WRITER-ADAPTATION FOR ON-LINE HANDWRITTEN CHARACTER RECOGNITION

N. Matić, I. Guyon, J. Denker, and V. Vapnik
AT&T Bell Laboratories
Holmdel, NJ 07733

Abstract

We have designed a writer-adaptive character recognition system for on-line characters entered on a touch-terminal. It is based on a Time Delay Neural Network (TDNN) that is first trained on examples from many writers to recognize digits and uppercase letters. The TDNN without its last layer serves as a preprocessor to an Optimal Hyperplane classifier, that can be easily retrained to peculiar writing styles. This combination allows for fast writer dependent learning of new letters and symbols. The adaptation module is memory and speed efficient.

1 INTRODUCTION

With the emergence of "pen-based" computers and their first real world applications, the recognition accuracy requirements for on-line handwriting recognition have increased. Some users are ready to limit themselves to form filling applications and write only either digits or uppercase letters in boxes, provided the recognition accuracy gets close to 99%. In this paper we address the problem of isolated character recognition of digits and uppercase letters which is of more limited difficulty than cursive writing but is of practical interest.

Various approaches have been proposed in the literature to address this challenging but still unsolved problem [1]. Highly structured back-propagation neural networks, so called, Time Delay Neural Networks (TDNN) have been successfully applied to speech [2, 3]. Recently they have been developed for on-line character recognition [4]. By using this network, feature extraction and classification is performed within single trainable module. We trained a *writer-independent* TDNN with many examples of handwriting obtained from a large number of writers, in order to optimize performance for an average writer. But we found that the writer-independent approach has limitations since it is not capable of providing the required accuracy for any potential user. On the other hand a large number of existing and future applications of "pen-computing" are oriented towards individual users.

One of the main reasons why it is so hard to obtain the required accuracy for writer-independent systems is the large variety of writing styles. This is specially

true for on-line handwritten characters. Here, the dynamics of the handwriting bring another level of variability to the already existing explosion of different handwriting styles due to geographical, social and professional origin and to differences between left-handed and right-handed writers [5].

Solutions to obtain very high recognition accuracy include training the recognition system to be *writer-dependent* [6, 7], or introducing *writer-adaptation*. In writer-adaptive systems the writer starts using the writer independent system and provides more examples of his own handwriting while using it in order to fine tune the recognizer to his own style.

We have identified a few desired features for a good writer adaptive system:

- Fast adaptation phase, where the user needs to give only a few additional examples of his own handwriting.
- Possibility to easily add new classes.
- No recognition speed degradation.
- Little additional memory requirements, so that several versions of customized recognizers can be installed on the same system.

We designed our system with these requirements in mind. We propose a supervised adaptation scheme where most of the TDNN writer-independent recognizer is preserved. The modification is that the last layer of the TDNN is replaced with a tunable classifier. For example, the TDNN without its last layer can be combined with k-nearest neighbor classifier as in [8] . This adaptation scheme fulfills the first 2 of the above requirements, while others are only partially fulfilled. In our implementation the TDNN without its last layer is preprocessor to the Optimal Hyperplane classifier [9]. Optimal Hyperplane classifier is now last (classification) layer, that can be easily retrained for the adaptation purposes. Here the training set for the adaptation consists of: examples of rare writing styles and only a *small subset* of the original training set. The Optimal Hyperplane is a linear classifier, therefore, after the recognition system is customized, there is no recognition speed degradation compared to the writer-independent case. This approach satisfies all the above requirements, and provides fast and high accuracy adaptation.

The paper is organized as follows. In section 2 we describe the writer-independent TDNN, the core of our writer-adaptation system. In section 3 we describe the Optimal Hyperplane classifier, its training algorithm and its on-line adaptation. In section 4 we describe the experiments and present the results.

2 THE WRITER INDEPENDENT CORE OF THE SYSTEM

We now describe a Time Delay Neural Network (TDNN) which processes the pen trajectory by extracting and combining local topological features [4]. This is multi-layer

feed-forward network, trained to be *writer independent* with the backpropagation algorithm . Drawing actions are recorded by a touch-sensitive pad as a sequence of coordinates $[x(t), y(t)]$ regularly spaced in time. Characters are entered in disjoint boxes, therefore no segmentation is needed in the preprocessing stage.

2.1 Preprocessing

The preprocessing has been designed to preserve the temporal structure of the input data. The resulting representation is a sequence of vectors ordered with increasing time. The first steps of the preprocessing,*resampling, centering and rescaling*, significantly reduce the variability within classes. In particular time and scale distortions are removed. The resampled characters are represented as a sequence of points $[x(n), y(n)]$, regularly spaced on the trajectory, as opposed to the input sequence, which was regularly spaced in time. The last step of the preprocessing enhances the factors that help discriminate between classes by including information about local slope and the local curvature in the representation. The preprocessing results in a representation suitable for a TDNN recognizer. It consists of 90 feature vectors of 7 components encoding local information of the pen trajectory: pen-up or down, centered and normalized x and y coordinates, 2 components for local slope and 2 for local curvature [4].

2.2 The Network Architecture and Training

The representation produced by the preprocessor captures local topological features along the curve of the character. It is then used as the input of a neural network which is trained to make use of these simple local features and extract more complex, more global features. The neural network classifier, the TDNN is a feed-forward *layered* network, having four layers of weights connecting five layers of units (the input is counted as a degenerate "layer #0"). The connections are restricted to be *local*. A particular unit has a receptive field that is limited in the time direction; that is, all its inputs come from a group of consecutive feature vectors in the previous layer. The receptive field of unit i will have considerable overlap with the receptive field of unit $i-1$. This induces a topology on the input space, giving the network hint that it should be looking for sequences.

The network is further constrained to be *convolutional*. The weights connecting one feature vector in one layer to its receptive field in the layer below it are the same as the weights connecting the next feature vector to the next receptive field. The motivation for this is that we expect that a particular meaningful feature (e.g. a line or a curve) can occur at different times within the sequence. It also means that there are far fewer free parameters, which facilitates training and improves generalization [10]. The particular network, that we are going to use as a base for the adaptation has 39,016 connections, but only 7462 independent weights.

The final specialization of our network is that each layer has a coarser time representation than the proceeding layer. This is implemented by *subsampling* [10, 11]: only one every s values in the convolution is kept (and actually computed).

The loss of time resolution in the feature vectors (due to subsampling) is partially compensated by an increase in the number of features ("bi-pyramidal" scaling).The number of units, and therefore the information-carrying capacity, in each layer is reduced less than the subsampling alone might suggest. The output layer consists of only one feature vector of 36 coefficients (10 for digits and 26 for letters) that can be considered the ultimate , most abstract features, and provide scores fo each class. More details about the specifications of the network can be found in reference [4].

The network was trained on the *writer-independent* task (350 writers contributed to the training set, see section 4). It was trained to recognize either digits or uppercase letters. This means that the network was not penalized during training for confusions between digits and letters. During utilization we assume that contextual information is available to determine whether the character is a digit or a letter.

The weights are adjusted during a supervised training session, using back-propagation [12] which performs gradient descent in weight space with a mean-squared-error (MSE) objective function. The target values are binary. Before training, the weights are initialized with random values, uniformly distributed between $-2.4/F_i$ and $2.4/F_i$, where F_i is the total number of inputs to unit i. The learning rate is adjusted during training with the help of the diagonal approximation of the Hessian matrix [13]. The algorithm cycles through the training patterns and a modification of the weights occur at each presentation of a patterns (which is similar to a *stochastic gradient* procedure). Examples of different classes and from different writers should be alternated. Shuffling the examples is critical for the improvement of the speed of convergence.

The TDNN is the core of our adaptation system.

3 THE ADAPTATION SCHEME

3.1 Optimal Hyperplane Classifier Training Algorithm

For adaptation, we substitute the last layer of the TDNN with and Optimal Hyperplane (OH) classifier. In this section we provide a general analytical background for the OH classifier [9] that belongs to a class of linear decision rules. The OH classifier is parameterized by a *subset* of training patterns, so called "supporting patterns". It has a small effective capacity while at the same time it has good generalization capability [14].

For a two class problem the OH classifier is defined by a Hyperplane, whose direction $\mathbf{w}$ can be determined through a training procedure described in reference [9]. The hyperplane decision boundary is given by:

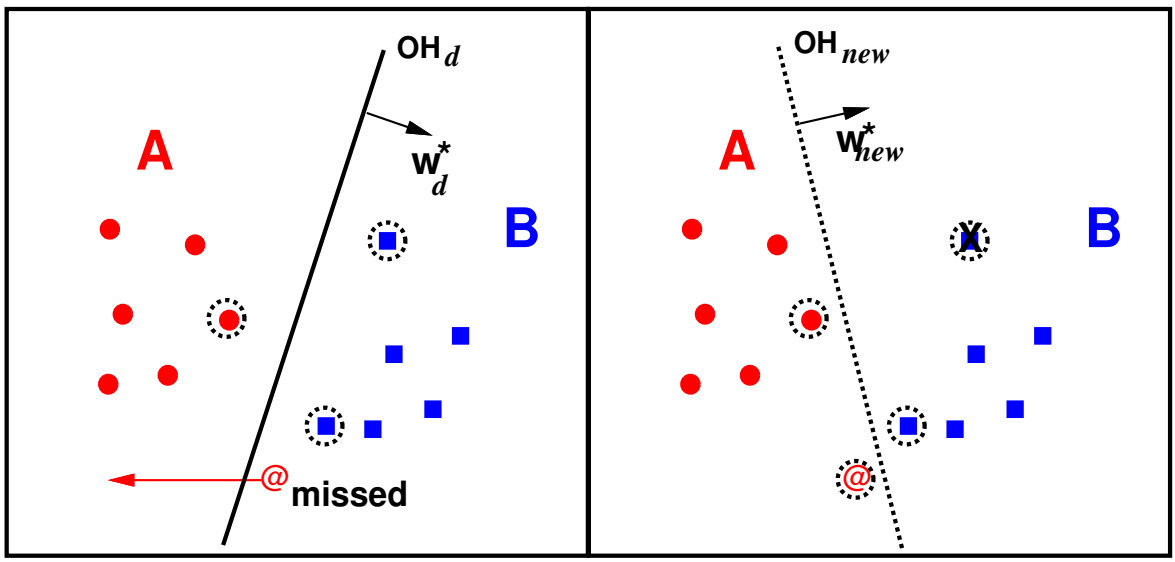

Figure 1: Optimal Hyperplane Classifier: a) Two class separation, "supporting patterns" are circled. The example "@" provided by a new user is misclassified. b) On-line adaptation, OH_{new} is generated. The set of "supporting patterns" is modified: the example "@" is introduced and the crossed pattern is removed.

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{1}$$

and separates a set of vectors $\mathbf{x}$, of the dimension n, that belong to two classes A and B. The input to the training algorithm is a set of p examples $\mathbf{x}_i$ with labels y_i:

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_p, y_p)$$

where $y_i = 1$ if $\mathbf{x}_i \in A$ and $y_i = -1$ if $\mathbf{x}_i \in B$.

Since there are an infinite number of equivalent classifiers which differ only by the norm of their weight vector $\mathbf{w}$, if the training set is linearly separable, there is a solution such that the inequalities:

$$\begin{aligned} \mathbf{w} \cdot \mathbf{x}_i + b \geq 1 \text{ if } \mathbf{x}_i \in A \\ \mathbf{w} \cdot \mathbf{x}_i + b \leq -1 \text{ if } \mathbf{x}_i \in B \end{aligned} \tag{2}$$

hold, for all the training patterns $\mathbf{x}_i$.

It can be shown [9] the the OH classifier, which maximizes the margins between training examples and a the class boundary, is solution of the optimization problem:

$$\min_{\mathbf{w}} \|\mathbf{w}\|^2$$

under the condition (2).

After training, in the *classification phase*, patterns are classified according to the rule:

$$\begin{cases} \mathbf{x} \in A & \text{if} \quad w \cdot \mathbf{x} + b \geq 0 \\ \mathbf{x} \in B & \text{if} \quad w \cdot \mathbf{x} + b < 0 \end{cases}$$

The minimization problem can be solved directly with numerical methods. Unfortunately that is impractical when the dimensionality of the input space is high. The minimization problem can be transformed in the dual space through Lagrangian:

$$L \quad = \quad \frac{1}{2}\mathbf{w}^2 - \sum_{i \in A} \alpha_i (\mathbf{w} \cdot \mathbf{x}_i + b - 1) + \sum_{i \in B} \beta_i (\mathbf{w} \cdot \mathbf{x}_i + b + 1) \tag{3}$$

where $\alpha_i \geq 0$ and $\beta_i \geq 0$ are Lagrange multiplier of the constraints (2) and they satisfy the conditions:

$$\begin{aligned} \alpha_i \cdot (\mathbf{w} \cdot \mathbf{x}_i + b - 1) = 0, \text{ for } \mathbf{x}_i \in A \\ \beta_i \cdot (\mathbf{w} \cdot \mathbf{x}_i + b + 1) = 0, \text{ for } \mathbf{x}_i \in B \end{aligned} \tag{4}$$

The optimization problem is now equivalent to finding the maximum of the Lagrange function with respect to parameters $alpha_i$ and β_i. It can be shown that the minimal in absolute value vector $\mathbf{w}$ is given in the form:

$$\mathbf{w}^\star \quad = \quad \sum_{i \in A} \alpha_i \mathbf{x}_i - \sum_{i \in B} \beta_i \mathbf{x}_i \tag{5}$$

where the summation runs only on the supporting patterns $\mathbf{x}_i$ that satisfy the equality:

$$\mathbf{w} \cdot \mathbf{x}_i + b = y_i \ . \tag{6}$$

The very important feature, that we make use of for our adaptation scheme described in the next section, is that the OH solution is a function only of the *supporting patterns*, which are only a small fraction of the total number of training patterns. In (5) all other patterns have zero coefficients α_i and β_i. The supporting patterns are the patterns that are closest to the decision boundary (see figure 1a). The capacity (or VC-dimension) of the OH classifier is smaller, or at most equal to the dimension of the input vector $\mathbf{x}$ [14]. Computational complexity of the classifier is proportional only to the number of supporting patterns. The practical consequences are that the computational and memory requirements for this classifier are small.

The problem of on-line handwritten character recognition involves patterns from more than two classes and it can be solved with multiple hyperplanes.

3.2 The on-line adaptation version

The property that the OH is expressed as a linear combination of the supporting patterns only, is suitable for the adaptation purposes.

First we are going to explain the adaptation process on the simple 2 class example. Let us assume that an OH is generated (as described in 3.1), to separate two classes A and B. (see figure 1a). We call such hyperplane and the set of its supporting patterns "default "(OH_d). Its direction is specified by a vector $\mathbf{w}_d^\star$:

$$\mathbf{w}_d^\star \ = \ \sum_{i \in A-default} \alpha_i \mathbf{x}_i - \sum_{i \in B-default} \beta_i \mathbf{x}_i \tag{7}$$

Let us assume that a new example of class "A" is presented for classification. If it is misclassified (see figure 1a), then the retraining scheme generates OH_{new}. The training set for OH_{new} consists of "default" supporting patterns augmented by the new example. The weight vector is initialized to $\mathbf{w}_d^\star$. The resulting OH_{new} is a linear combination of a "new" set of supporting patterns, that will include the new example. Its direction is specified by the vector $\mathbf{w}_{new}^\star$:

$$\mathbf{w}_{new}^\star \ = \ \sum_{i \in A-new} \alpha_i \mathbf{x}_i - \sum_{i \in B-new} \beta_i \mathbf{x}_i \tag{8}$$

The OH_{new} now represents default classifer for potential new misclassified pattern.

In the case of the K-class problem, we use K OH classifiers, each separating one class from all the other ones. For each character that is misclassified by the system, the following takes place: Assume that the misclassified character x has desired label

X. Assume that the answer of the recognizer when the character was presented was Y. Let the corresponding default OH classifiers be OH_d^X and OH_d^Y. The OH_d^X is retrained as described in 3.2, with character x as a positive example. The OH_d^Y is retrained with the same character as a negative example. The resulting hyperplanes are OH_{new}^X and OH_{new}^Y.

4 EXPERIMENTS AND RESULTS

Our adaptation system consists of (see figure 2):

- The writer independent TDNN described in the section 2 without its last layer. In the following, this network will be referred as the writer-independent core (WIN-core). The WIN-core is used as a preprocessor that provides high level feature representation.
- A a set of 36 OH classifiers replacing the last layer of the TDNN. Initially 36 OH are generated for the writer independent task. They are split into two groups: 26 hyperplanes are trained to separate one upper-case letter from other upper-case letters; 10 hyperplanes are generated to separate one digit class from all other digit classes. The maximum output in the field designated by contextual information indicated the classification result.

When the pattern is presented, it is first transformed to abstract internal representation provided by the WIN-core. That is, a vector with the dimension 120. The transformed pattern is then presented to the last layer (i.e. Optimal Hyperplanes) for final classification. In case of classification error, the user is asked to provide the desired label and adaptation takes place, as described in the previous section. If an example of the non-predefined class (i.e. unknown to the WIN) is presented to the system, a new OH for that class is initialized with that pattern as a positive example. Later, it is adapted as any other hyperplane.

Training of the writer-independent TDNN (see section 2) was performed on the database collected among the staff at $AT\&T$. The training set consists of 16,000 examples from 350 writers. The TDNN core was then used as a preprocessor for the OH classifiers which were trained with the same data. The resulting classifiers had a total of xxx supporting patterns which were store in the WIN-core representation (of 120 coefficients).

Adaptation tests were performed on data provided by seven writers. Characters include uppercase letters and digits on which the WIN was trained, and 7 new symbols $\{+, -, *, /,), (, =\}$ unknown to the WIN. Whenever character was missed, it was used together with its label to fine tune the corresponding hyperplanes. The test was performed with 450 characters per writer. Results are shown in figure 3. The total number of errors since the adaptation started is plotted against the number of examples presented for the adaptation. The slope of the curve gives the estimate

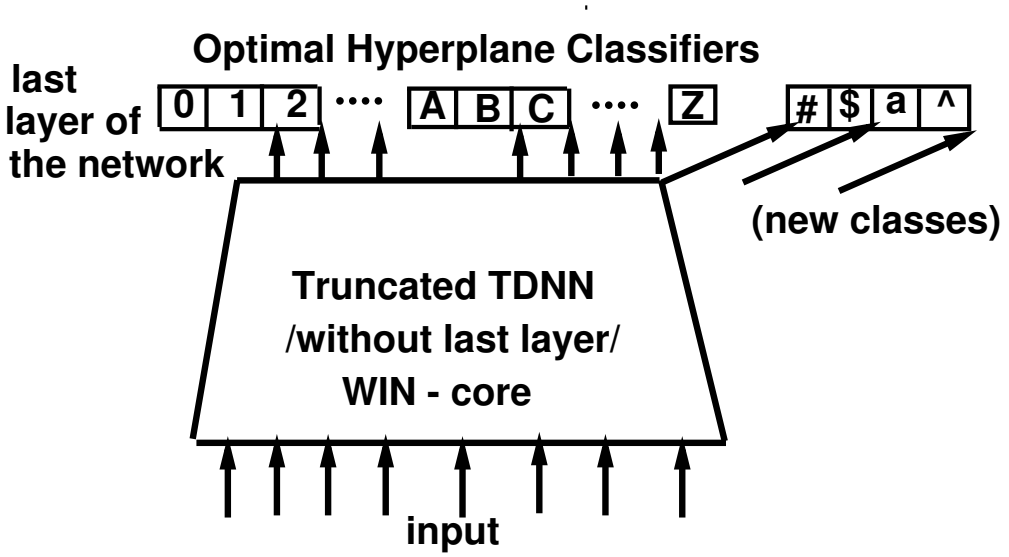

Figure 2: Writer adaptation system

of the error rate. The slope decreases (as the system adapts to the writer) as adaptation progresses. At the end of adaptation, the error rate averaged over the seven writers is 2.5%. One can notice that there are important differences between writers, some reaching easily less than 1% error and some remaining around 4% error. The total number of examples needed to learn the 7 new symbols and the writer dependent variants of uppercase letters and digits never exceeded 40 in this experiment.

5 CONCLUSIONS

A writer adaptation system was presented. It consists of two components. First, a preprocessor which stores the new pattern in compact representation provided by the TDNN without its last layer. Second a linear Optimal Hyperplane classifier, which can easily adapt to rare writing styles, or to new symbols, by changing its default set of "supporting patterns" into a new set, that reflects both good properties of the writer independent training set, as well as the new examples. Only a few new samples are required to learn the new symbols or atypical patterns. Since we replace the last layer of the TDNN by another linear classifier, there is no recognition speed degradation compared to original writer-independent recognizer.

Figure 3: Cumulated number of errors

6 ACKNOWLEDGMENTS

The authors thank all the Neural Network research group headed by L. Jackel at Bell Labs, Holmdel, for their suggestions regarding this paper and in particular L. Bottou and C. Cortes for the programs for Optimal Hyperplane Training. The simulations were performed using Neural Network simulator SN of L. Bottou and Y. Le Cun. The database for training the writer independent network was provided by the Touch Entry Terminal software development group in *AT&T* Columbus. The data to test the writer adaptation capabilities was provided by the character recognition group at IBM Watson Research Center.

References

[1] C. C. Tappert, C. Y. Suen, and T. Wakahara. The state of the art in on-line handwriting recognition. *IEEE trans. on PAMI*, 12(8):787–808, 1990.

[2] K. J. Lang and G. E. Hinton. A time delay neural network architecture for speech recognition. Technical Report CMU-cs-88-152, Carnegie-Mellon University, Pittsburgh PA, 1988.

[3] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37:328–339, March 1989.

[4] I. Guyon, P. Albrecht, Y. Le Cun, J. Denker, and W. Hubbard. Design of a neural network character recognizer for a touch terminal. *Pattern Recognition*,

24(2), 1991.

[5] J.R. Ward and T. Kuklinski. A model for variability effects in handprinting with implicatio ns for the design of handwriting character recognition systems. *IEEE Trans. Sys. Man. Cyber.*, 18:438–451, 1988.

[6] C. C. Tappert. Speed, accuracy, flexibility trade-offs in on-line character recognition, IBM Research Report RC13228, 1987.

[7] E. Mandler. Advanced preprocessing technique for on-line script recognition of nonconnected symbols. pages 64–66, 1987.

[8] I. Guyon, D. Henderson, P. Albrecht, Y. Le Cun, and J. Denker. Writer independent and writer adaptive neural network for on-line character recognition. In S. Impedovo, editor, *From pixels to features III*, Amsterdam, 1992. Elsevier.

[9] V. Vapnik. *Estimation of Dependences Based on Empirical Data.* Springer-Verlag, New York, 1982.

[10] Y. Le Cun. Generalization and network design strategies. In R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, editors, *Connectionism in Perspective*, Zurich, Switzerland, 1989. Elsevier.

[11] L.-Y. Bottou. Master's thesis, EHEI, Universite de Paris 5, 1988.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel distributed processing: Explorations in the microstructure of cognition*, volume I, pages 318–362. Bradford Books, Cambridge, MA, 1986.

[13] S. Becker and Y. Le Cun. Improving the convergence of back-propagation learning with second-order methods. Technical Report CRG-TR-88-5, University of Toronto Connectionist Research Group, September 1988.

[14] B. Boser, I. Guyon, and V. Vapnik. An algorithm for optimum margin classifiers. In *Proceeding of the ACM workshop COLT'92*, Pittsburg,PA, 1992. IEEE.