MINIMUM CLASSIFICATION ERROR TRAINING OF HIDDEN MARKOV MODELS FOR HANDWRITING RECOGNITION

Alain E. Biem

IBM T. J. Watson Research Center
P. O. Box 218, Yorktown Heights, NY 10598, USA
biem@us.ibm.com

ABSTRACT

This paper evaluates the application of the Minimum Classification Error (MCE) training to online-handwritten text recognition based on Hidden Markov Models. We describe an allograph-based, character level MCE training aimed at minimizing the character error rate while enabling flexibility in writing style. Experiments on a writer-independent discrete character recognition task covering all alpha-numerical characters and keyboard symbols show that MCE achieves more than 30% character error rate reduction compared to the baseline Maximum Likelihood-based system.

1. INTRODUCTION

This paper evaluates the application of Minimum Classification Error (MCE) training to Hidden Markov Models (HMM)-based handwriting recognition. With the advent of wireless technologies, Personal Digital Assistants (PDA) and handheld computers, as convenient replacements for standard keyboard-based input systems, pen-based man-machine communication has received a renewed interest in the research community, and accurate on-line handwriting recognition has become a critical and urgent issue. On-line handwriting recognition attempts to recognize characters and treats the input as a time-series. Among the various known techniques used to process the handwriting signal, Hidden Markov Modeling has become the method of choice, inspired by its success in speech recognition [1].

In handwriting recognition, the input signal is the pen trajectory ("electronic ink"), recorded as a two-coordinate signal. Preprocessing, denoising, and normalization transform the electronic ink into a sequence of feature vectors $X = x_1, ..., x_T$ of length T. Decoding is based on the Bayes decision theory, which chooses the character or word C that has the maximum *a Posteriori* probability. That is,

$$\widehat{C} = \arg\max_{C} P(X|C)P(C). \quad (1)$$

In HMM-based handwriting recognition, the widespread approach to implementing Bayes decision theory is to model the class conditional probability $P(X|C)$ by an HMM, and estimate the parameters of the model by the Maximum Likelihood (ML) criterion. The ML criterion can be conveniently implemented by the EM reestimation procedure, a hill-climbing process, leading at least to a local optimum in parameter space. A theorem by Nadas [2] has shown that if the family of distributions generated by the model contains the true distribution, the ML criterion converges asymptotically to the true distribution of the data, assuming that a sufficient number of data are available. In general, the number of data is limited, the true distribution is unknown, and the Markov assumption is a convenient but not an accurate one. Furthermore, likelihood increase does not guarantee a decrease in error rate, which is the primary goal in the design of a recognition system.

To alleviate some of the shortcomings of the ML criterion, such as its sensitivity to the form of the model, and its only indirect link to the error rate of the system, MCE has been proposed as a discriminative training procedure that minimizes a smooth approximation of the error rate [3]. Unlike ML, MCE does not attempt to fit a distribution, but rather to discriminate against competing models [3]; a model is simply viewed as a method to generate a score and is not required to be a probability. By minimizing a smoothed error count measure, MCE training is more directly aimed at reducing the recognizer's mistakes than ML and has been quite successful in increasing the performance of speech recognizers [4]. To our knowledge, MCE has received very limited application to handwriting.

In this paper, we describe an application of MCE training to HMM-based on-line handwriting recognition. We particularly address the issue of writer variability by applying MCE to allograph-based HMMs, using a character-based loss matrix to focus learning at the character level while enabling flexibility in writing style. Experimental results show that the MCE-trained system discriminates confusing characters in a more efficient manner than the ML-based system and shows a significant reduction in error rate compared to the baseline ML-based system.

2. HMM-BASED LEXEME MODELING

In order to cope with the variety in style of writing, it is advantageous to model character's allographs, known as lexemes, instead of characters [5]. Lexemes represent qualitatively different ways of writing a character, and differ in various features such as shape, pen lifts, and direction of the pen movements. By clustering instances of a character that are similar in writing style, writer variability can be efficiently managed. Lexeme can be generated manually or through clustering techniques on the feature space. Figure 1 shows examples of lexemes of a selected set of characters, written in isolation.

Each lexeme is modeled by an HMM. The topology of the HMM, defined as the number of states in the model, the number of mixtures per state and the allowed connections between states, is designed on a lexeme-by-lexeme basis. This non-uniform topology helps in dealing with the discrepancy between characters due

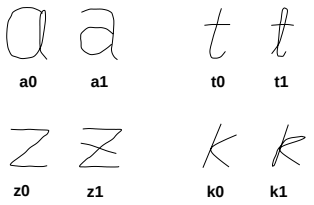

Fig. 1. Examples of lexemes for "a", "t'", "z", and "k".

to character's lengths or the number of strokes. For each HMM, the number of states in the model is the mode of the histogram of the number of frames of the corresponding lexeme, computed from training data. We used a Gaussian continuous-density, left-to-right HMM, with a two-state transition, for handling examples of characters that have a number of frames shorter than number of states in the model.

3. LEXEME-BASED MCE TRAINING

3.1. Lexeme-based MCE loss

We are given a finite number M of characters or words $C = \{C_1, ..., C_M\}$, where each C_i is modeled by a set of parameters $\Lambda_i = \{\Lambda_{il}\}$ for $l = 1, ..., L_i$. L_i is the number of lexemes of character C_i. Λ_{il} represents the HMM parameters of the l-th lexeme of character C_i. We also define $\Lambda = \{\Lambda_i\}$ for $i = 1, ..., M$, as the parameter space of the overall recognizer. Given a sequence of feature vectors $X = \{x_1, ..., x_T\}$ of length T, the lexeme-based MCE training procedure is implemented as follows.

First, to account for multiple instances of a character in the lexeme space, we define the discriminant function of character C_i as

$$g_i(X; \Lambda_i) = \left[\sum_{l=1}^{L_i} g_{il}(X; \Lambda_{il})^{\nu} \right]^{\frac{1}{\nu}} \tag{2}$$

where

$$g_{il}(X; \Lambda_{il}) = \log(P(X|C_i; \Lambda_{il}))$$

is the score of l-th lexeme's model of character C_i. ν is a positive constant. By choosing a large ν, the discrimination function term is dominated by the lexeme of maximum score. By varying ν, the contribution of each lexeme's score to the character discrimination function can be controlled.

The score of a lexeme $g_{il}(X; \Lambda_{il})$ is the log of the class-conditional probability of the corresponding HMM, given the input sequence X, and is typically computed by the Forward algorithm. However, in this paper, as in most HMM-based decoding schemes, the score was computed along the best path through the HMM states as found by the Viterbi algorithm. This has the advantage to simplify the computational load of the MCE algorithm. Decoding is done by choosing the character that has highest discrimination measure. That is,

$$\text{choose} \quad C_i \quad \text{if} \quad i = \arg\max_j g_j(X; \Lambda_j). \tag{3}$$

Second, assuming that X belongs to C_i, and to better cope with the non-uniform HMM topology across lexemes as well as the variety of character's lengths, we define the frame-length normalized misclassification measure of character C_i as

$$d_i(X; \Lambda) = \frac{\left[-g_i(X; \Lambda_i) + \log \left[\frac{1}{M-1} \sum_{j, j \neq i} e^{g_j(X; \Lambda_j)\eta} \right]^{\frac{1}{\eta}} \right]}{T} \tag{4}$$

with a large positive η to focus learning on the most competing categories. The sign of the misclassification measure indicates a correct or a wrong decision made by the classifier. A negative sign signifies a correct classification and a positive sign is an error. By choosing a large $\eta \to +\infty$, the misclassification measure is reduced to a difference in score between the best but incorrect category and the true category; learning becomes similar to a two-class classification at each iteration. Using a large η also reduces the computational load and makes the sign of the misclassification measure to better match the classification decision.

Finally, the MCE loss assigned to X is defined as $\ell(X; \lambda) = \ell(d_i(X; \lambda)$ where $\ell(\cdot)$ is a smooth approximation of the step-wise $0-1$ loss function, which is equal to one for positive values and zero otherwise. By choosing a large η and a monotonic function as a loss, the MCE loss function varies monotonically with the error rate of the system on a training data set. Among several available choices for the loss, we used the truncated sigmoid defined as

$$\ell(d) = \begin{cases} \frac{1}{1+\exp(-\alpha d)} & \text{if } d > \theta \quad (\theta \leq 0) \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

with a positive α. This loss enables learning to occur for confusable categories whose misclassification measures are above the threshold value θ. By varying θ, one can control the tolerable degree of discrimination during MCE training. A judicious choice of θ and α helps generalization. The MCE loss is an attempt to have an objective function that reflects the variations of the error rate and is smooth enough to allow minimization by standard gradient descent techniques.

3.2. Second-order optimization

Having defined the MCE loss, the next step is to minimize the expected loss $L(\Lambda)$, defined as a functional of the overall parameter set:

$$L(\Lambda) = \sum_{i=1}^{M} \int_{X \in C_i} \ell(X; \Lambda) P(X) dX \tag{6}$$

The minimization of this loss can be done using a stochastic gradient descent technique such as GPD [6], which updates the parameters of the system in a token-by-token basis. The GPD theorem guarantees convergence for an infinite run of the GPD adjustment; it has an asymptotic efficiency. In practice, MCE training focuses on the minimization of the MCE average loss defined over a body of training data of size N as

$$L_N(\Lambda) = \frac{1}{N} \sum_{i=1}^{M} \ell(X; \Lambda). \tag{7}$$

$L_N(\Lambda)$ has its values close to the empirical error rate, given a body of training data; it is a convenient method to monitor the MCE learning process and the performance of the system as learning proceeds. A version of GPD adapted to the use of a finite number of data is usually the preferred method of optimization.

Although GPD is a well-defined theoretical algorithm, it is extremely sensitive to accurate tuning of learning parameters such as the learning rate. Given a body of data and a smooth and differentiable empirical loss, one can use an optimization scheme less sensitive to the learning parameter issue. Gradient-based second order methods, such as the Newton algorithm, are an attractive choice, but they require the computation of the Hessian matrix. We used the quick-prop algorithm, which combines a gradient descent technique and the Newton algorithm, and uses an approximation of the Hessian matrix that does not require any extra computation [7]. The parameter set is updated as follows:

$$\Lambda_{\tau+1} = \Lambda_\tau - [\nabla^2 L(\Lambda_\tau) + \mu I]^{-1} \nabla L(\Lambda_\tau) \quad (8)$$

where μ is a learning rate. The Hessian matrix is assumed to be diagonal. For a parameter λ of the system, the Hessian is approximated by

$$\frac{\partial^2 L(\Lambda\tau)}{\partial^2 \lambda} \approx \frac{\frac{\partial L(\Lambda_\tau)}{\partial \lambda} - \frac{\partial L(\Lambda_{\tau-1})}{\partial \lambda}}{\lambda_\tau - \lambda_{\tau-1}}. \quad (9)$$

The true Hessian is positive definite, which is not strictly true for its approximation. The quickprop algorithm replaces the approximated Hessian by zero whenever its sign does not change at learning time τ and $\tau - 1$. This indicates that the Hessian is not sufficiently positive definite and the approximation cannot be trusted. Instead, a simple gradient descent update is used in this case. Compared to the GPD update, this algorithm, although heuristic, appears less sensitive to the learning parameters issue as it makes use of the Hessian. The parameters set is typically updated after an epoch, where an epoch is defined as one run over the entire data set.

4. EXPERIMENTS

We performed a set of experiments to compare Maximum Likelihood estimation to MCE training. The task consisted in classifying a 92-character set, which includes letters from the English alphabet, digits and keyboard symbols. The front-end of the system under consideration uses a digitizer that captures the successive pen-tip positions. After resampling, local position and curvature information are calculated and Principal Component Analysis is used to generate a sequence of nine-dimensional feature vectors.

The data consisted of 106,396 examples of discretely-written characters, written by 174 writers. The data were divided into two sets: 96,563 examples, written by 157 writers were used as training tokens and the remaining 9,832, written by 17 writers, were used as testing tokens. The writers in the training data and testing data are mutually exclusive. The set of 92 characters were manually clustered into 404 lexemes; the number of lexemes per character is not uniform. 8 Gaussian mixtures with diagonal covariance matrix was assigned to each HMM state with the topology of the HMM matched to the individual lexeme. The Maximum Likelihood criterion was implemented by the Segmental K-means algorithm [8], which considers only the most likely path in the re-estimation procedure.

Starting from the models generated by ML, we ran MCE training, monitoring learning using the MCE average loss. A typical MCE learning curve is displayed in Figure 2. The solid line shows the evolution of the error rate over the training set. The dashed line shows the average MCE loss. The x-axis displays the number of epochs. As can be seen from the figure, the MCE loss function and the error rate over the training data display similar variations. As learning proceeds, the number of mis-classified examples diminishes, and an increasing number of examples have a loss value of zero, resulting into a convergence of the two curves.

Training strategies	MLE	MCE
character-level/character-model	83.9%	88%
lexeme-level/lexeme-model	87.9%	90.9%
character-level/lexeme-model	N/A	91.7%

Table 1. Character recognition rate of the ML- and MCE-trained systems in the writer-independent, 92-character classification task.

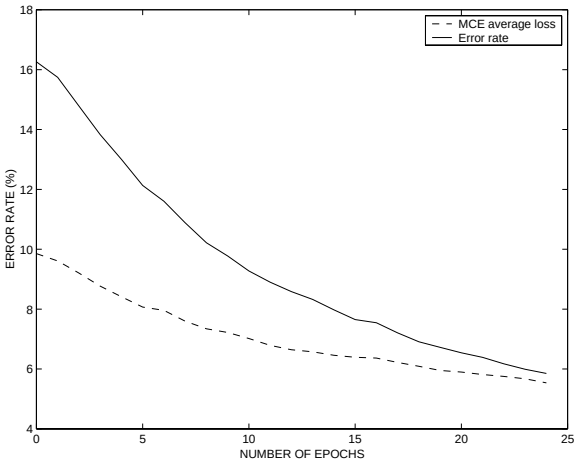

Fig. 2. MCE average loss and error rate as function of the number of epochs in training data.

We ran MCE within three schemes of training: a) character-level/character-model training where each character is modeled by a single HMM model and no lexeme clustering is performed. b) lexeme-level/lexeme-model training, where each character is modeled by a set of lexeme-based HMMs; each lexeme is an individual category and the focus is on minimizing the lexeme error rate. c) character-level/lexeme-model training, where each character is modeled by a set of lexeme-based HMMs but training focuses on minimizing the character error rate as described in the previous section. In all experiments, we set $\nu \to \infty$ and $\eta \to \infty$, which means considering the most representative lexeme of a character and the closest competing category at each MCE iteration.

Table 1 shows the overall performance of ML and MCE training in terms of character recognition rate. It is clear from these results that MCE out-performs ML in all strategies of training. Using character models, MCE improves the performance of the baseline ML system from 83.9% to 88%, resulting into a 25.4% error rate reduction. Using lexeme models, MCE improves the recognition rate from 87.9% to 90.9%, when trained at the lexeme level, and to 91.7%, when trained at the character level. This is equivalent to an error rate reduction of 24.8% for the lexeme-level/lexeme-model training strategy, and 31.4% for the character-level/lexeme-model training strategy.

Note that minimizing the lexeme error rate did improve character recognition. This result is explained by the observation that during lexeme-level MCE training, the competing lexemes usually

Character	Accuracy	Confusable character
0	43.08%	O (32%)
f	52.34%	F (11%)
,	60.71%	’ (21%)
.	61.67%	: (33%)
e	62.94%	E (28%)
=	70.49%	- (23%)
”	70.59%	’ (12%)
#	70.73%	t (12%)
+	70.83%	t (14%)
K	72.12%	k (25%)

Table 2. 10 worst-recognized characters by ML.

Character	Accuracy	Confusable character
O	56.14%	0 (26%)
0	58.46%	O (34%)
,	73.21%	’ (18%)
.	78.33%	: (20%)
e	79.02%	E (15%)
f	79.44%	F (8%)
G	80.00%	g (17%)
t	80.65%	+ (6%)
’	80.95%	, (11%)
;	82.50%	j (10%)

Table 3. 10 worst-recognized characters by MCE.

belong to different characters and there is little within-character competition. As can be expected, minimizing the character error rate directly with the use of lexeme-based models shows the best performance: an 8.8% improvement in character error rate over lexeme-level training.

Another conclusion that is drawn from these results is that making use of a set of lexemes to model a character is more efficient than character-based modeling. This is particularly true in the case of ML, where the use of lexeme models has improved the performance of ML from 83.9% to 87.9%, meaning a 24.8% error rate reduction compared to the single character-based modeling system. In the case of MCE training, using lexeme-based models, instead of single character-based models, means a 24.1% character error rate reduction when training is done at the lexeme level and 30.8% error rate reduction when training is done at the character level.

Also note that MCE training of single character-based models is similar in performance to ML training of lexeme-based models (88% compared to 87.9%). ML accuracy depends on better modeling of characters’ variability, which may require a higher number of parameters in the system, leading to a higher computational load and the need for more data. MCE seems to alleviate these requirements as it makes better use of available resources in order to reduce the error rate at the specified level of training.

For illustration, the accuracies of the ten worst-recognized characters by the ML-trained system and the character-level MCE-trained one are shown in Tables 2 and 3, respectively. The first column shows the characters, ordered by decreasing order of their accuracies in the testing set (the accuracies are shown in the second column). The last column shows the mostly detected character among wrong characters; the percentage of misrecognized examples are in parenthesis. Thus, from Table 2, 43.08% of number ”0” (zero) were correctly classified by the ML-trained system, while 32% of the misrecognized examples were recognized as the letter ”O”. Clearly, as can be seen from the table, MCE training has managed to increase discrimination in all cases, even between very similar characters such as the letter ”O” and the number ”0” (zero) or the comma and the apostrophe. How these results depend on the choice of the lexeme clustering algorithm, the number of lexemes, and the HMM topology, deserves further investigation.

5. CONCLUSION

We described an application of the Minimum Classification Error (MCE) algorithm to on-line handwritten character recognition. Comparison of the MCE to MLE in the task of classifying a 92-character set shows that MCE realizes more than 30% relative error reduction from the ML baseline and discriminates characters in a more efficient way than MLE does.

6. ACKNOWLEDGMENTS

The author would like to thank John Pitrelli, for revising the manuscript, Michael Peronne, Jay Subrahmonia, Gene Ratzlaff of IBM T.J. Watson Research Center.

7. REFERENCES

[1] K. S. Nathan, H. S. M. Beigi, J. Subramonia, G.J Clary, and H. Maruyama, “Real-time on-line unconstrained handwriting recognition using statistical methods,” *ICASSP*, vol. 4, pp. 2619–2623, 1995.

[2] A. Nadas, “A Decision Theoretic Formulation of a Training Problem in Speech Recognition and a Comparison of Training by Unconditional Versus Conditional Maximum Likelihood,” *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 31, no. 4, pp. 814–817, 1983.

[3] B.-H. Juang and S. Katagiri, “Discriminative Learning for Minimum Error Classification,” *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 40, no. 12, pp. 3043–3054, Dec. 1992.

[4] A. Biem, S. Katagiri, and B.-H. Juang, “Pattern Recognition based on Discriminative Feature Extraction,” *IEEE Transactions on Signal Processing*, vol. 45, no. 02, pp. 500–504, 1997.

[5] M. P. Perrone and S. D. Connell, “K-means clustering for Hidden Markov Models,” *Proceedings of IWFHR VII*, pp. 229–238, Sep. 2000.

[6] S. Katagiri, C-H. Lee, and B.-H. Juang, “New Discriminative Training Algorithms Based on the Generalized Descent Method,” in *Proc. IEEE Worshop on Neural Networks for Signal Processing*, 1991, pp. 309–318.

[7] S. E. Fahlman, “An empirical study of learning speed in back-propagation networks,” Technical Report CMU-CS-88-162, Carnergie Mellon University, 1988.

[8] B.-H. Juang and L. Rabiner, “The Segmental k-means Algorithm for Estimating Parameter of Hidden Markov Models,” *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 38, no. 9, pp. 1639–1641, 1991.