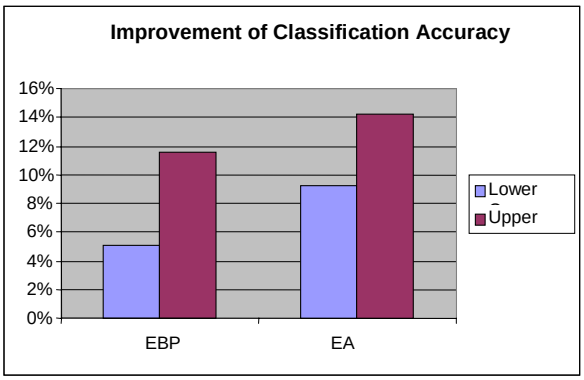

Figure 5: Improvement of Classification accuracy for CEDAR dataset

The following figure (Figure 6) shows the improvement of test classification accuracy in percentage over the standard EBP and the evolutionary algorithm (EA) in UCI Machine Learning repository (Upper case) dataset. The improvement was 2% for EBP and 5.6% for EA method.

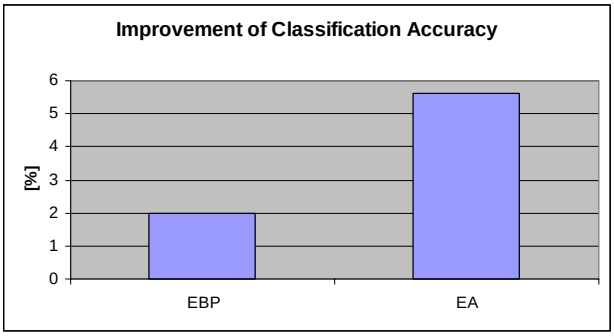

Figure 6: Improvement of Classification accuracy in UCI Machine Learning repository (Upper case) dataset

The following figure (Figure 7) shows the improvement of time complexity of the training dataset in percentage over the standard EBP and the evolutionary algorithm in CEDAR benchmark dataset. From the Figure 7, it shows that in cases for the proposed algorithm worked much faster than the standard EBP and EA methods. In case of EBP the improvement was 41% for Lower case and 42% for Uppercase. In case of EA, the improvement was 43.5% for lowercase and 45.5% for Uppercase.

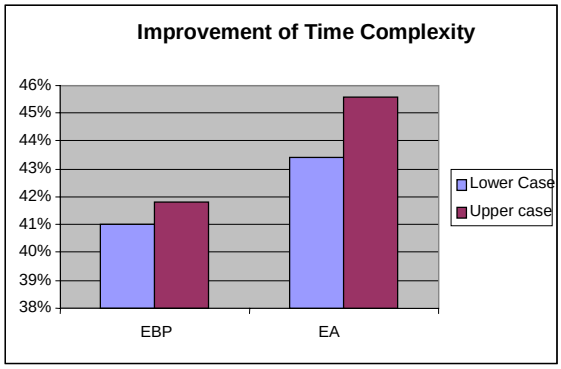

Figure 7: Improvement of Time Complexity in CEDAR dataset

The following figure (Figure 8) shows the improvement of time complexity of the training dataset in percentage over the standard EBP and the evolutionary algorithm in UCI Machine Learning repository (Upper case) dataset benchmark dataset. In case of EBP the improvement was 48% and in case of EA the improvement was 52%.

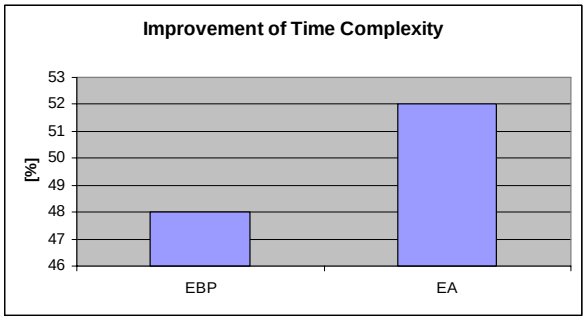

Figure 8: Improvement of Time Complexity in UCI Machine Learning repository (Upper case) dataset

Conclusion

The paper has concentrated on a hybrid evolutionary technique for offline handwriting recognition. It uses a combination of genetic algorithm and matrix based solution methods such as QR factorisation. The technique produces satisfactory results with improvements in character recognition and time complexity over the use of a Back Propagation network and Evolutionary algorithm. The result further improves with application of hamming neural network as lexicon analyser.

References

Eastwoord, B., and Jennings, A., and Harvey, A. 1997. A feature based neural network segmenter for handwritten words. In *Proceedings of the International Conference on Computational Intelligence and Multimedia Applications*, 286-290. Gold Coast, Australia.

Blumenstein, M; and Verma, B. 1999. A new segmentation algorithm for handwritten word recognition. In *Proceedings of International Joint Conference on Neural Networks,* 872-882. Washington, U.S.A.

Blumenstein, M; and Verma, B. 1999. A Neural-based solutions for the segmentation and recognition of difficult handwritten word from a benchmark database. In *Proceedings of International Conference on Document analysis and Recognition,* 281-284. Bangalore, India.

Lu, Y.; and Shridhar, M. 1988. Character segmentation in handwritten word – An overview. In *Proceedings of Pattern Recognition,* 77-96.

Ghosh, R.; and Verma, B. 2003. Finding architecture and weights for ANN using evolutionary based least square algorithm. *International Journal on Neural Systems*, 13(1): 13-24.