Proceedings 2003 IEEE International Symposium on
Computational Intelligence in Robotics and Automation
July 16-20, 2003, Kobe, Japan

Offline Cursive Handwriting Recognition System based on Hybrid Markov Model and Neural Networks

Yong Haur Tay, Marzuki Khalid*, Rubiyah Yusof, and C. Viard-Gaudin
Centre for Artificial Intelligence and Robotics (CAIRO),
Universiti Teknologi Malaysia, Jalan Semarak, 54100 Kuala Lumpur, Malaysia.
E-mail: marzuki@utmkl.utm.my (*contact person)

Abstract

An offline cursive handwriting recognition system, based on hybrid of Neural Networks (NN) and Hidden Markov Models (HMM), is described in this paper. Applying SegRec principle, the recognizer does not make hard decision at the character segmentation process. Instead, it delays the character segmentation to the recognition stage by generating a segmentation graph that describes all possible ways to segment a word into letters. To recognize a word, the NN computes the observation probabilities for each segmentation candidates (SCs) in the segmentation graph. Then, using concatenated letter-HMMs, a likelihood is computed for each word in the lexicon by multiplying the probabilities over the best paths through the graph. We present in detail two approaches to train the word recognizer: 1). character-level training 2). word-level training. The recognition performances of the two systems are discussed.

I. Introduction

Cursive handwriting recognition problem has only been profoundly studied since last decade. Given the ambiguity of the human cursive handwriting, the task to develop a reliable recognizer presents a technical challenge. At present, most of the successful commercially applied handwriting recognizers are for environment with small and specific lexicon, for example as in bank cheque's legal amount recognition [1]. The increase of the size of lexicon expands the complexity of the recognition task, specifically the computation and recognition performance. Ideally, word recognition process can be considered as an extension of character recognition process, whereby a word image is segmented into letters, then recognized individually by a character recognizer. However, the process of character segmentation is not simple without the knowledge of characters. Likewise, the character recognizer cannot recognize correctly without the true segmentation. This is referred to as the Sayre's paradox [2].

Realizing the interdependency of segmentation and recognition process, our approach that is based on Segmentation by Recognition (SegRec) principle proposes to delay the character segmentation process until later stage by the character recognizer. In order to recognize a word image, it is segmented into sub-character image frames using a fast sliding window technique. Then, by combining several image frames, we generate a segmentation graph consisting of segmentation candidates (SCs) that suggest all possible ways to segment a word image into letters. Having all the character recognition probabilities for each SC, the final process decides the best segmentation based on the character recognition results. This approach reduces information lost from one process to the successive one. However, as the SCs may consist of sub-characters or multiple characters, or we termed them 'junks', the character recognizer must model those SCs to only give low probabilities for character classes. We describe in detailed two approached to train the word recognizer: 1). Character-level training 2). Word-level training.

In this paper, we first introduce the fundamental processing and configuration in our cursive handwritten word recognizer in Section 2. This is followed by presentation of the character-level training scheme. In Section 4, we present another method to train the NN using the word-level criteria. Experimental results have been obtained using the IRONOFF isolated word database [3].

II. NN-HMM Hybrid Recognition System

Our cursive handwritten word recognizer is a segmentation-based, lexicon-driven system. Detailed description of the system can also be found in [4].

A. Slant Correction

Slant correction is required in order to reduce the variability of handwriting styles. Our slant correction process utilizing the techniques used in [1]. For a given word, we run a contour-following algorithm on each connected component. Each external contour of a connected component is described as a list of contour vectors, each of which can be classified as horizontal(n_h), vertical(n_v), diagonal +45°(n_+), or diagonal -45°(n_-). We count the contour vectors of each class and compute the slant Θ as the average orientation of the vertical parts of the word:

$$\Theta = \arctan\left(\frac{n_+ - n_-}{n_+ + n_v + n_-}\right) \quad (2.1)$$

Slant correction is applied on the whole word image by shearing the image horizontally:

$$x' = x - y\sin\Theta \qquad \text{and} \qquad y' = y \quad (2.2)$$

B. Reference Line Detection

Reference lines carry important information for handwriting recognition systems, as they help in normalizing the image size and in extracting geometrical features related to the position of letters with respect to their context. Given an input word image, our goal is to find four parallel straight lines and their respective position:

1. Ascender line, positioned at the top of letters like 'K', 'h', and 't',
2. Core line, positioned at the top of lower case letters like 'a', 'e', and 'm',
3. Base line, positioned at the bottom of letters like 'a', 'e', and 'm',
4. Descender line, positioned at the bottom of characters like 'p', 'q', and 'y'.

To detect the reference lines, we first smooth the binary image. Then, we extract the local minima and maxima of the handwriting signal by running a contour following algorithm on the internal and external contours of the word image. Together with the a priori probability distribution of the line positions, these extrema are used as the observations for an Expectation-Maximization (EM) algorithm in order to estimate position and slant of the 4 straight parallel reference lines. More details on this algorithm can be found in [5].

C. Segmentation

The word image is cut from left to right into slices of variable width. In order to achieve size invariant segmentation, the average slice width depends on the height of the core-zone of the word. The parameters of the segmentation algorithm are chosen such that each slice contains either a letter or part of a letter, but not more than a letter (over-segmentation). The exact position of each cut is determined such that a minimum number of ink pixels have to be crossed. From the left-right ordered sequence of slices, we combine several consecutive slices to form SCs. A segmentation graph represents all possible ways to segment the word into letters. Figure 1 depicts an example of a word image cut into four slices. The right side of the figure shows the segmentation graph that consists of SCs composed of 1 to 4 consecutive frames. A maximum of 9 consecutive frames is allowed for a SC.

D. Feature Extraction

The main objective of the feature extraction process is to capture the most relevant and discriminant characteristics of the character to recognize. We extract 140 geometrical features from each SC. The features are 1) Dimension and aspect ratio of the bounding box of ink-pixels in a frame, 2) Center of gravity, 3) Distance to the core-zone, 4) Profiles in 8 directions, and 5) Number of transitions from non-ink-pixel to ink-pixel, for vertical, horizontal, +45° and -45° diagonal direction. All features are normalized by the height of the core zone, h, in order to make the features invariant with respect to the size of the handwriting.

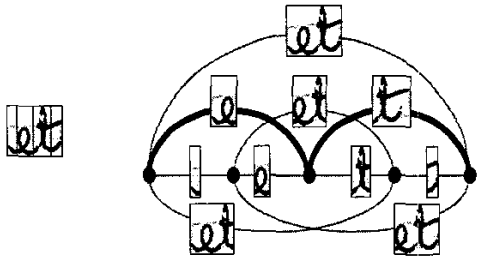

Fig. 1. (Left) Word 'et' is segmented in 4 slices. (Right) Segmentation graph describing all possible ways to segment the word into letters. The bold lines indicate the true segmentation paths for the word 'et'.

III. Character Recognition using NN

For each SC in the segmentation graph, we perform character recognition. We first extract geometrical features from the SC, and feed them into a trained NN for classification. The topology of the NN is a 3-layered Multi-layered Perceptrons (MLP), with softmax activation at the output layer (see Fig). The output of this process is a list of character probabilities. Notice from the Fig2 that there could be many non-characters exist as SCs, therefore, it is important that the NN must be able to model those SCs. It has to give low probability for all character classes if those junks are presented. To solve this, we propose to add a 'junk' output neuron to the NN. Approach to train the junk class will be discussed in succeeding sections.

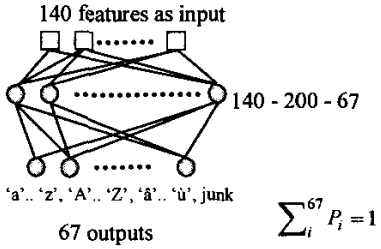

Fig. 2. Topology of the softmax MLP

A. Word-likelihood Computation using HMM

For each entry in the lexicon, we build a word HMM (Hidden Markov Models) by concatenating letter HMMs and ligature HMMs as illustrated in Fig. 3. The gray states are 'dummy' states that do not emit observation probabilities. They allow for the alignment with the observation sequence where each slice is an observation, and thereby allow the use of the usual HMM tools. The observation probabilities in

each emitting state of the basic HMMs (letter- and ligature-HMMs) are computed by the NN. The likelihood for each word in the lexicon is computed by multiplying the observation probabilities over the best path through the graph using the Viterbi algorithm. The word HMM with the highest probability is the first recognition candidate.

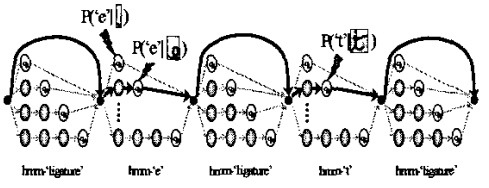

Fig. 3. Word HMM 'et' is composed of letter HMMs 'e' and 't', with ligature-HMMs inserted before and after. Lines in bold indicate the best path through the model for the example of Fig. 2.

B. Character-Level Discriminant Training

The basic idea in character-level training scheme is that we train the NN as a character recognizer. The NN is trained using isolated characters segmented from the word images. With the NN to generate observation probabilities, we train the transition probabilities in HMM to estimate the duration frequency of a character. And finally, we combine the two at the recognition stage. Figure 4 illustrates the structural training scheme to improve the performance of the recognizer. For this training scheme, we initially use isolated characters generated by our baseline word recognition, which is based on discrete HMM, to automatically segment word image into characters (Bootstrap process). Once we have a word recognizer, we then use the word recognizer to regenerate isolated characters to restart the process again (Viterbi B). And finally, we train the NN to model junk examples (Discriminant Training).

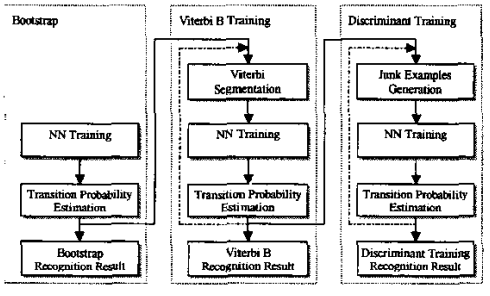

Fig. 4. Training Stages of the NN-HMM hybrid recognizer

Bootstrap

We first need to train the NN to recognize characters. This can be done by manually segmenting word images into isolated character images with associated class labels. As this approach is rather laborious and time-consuming, we opted for an automatic approach, which uses the trained baseline recognizer to segment the word images in the training set into isolated characters. This can be done by running the Viterbi backtracking algorithm to select the best segmentation path, given the true HMM. Baum-Welch training is performed after this to estimate the transition probabilities.

Viterbi B

Given the discriminant power of the NN, the initial bootstrapped recognition result shall already be better that the baseline recognizer. We iterate this training procedure a few times by using the obtained hybrid recognizer to segment the word images into letters for the NN training. Baum-Welch training is again performed to estimate the transition probabilities.

Discriminant Training

During the first two stages, the recognizer managed to improve its recognition accuracy significantly. However, one problem remains unsolved in which the NN was not trained to reject non-characters, also termed "junk". For these SCs, e.g. part of a character or combination of a few characters, the NN will gave unpredictable results. This is known as the collapse problem [6]. To solve this problem, we have added another output to the NN, which is responsible for giving a high probability if a non-character/junk is presented. Although the HMM will not use this probability directly, a high probability at this junk class output will eventually flatten the probability distribution of other character classes if a junk example is presented (Remember that we are using softmax normalization at the NN output layer, which sum all outputs to 1.0). At this training stage, we generate junk examples and combine character examples to retrain the NN. Baum-Welch training for transition probabilities is again performed after the NN retraining.

Given a true HMM, all SCs that do not belong to the best segmentation are junk examples. This method can generate almost the entire junk examples that can be found in the training database. However, the problem of this using this method is that a huge number of junk examples will be generated to train the NN. With more training examples, the training process would be very time-consuming. Some of the SCs are not in the best segmentation, given the true HMM, it might resemble other characters. For examples, 'w' can be combination of two 'u's, left part of 'd' could look like a 'c', and so on. Generating those SCs as junk examples will create competition between junk class and those character classes which will eventually pull down the probability of those classes compared to others that have no such problem.

Thus, we introduce the discriminant training, which is a process to carefully choose the SCs that cause recognition errors as junk examples. The key idea of the training is that, given a true HMM, λ^{τ} ,and the best HMM, λ^{*}, we compute the difference of γ between each SC. The γ probability represent probability that a given observation is a true letter observation for the considering word-HMM, λ. γ can be computed using the forward α, and backward β, variables as defined as below:

$$\gamma_i = \frac{\alpha_i \beta_j}{P(O \mid \lambda)} \quad (3.1)$$

The SCs that have the minimum of difference between two γ is considered as the junk examples. The junk example from the SCs can be formulated as:

$$Junk = \arg\min_{i \in T} \gamma_i^{\tau} - \gamma_i^{*} \quad (3.2)$$

where T is the total number of SCs.

We run the training for a few iterations until the optimum recognition is obtained.

IV. Experimental Results

We have tested our recognition system on isolated words from the IRONOFF [3] database. The database contains a total of 36,396 isolated French word images from a 196-word lexicon. The offline handwriting signals are sampled with spatial resolution of 300 dots per inch (DPI), with 8 bits per pixel (256 gray level). The training data set contains 24,177 word images, and another 12,219 images are used as test data set. The scriptors of the two data sets are different. This reflects an omni-scriptor situation where only some types of handwriting styles are available during the designed training of the system. We name the full database as IRONOFF-196.

In all the presented experiments, recognition scores at the word level were evaluated using two performance measures. The recognition rate is the percentage of samples that are correctly classified. Generalizing this concept, we also compute the recognition rate, Rec (K), in the second, third and subsequent position, which is the cumulated recognition rate of the first K position in the candidate list. The second measure is the average position, $\overline{pos}$, of the true class in the candidate list. This measure tells more about the distribution of the probabilities within the candidate list, and is therefore more informative with respect to the performance of the complete recognition system. If the true class is not recognized in the first position, but in the second or third position, and with a relatively high probability, the true class may still be recognized in subsequent processing step.

Table 1. Recognition performance (test set) for each training stage on IRONOFF-196

Training Stage	Rec(1)
Bootstrap	91.7
Viterbi B	95.0
Discriminant Training (196-lex)	95.5
Discriminant Training (2000-lex)	96.1

Table 1 shows the recognition performance of the NN-HMM hybrid recognizer for each training stage. The training and testing is performed on IRONOFF-196 database. The NN is initially trained with segmentation by the baseline recognizer, which achieve Rec(1) of 89.3%. After the bootstrap process, the recognition of the hybrid recognizer is 91.7%, which is 2.3% better than the baseline recognizer. This indication proves the strength of the hybrid recognizer. By regenerating isolated characters using the hybrid recognizer in the Viterbi B stage, it then gains 4.3% improvement. This is because the hybrid recognizer, having better recognition, is able to segment word into characters more precisely. We separate the discriminant training stage into 2. First, we generate junk examples using the original 196-word lexicon. Rec(1) improve to 95.5% after this stage. We felt that the performance could be better by letting the NN to have more examples of other junks. Thus, at the second discriminant training stage, we instead use a lexicon of 2000 words that closely resemble words in the original 196-word lexicon. This will make the recognition tougher and thus, more junk examples can be generated. And finally it achieves 96.1% of recognition rate, which is 7.5% of recognition improvement over the baseline recognizer, or in other words, a reduction of 70% error (from 10.7% error to 3.2% error).

V. Word-Level Discriminant Training

Although character-level training yields better recognition, it trains the NN and HMM separately, i.e. by segmenting words into isolated letters, which are then used to train the NN [7, 8]. Therefore, the NN is optimized at the character level, which does not guarantee optimal recognition performance at the word level. Secondly, within this type of approach, the outputs of the NN are divided by the class prior probabilities. This results in 'scaled-likelihood', which are used as the observation probabilities in the HMMs. This normalization often generates problems if some letter classes have very low prior probabilities. For instance capital letters usually have relatively small priors (thus large scaled-likelihoods) compared to lower-case letters. Furthermore, character level training implies that we need to provide examples of letters as well as non-letters ('junk') to train the NN, which is not an easy task [4].

Word level discriminant training seems to be an answer to our problem [6, 9]. Instead of generating isolated letters from the word images in order to train

the NN separately, we want to instantly back-propagate the error at the word level into the NN to update its parameters. All transition probabilities in the HMMs are set to 1 and are not modified during training. We base the word level objective function L on the Maximum Mutual Information (MMI) as follows:

$$L = \log \frac{P(O|\lambda^r)}{\sum_{\lambda} P(O|\lambda)} = \log P(O|\lambda^r) - \log \sum_{\lambda} P(O|\lambda) \quad (4.1)$$

where $P(O|\lambda)$ is the likelihood of the observation sequence $O = O_1 O_2 ... O_T$, given the HMM λ. λ^r is the true word HMM, and λ varies over all word HMMs in the given lexicon. The objective function is optimized using gradient descent (back-propagation).

To reduce the computational complexity of Eq. (1), an approximate cost of L can be written as

$$L' = \log P(O|\lambda^r) - \log P(O|\lambda^*) \quad (4.2)$$

where λ^* is the HMM with the largest likelihood among all λ or

$$\lambda^* = \arg\max_{\lambda} P(O|\lambda) \quad (4.3)$$

Eq. (4.2) states that the word-level objective function is based on the difference between the log-likelihood of the true HMM, λ^r and the best HMM, λ^*. If λ^r is also the best HMM, then $L' = \log P(O|\lambda^r) - \log P(O|\lambda^r) = 0$, and thus, no error is back propagated to update the NN. Otherwise, we change the weights W of the NN using the chain rule:

$$\frac{\partial L'}{\partial W} = \sum_j \frac{\partial L'}{\partial b_j(O_t)} \cdot \frac{\partial b_j(O_t)}{\partial W} \qquad \forall O_t \quad (4.4)$$

The word likelihood of HMM λ, given observation sequence O can be defined as:

$$P(O|\lambda) = \sum_{\Gamma} \prod_{t=1}^{T} a_{q_{t-1}q_t} b_{q_t}(O_t) \quad (4.5)$$

where a, $b(O_t)$ are the transition probabilities, and observation probabilities, respectively, and the sum runs over all paths Γ through the HMM λ [10]. The derivatives of a, $b(O_t)$ can be written as:

$$\frac{\partial a_{q_{t-1}q_t}}{\partial a_{ij}} = \delta_{i,q_{t-1}} \delta_{j,q_t}$$
$$\frac{\partial b_{q_t}(O_t)}{\partial b_j(O_t)} = \delta_{j,q_t} = \delta_{j,q_t} \cdot \frac{b_{q_t}(O_t)}{b_j(O_t)} \quad (4.6)$$

where $\delta_{i,j} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$

Therefore, the derivative of the word likelihood with respect to the observation probability can be written as:

$$\frac{\partial P(O|\lambda)}{\partial b_j(O_t)} = \sum_{\Gamma} \prod_{t=1}^{T} a_{q_{t-1}q_t} \delta_{j,q_t} \cdot \frac{b_{q_t}(O_t)}{b_j(O_t)}$$
$$= \frac{1}{b_j(O_t)} \sum_{\Gamma} \prod_{t=1}^{T} a_{q_{t-1}q_t} \delta_{j,q_t} b_{q_t}(O_t) \quad (4.7)$$
$$= \frac{1}{b_j(O_t)} \sum_{\Gamma} P(O,\Gamma,q_t = j|\lambda)$$
$$= \frac{1}{b_j(O_t)} P(O,q_t = j|\lambda)$$

where $P(O,q_t = j|\lambda)$ can be computed by a dynamic programming algorithm [10].

The first term in Eq. (4) can be further derived as:

$$\frac{\partial L'}{\partial b_j(O_t)} = \frac{1}{P(O|\lambda^r)} \cdot \frac{\partial P(O|\lambda^r)}{\partial b_j(O_t)} - \frac{1}{P(O|\lambda^*)} \cdot \frac{\partial P(O|\lambda^*)}{\partial b_j(O_t)} \quad (4.8)$$
$$= \frac{1}{P(O|\lambda^r)} \cdot \frac{P(O,q_t = j|\lambda^r)}{b_j(O_t)} - \frac{1}{P(O|\lambda^*)} \cdot \frac{P(O,q_t = j|\lambda^*)}{b_j(O_t)}$$
$$= \frac{1}{b_j(O_t)} \left(\frac{P(O,q_t = j|\lambda^r)}{P(O|\lambda^r)} - \frac{P(O,q_t = j|\lambda^*)}{P(O|\lambda^*)} \right) \quad (4.2)$$

where $\sum_j \frac{P(O,q_t = j|\lambda)}{P(O|\lambda)}$ is the probability that a given observation is one of the letters in the considered word HMM λ.

For the second term of the equation (4.4), as $b_j(O_t)$ is actually the output of the NN given O_t, we can use the usual back-propagation algorithm to update the weights of the NN.

We tested our recognition system on the IRONOFF database. We bootstrapped the recognizer by using the NN that have been trained at the character level. Once the NN has been bootstrapped, we started to train the recognizer using the word-level discriminant training. Table 2 shows the recognition performances of the recognizers using two different approaches. For IRONOFF-196, word-level training is 1.0% more accurate than the character-level training (about 25.6% of error reduction). To further analyze the performance of the recognizers, we tested the systems using larger size of lexicon, i.e. 2000 words. It shows that for IRONOFF-2000, word-level training is 5.0% better in recognition performance (about 29.5% of error reduction), as well as significant improvement of the average position, $\overline{pos}$.

Table 2. Recognition performances of the recognizers on two test data sets.

Database	Char-level Training		Word-level Training	
	Rec(1)	$\overline{pos}$	Rec(1)	$\overline{pos}$
IRONOFF-196	96.1	1.4	97.1	1.2
IRONOFF-2000	83.1	5.9	88.1	3.0

VI. Conclusions

In this paper, we described an offline handwriting recognition system using hybrid of NN and HMM. In order to minimize lost of information, the segmentation process proposes all possible ways for cutting a word image into letters. By using character recognition results on each SC by the NN, the HMM decides the best segmentation path based on the word likelihood computation. Two approaches to train the word recognizer are presented, namely character-level discriminant training and word-level discriminant training. Recognition performances on IRONOFF are presented and show the superiority of word-level approach compared to character-level training. In addition to that, word-level discriminant training eliminate the used of scaled-likelihood, which is difficult to adjust. Further experiments will be carried out to train the recognizer directly using random initialization. This will eliminate the process of bootstrapping using another recognizer.

Acknowledgements

The authors would like to express their deepest gratitude to Dr. Pierre-Michel Lallican and Dr. Stefan Knerr from Vision Objects for their help and guidance.

References

[1] S. Knerr, E. Augustin, O. Baret, and D. Price, "*Hidden Markov Model Based Word Recognition and Its Application to Legal Amount Reading on French Checks*", Computer Vision and Image Understanding, vol. 70, no. 3, June 1998, pp. 404-419.

[2] T.Steinherz, E.Rivlin and N. Intrator, "*Off-Line Cursive Script Word Recognition – A Survey*", Int'l Journal of Document Analysis and Recognition, 1999.

[3] C. Viard-Gaudin, P.M. Lallican, S. Knerr, P. Binter, "*The IRESTE On/Off (IRONOFF) Dual Handwriting Database*", International Conference on Document Analysis and Recognition, 1999.

[4] Y.H.Tay, P.M.Lallican, M.Khalid, C.Viard-Gaudin and S.Knerr, "An Offline Cursive Handwritten Word Recognition System", Proc. of TENCON, 2001, Singapore, 2001.

[5] Y. Bengio, Y. LeCun, "Word Level Training of a Handwritten Word Recognizer Based on Convolutional Neural Networks", International Conference on Pattern Recognition, pp. 88-92, 1994.

[6] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, "*Gradient-Based Learning Applied to Document Recognition*", Proceedings of IEEE, Vol. 86, No. 11, pp. 2278-2324, 1998.

[7] R.Plamondon, S.N.Srihari, "On-Line and Off-line Handwriting Recognition: A Comprehensive Survey", IEEE Transactions on PAMI, Vol.22, No. 1, pp.63-84, 2000.

[8] S. Knerr, E. Augustin, "A Neural Network-Hidden Markov Model Hybrid for Cursive Word Recognition", International Conference on Pattern Recognition 98, Brisbane, 1998.

[9] Y. Bengio, R. De Mori, G. Flammia, R. Kompe, "Global Optimization of a Neural Network-Hidden Markov Model Hybrid", IEEE transactions on Neural Networks, Vol. 3, No. 2, pp. 252-258, 1992.

[10] L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition", Proceedings of the IEEE, vol. 77, pp. 257-285, 1989.