Combining Multiple OCRs for Optimizing Word Recognition

Prasun Sinha
Department of Computer Science
Univ. of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
prasun@crhc.uiuc.edu

Jianchang Mao
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120, USA
mao@almaden.ibm.com

Abstract

In this paper, we present a method of combining multiple classifiers for optimizing word recognition. As opposed to existing techniques for combining multiple OCRs, where the combination scheme is selected by either using some heuristics or using a character-level training procedure, the proposed method combines the results of individual classifiers in such a way that the correct word is more likely to be hypothesized. This method provides a solution to the crucial issue of assigning reliable cost to the edges of the segmentation graph in the popular *over-segmentation followed by dynamic programming* approach for word recognition. Three combination functions are proposed and implemented. Experiments show that proposed method for combining multiple classifier has a significant improvement on the word recognition accuracy.

Keywords: Classifier combination, word recognition, OCR.
Track: Pattern Recognition and Analysis.
Correspondence to: Dr. Jianchang Mao.

1 Introduction

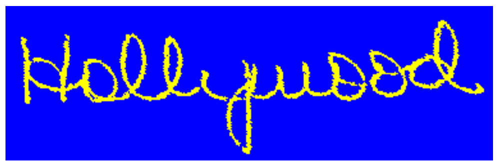
Handwritten word recognition is a challenging problem encountered in many real-world applications, such as postal mail sorting, bank check recognition, and automatic data entry from business forms. A great deal of research has been focused on improving character recognition accuracy. Such effort includes a recent trend to combine multiple character classifiers (OCRs), which leads to the development of a large number of techniques for combining multiple OCRs [5, 6, 7, 11]. In these techniques, a combination scheme is selected by either using some heuristics or using a character-level training procedure. However, a character recognizer optimized at the character level does not necessarily produce a high recognition accuracy at word level due to the high ambiguity in writing individual characters in a word and the enormous difficulty in segmenting word into correct characters.

A prevalent technique for off-line cursive word recognition is based on *over-segmentation followed by dynamic programming* [2, 4]. It seems to outperform segmentation-free Hidden Markov Models (HMMs) using a sliding window [8]. Over-segmentation based HMMs can also be built [3]. In general, the over-segmentation followed by dynamic programming approach does not require a probabilistic modeling. Heuristics can be easily incorporated into the recognition. On the other hand, word-level optimization is difficult to perform without any probabilistic modeling. In over-segmentation based word recognition, a more crucial issue is to estimate reliably the probability or confidence of a segment being a character category, rather than to hypothesize accurately the identity for each individual segment.

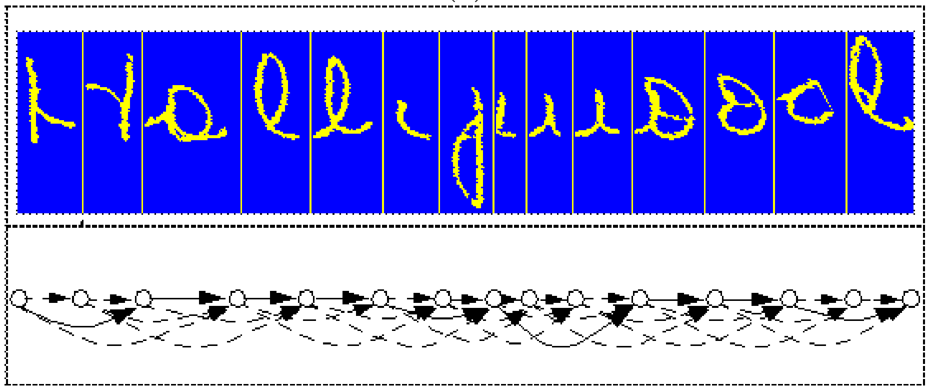
In this paper, we propose a method of combining multiple classifiers for optimizing word recognition. The proposed method employs a word-level cost function which is directly related to word recognition instead of character recognition. Minimizing this cost function is equivalent to maximizing the mutual information between the desired interpretation and all possible interpretations, i.e., maximizing the probability of obtaining the desired interpretation of the word image.

2 Word-level Cost function

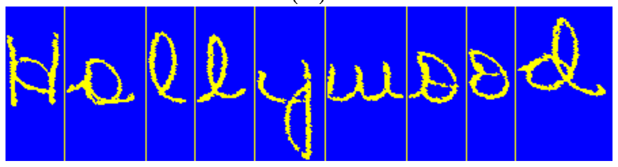
We use the over-segmentation followed by dynamic programming approach for cursive word recognition. In this approach, the word recognition problem is posed as a problem of finding the best path in a graph named *segmentation graph* (see Figure 1(b)). A set of split points on word strokes is chosen based on heuristics to divide the word into a sequence of graphemes (primitive structures of characters, see Figure 1(b)). A character may consist of one, two or three graphemes. Each internal node in the graph represents a split point in the word. The leftmost node and rightmost node indicate the word boundary. Each edge represents the segment between the two split points connected by the edge. Since our over-segmentor rarely produces more than three graphemes for a character, we remove all the edges which cover more than three graphemes. A character classifier is usually used to assign a cost to each edge in the segmentation graph. The dynamic programming technique is then used for finding the best path from the leftmost node to the rightmost node. A sequence of characters can then be obtained from the sequence of segments on the best path (see Figure 1(c)). Note that this sequence of characters may not form a valid word in a dictionary. If a lexicon of limited size is given, the dynamic programming technique is often used to rank every word in the lexicon. The word with the highest rank is chosen as the recognition hypothesis.



(a)



(b)



(c)

Figure 1: Over-segmentation followed by dynamic programming. (a) Original word image. (b) Graphemes and segmentation graph. (c) Segments on the correct path in the segmentation graph.

One challenging problem which is very crucial to recognition accuracy is how to assign a reliable cost to each edge in the segmentation graph such that the best path chosen by the dynamic programming corresponds to the correct path with a high probability. A common practice is to use a character classifier to assign costs to the edges. The classifier is first trained at character level to optimize recognition accuracy. Unfortunately, a character recognizer optimized at the character level does not necessarily produce a high recognition accuracy at word level due to the high ambiguity in writing individual characters in a word and the enormous difficulty in segmenting word into correct characters.

This paper proposes a method for improving the cost assignment by combining multiple character classifiers in such a way that the correct path through the graph becomes the highest probability path. We choose the following cost function:

$$\mathcal{C} = -\ln\left(\frac{P^{(d)}}{\sum_{i=1}^{N} P^{(i)}}\right) \tag{1}$$

where N is the total number of paths from the leftmost node to the rightmost node in the segmentation graph, the path-number for the correct or the desired path is d, and $P^{(i)}$ represents the probability of the i-th path. Minimizing this cost function is equivalent to maximizing the mutual information between the observations and the desired interpretation, i.e., maximizing the probability of obtaining the desired path.

Let S_{ij} be the segment corresponding to the j-th edge on the i-th path. Further, let $y_k(S_{ij})$ denote the k-th output of the word level trained classifier for the segment S_{ij}. The value of k in $y_k(S_{ij})$ is chosen to correspond to the j-th character on the i-th path. Then, the probability of the i-th path in the graph can be estimated as

$$P^{(i)} = \prod_{j=1}^{n_i} y_k(S_{ij}), \tag{2}$$

where n_i is the number of edges on the i-th path.

Bottou et al. [1] used the similar mutual information criterion for globally training a document processing system represented by graph transformer networks. They used a kind of back-propagation procedure to optimize the parameters in the system, including parameters in field locator, segmentor, and a neural network classifier. However, no one has applied word level criteria to optimize the combination of multiple classifiers (OCRs).

3 Combining Multiple Classifiers

Consider a set of M classifiers, each of which generates a C-dimensional output (C is the number of classes), $\mathbf{y}_i = (y_{i1}, \cdots, y_{iC})$, $i = 1, 2, \cdots, M$. We do not require that y_{ij} be an estimate of the *a posteriori* probability for the j^{th} class from the i^{th} classifier. Let $\mathbf{y} = (y_1, \cdots, y_C)$, be the C-dimensional output of the combination of the M classifiers. We have

$$\mathbf{y} = f_\Theta(\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_M),$$

where $\Theta = \{\theta_1, \cdots \theta_p\}$ are the p parameters of the combination function. The goal is to optimize these parameters so as to minimize the word-level cost function.

A larger number of combination functions can be used. In this paper, we demonstrate the effectiveness of combining multiple classifiers for optimizing word recognition using the following three combination functions of two neural network classifiers.

1. Softmax Function
$$y_k = \frac{e^{\alpha y_{1k} + \beta y_{2k}}}{\sum_{l=1}^{n} e^{\alpha y_{1l} + \beta y_{2l}}}, \tag{3}$$

2. Sigmoid Function
$$y_k = \frac{e^{\alpha y_{1k} + \beta y_{2k} + \gamma}}{1 + e^{\alpha y_{1k} + \beta y_{2k} + \gamma}}, \tag{4}$$

3. Nonlinear Scaling Function
$$y_k = \frac{\alpha y_{1k}^{\beta} + y_{2k}^{\gamma}}{\sum_{l=1}^{n} \alpha y_{1l}^{\beta} + y_{2l}^{\gamma}}, \tag{5}$$

where α, β, and γ are real numbers. When $\alpha = \beta = \gamma = 1$, the nonlinear scaling function reduces to the normalized averaging function. The softmax and sigmoid functions have a nice property of approximating probabilities well. Their derivatives are also well-behaved and easy to compute. On the other hand, the nonlinear scaling function has a problem with convergence, since the derivative has exponential and logarithmic terms which often can become too small or too large resulting in a bad approximation to the partial derivatives due to the finite numerical precision. Hence for some starting values of the parameters, the values of the parameters keep diverging. Even otherwise, it takes some 20 epochs to converge as opposed to 5 epochs in the case of sigmoid or softmax functions.

4 Optimization Method

We use the gradient descent method to estimate the set of parameters in a combination function to minimize the cost function defined in Equation (1). The parameter updating rule is as follows.

$$\theta_m^{(t+1)} = \theta_m^{(t)} - \eta \frac{\partial C}{\partial \theta_m^{(t)}}, \; m = 1, 2, \cdots, p, \tag{6}$$

where t is the index of iteration and η is the learning rate.

The partial derivatives of the cost function with respect to a parameter θ_m can be evaluated as follows.

$$\begin{aligned} \frac{\partial C}{\partial \theta_m} &= \sum_{j=1}^{n_d} \{ \frac{1}{y_k(S_{dj})} \times \frac{\partial y_k(S_{dj})}{\partial \theta_m} \} - \\ & \frac{\sum_{i=1}^{N} \sum_{j=1}^{n_i} \{ P^{(i)} \times \frac{1}{y_k(S_{ij})} \times \frac{\partial y_k(S_{ij})}{\partial \theta_m} \}}{\sum_{i=1}^{N} P^{(i)}} \end{aligned} \tag{7}$$

where the partial derivative $\frac{\partial y_k(S_{ij})}{\partial \theta_m}$ is determined by the combination function used to combine the results of multiple classifiers.

The computation of the above partial derivative involves an evaluation of two summations over all possible paths:

$$\sum_{i=1}^{N} P^{(i)} \text{ and } \sum_{i=1}^{N} \{ P^{(i)} \times \sum_{j=1}^{n_i} \frac{1}{y_k(S_{ij})} \times \frac{\partial y_k(S_{ij})}{\partial \theta_m} \}$$

A naive evaluation of these two summations is prohibitive because the value of N is often very large. Fortunately, we find that both these summations can be computed using the dynamic programming technique.

To simplify the notations, we introduce a new variable

$$T(S_{ij}, k) = \frac{1}{y_k(S_{ij})} \times \frac{\partial y_k(S_{ij})}{\partial \theta_m}.$$

Let us number the nodes in the segmentation graph from 0 to τ, where the 0-th node represents the leftmost node and the τ-th node represents the rightmost node. The algorithm requires calculation of partial results at each of these nodes from left to right. Let us define the two partial results $A^{(t)}$ and $B^{(t)}$ at node number t.

$$A^{(t)} = \sum_{i=1}^{N^{(t)}} P^{(i)} \tag{8}$$

$$B^{(t)} = \sum_{i=1}^{N^{(t)}} \{ P^{(i)} \times \sum_{j=1}^{n_i} T(S_{ij}, k) \} \tag{9}$$

$N^{(t)}$ corresponds to the total number of paths from the node 0 to node t. S_{ij} corresponds to the j-th segment on the i-th path.

Hence, the final summations that we are interested in are $A^{(\tau)}$ and $B^{(\tau)}$.

The algorithm is as follows:

1. *Initialization:* $A^{(0)} = 1$, $A^{(-1)} = 0$, $A^{(-2)} = 0$, $B^{(0)} = 0$, $B^{(-1)} = 0$ and $B^{(-2)} = 0$. $A^{(-1)}$, $A^{(-2)}$, $B^{(-1)}$ and $B^{(-2)}$ have been initialized to make the recursion step simple.

2. *Recursion:* Compute $A^{(t+1)}$ and $B^{(t+1)}$ from $A^{(t)}$ and $B^{(t)}$ for $t = 0 \cdots (T-1)$. Finally we get $A^{(\tau)}$ and $B^{(\tau)}$, the summations we were looking for. $S_{i;j}$ in the following equations refers to the segment corresponding to the edge in the segmentation graph from node i to node j.

$$A^{(t+1)} = A_{(t)} \times \sum_{k=1}^{n} y_k(S_{t;t+1}) + A_{(t-1)} \times \sum_{k=1}^{n} y_k(S_{t-1;t+1}) + A_{(t-2)} \times \sum_{k=1}^{n} y_k(S_{t-2;t+1}) \tag{10}$$

$$\begin{aligned} B^{(t+1)} = & \sum_{k=1}^{n} [y_k(S_{t;t+1}) \times \{B^{(t)} + A^{(t)} \times T(S_{t;t+1}, k)\}] + \\ & \sum_{k=1}^{n} [y_k(S_{t-1;t+1}) \times \{B^{(t-1)} + A^{(t-1)} \times T(S_{t-1;t+1}, k)\}] + \\ & \sum_{k=1}^{n} [y_k(S_{t-2;t+1}) \times \{B^{(t-2)} + A^{(t-2)} \times T(S_{t-2;t+1}, k)\}] \end{aligned} \tag{11}$$

5 Experiments

For our experiments on word level training, 5305 words are automatically extracted from cursive script USPS addresses. These words are over-segmented and each of the graphemes were then manually truthed. Another independent set of 4417 words is used for testing. For each word, two lexicons of size 10 and 100 are generated randomly with the true word always included. In our experiments, we use two neural network classifiers as the individual classifiers. The two neural networks have the identical topology with 27 outputs, 50 hidden units, and 108 input units. Twenty six output units (out of 27) correspond to the 26 alphabetic characters and the additional one represents a non-character. The upper-case and lower-case of each letter are combined into a single category. For each segment (edge in the segmentation graph), a total of 108 features are extracted, of which 88 are contour directional features extracted from the size-normalized bitmap [9]. The rest 20 features capture the relative size (height and width) of the segment with respect to its two neighboring

Table 1: *Character recognition and word recognition accuracies of the two individual classifiers. The lexicon size is 10.*

Cost function	*Character Recognition*		*Word Recognition*	
	Training	*Test*	*Training*	*Test*
Kullback-Leibler	70.37%	67.31%	87.07%	88.91%
Squared Error	74.40%	71.16%	89.41%	90.97%

graphemes, and the normalized position (column and row) of splitting points on both sides of the segment in the joint bounding box with its two neighboring graphemes, respectively.

The two classifiers are trained using two different character-level cost functions. One uses Kullback-Leibler function. In this network, the sigmoid function is used as the activation function in the hidden layer and the softmax function is used as the activation function in the output layer. The second network is trained using the squared error function. The sigmoid function is used as the activation function in both the hidden layer and the output layer.

After training is done, the two classifiers are separately tested for both character recognition and word recognition. The results on the training data and test data with a lexicon size of 10 are summarized in Table 1. We see that the network trained with the square-error cost outperforms the other at both the character-level and word-level recognition. Note that the character level recognition rates of the two classifiers are relatively low compared to results reported in the NIST conference on isolated character recognition [10]. This is due to two facts: (i) in our case the upper-case and lower-case characters are mixed, and (ii) the excessive ligature in cursive handwriting degrades the character recognition.

We then combine these two networks using the three combination functions. The parameters in each combination function are estimated to minimize the work-level cost function defined in Equation (1) on the training data. These parameters are listed in Table 2, which also summarizes the results on both the character recognition and word recognition accuracies using the three combination functions. The training were repeated several times, but we found that different trials produce very similar results.

From Table 2, we can see that the word level training does not necessarily improve the character recognition accuracy (in case of *Sigmoid* combination, it slightly improves, but in the other two cases, the character recognition accuracy drops). However, the word recognition accuracy is significantly improved over individual classifiers by using the combinations of

Table 2: *Character recognition and word recognition accuracies of the three combination functions whose parameters are obtained by the word level training. The lexicon size is 10.*

Combination	*Parameters*			*Character Recognition*		*Word Recognition*	
function	α	β	γ	*Training*	*Test*	*Training*	*Test*
Softmax	3.72	5.07		73.00%	70.28%	95.08%	94.95%
Sigmoid	6.72	9.27	-5.73	75.38%	72.29%	95.34%	95.20%
Nonlinear scaling	36.65	1.03	1.26	70.55%	67.43%	92.72%	92.05%

Table 3: *Word recognition results with a lexicon size of 100.*

Classifier 1: Kullback-Leibler	*Classifier 2: Square error*	*Combination: Sigmoid function*
70.70%	74.69%	84.47%

the two classifiers. The sigmoid combination function achieves the best performance, which is better than the best individual classifier by 5.93% on the training data and 4.23% on the test data. The nonlinear scaling function is significantly inferior to the sigmoid and softmax functions at both the character level and word level recognition.

We also perform the word recognition with a lexicon size of 100. For the sake of limited space, we report only the word recognition results on the test data using the two individual classifiers and the sigmoid function in Table 3. We see that an improvement of nearly 10% on word recognition accuracy is achieved by using the sigmoid combination function optimized at the word level.

6 Conclusions

We have presented a method for combining multiple OCRs for optimizing word recognition. Experiments have shown that the proposed method is an effective technique for improving word recognition accuracy, but is not necessarily effective for character level recognition. The proposed method can also be used for combining information from neighboring characters, which is much powerful because it makes use of context information to resolve inherent ambiguity in handwritten characters.

References

[1] L. Bottou, Y. Bengio, and Y. LeCun. Global training of document processing systems using graph transformer networks. In *Proc. IEEE Computer Soc. Conf. on Computer Vision and Pattern Recognition*, pages 489–494, Puerto Rico, June 1997.

[2] R. M. Bozinovic and S. N. Srihari. Off-line cursive word recognition. *IEEE Trans, PAMI*, 11:68–83, Jan 1989.

[3] M. Y. Chen, A. Kundu, and S. N. Srihari. Variable duration hidden markov model and morphological segmentation for handwritten word recognition. *IEEE Trans. on Image Processing*, 4(12):1675–1688, December 1995.

[4] P. D. Gader, J. M. Keller, R. Krishnapuram, J. H. Chiang, and M. A. Mohamed. Neural and fuzzy methods in handwriting recognition. *Computer*, pages 79–85, February 1997.

[5] T. K. Ho, J. J. Hull, and S. N. Srihari. Decision combination in multiple classifier systems. *IEEE Trans. Pattern Anal. Machine Intell.*, 16(1):66–75, 1994.

[6] J. Kittler. Improving recognition rates by classifier combination. In *Intl. Workshop on Frontiers in Handwriting Recognition*, pages 81–102, Colchester, UK, Sept. 1996.

[7] J. Mao and K. M. Mohiuddin. Improving ocr performance using character degradation models and multiple classifiers. *Accepted to appear in Pattern Recognition Letters.*

[8] M. Mohamed and P. Gader. Handwritten word recognition using segmentation-free hidden markov modeling and segmentation-based dynamic programming techniques. *IEEE Trans. Pattern Anal. Machine Intell.*, 18(5):548–554, May 1996.

[9] H. Takahashi. A neural net OCR using geometrical and zonal-pattern features. In *Proc. 1st Intl. Conf. on Document Analysis and Recognition*, pages 821–828, 1991.

[10] R. A. Wilkinson and J. Geist et al. (eds). The first census optical character recognition system conference. Technical report, NISTIR 4912, U.S. Department of Commerce, NIST, Gaitherburg, MD 20899, 1992.

[11] L. Xu, A. Krzyzak, and C. Y. Suen. Methods for combining multiple classifiers and their applications in handwritten character recognition. *IEEE Trans. Syst., Man, Cybern.*, 22(3):418–435, 1992.