On-Line Cursive Script Recognition using Time Delay Neural Networks and Hidden Markov Models

M. Schenkel†, I. Guyon‡, D. Henderson‡

† Swiss Federal Institute of Technology, CH-8092 Zürich
‡ AT&T Bell Laboratories, Holmdel, NJ 07733

Abstract

We present a writer independent system for on-line handwriting recognition which can handle a variety of writing styles including cursive script and hand-print. The input to our system contains the pen trajectory information, encoded as a time-ordered sequence of feature vectors. A Time Delay Neural Network is used to estimate *a posteriori* probabilities for characters in a word. A Hidden Markov Model segments the word in a way which optimizes the global word score, taking a dictionary into account. A geometrical normalization scheme and a fast but efficient dictionary search are also presented. Trained on 20k words from 59 writers, using a 25k word dictionary we reached a 89% character and 80% word recognition rate on test data from a disjoint set of writers.

Keywords: Handwriting Recognition, Neural Networks, Cursive Script, Hidden Markov Models, Dictionary Search.

1 Introduction

Pen interfaces should replace advantageously both mouse and keyboard in a variety of situations. Users who must use their computer while standing or sitting in a narrow space are primarily interested by this upcoming technology. In addition, for personal communicators and personal organizers, pen interfaces would help people who are unfamiliar with computers overcome their fear of the machine. Some computer manufacturers are already offering small computing devices with a pen interface. But, as reported in the medias, users are often disappointed by the limitations of handwriting recognition and find pen computers more a curiosity than a useful tool. It is thus important to propose new approaches to accurately recognize a large set of handwritten symbols, including all standard keyboard characters, independently of variations in writing styles.

It is interesting to notice that the recognition accuracies that have been reported in the literature, over the past 30 years, are invariably in the range 80-90% correct word recognition. The real progress in all those years has not been made in recognition rate but rather in the degree of distortion of the writing the system can deal with, the size of the dictionaries used and the degree of writer independence [1, 2, 3].

Early systems made many simplifying hypothesis about character segmentation [4, 5, 6]. Some of the major advances made since were the idea of combining segmentation and recognition [7], using trainable and not fixed features [8, 9] and the use of HMM's [10].

In a previous paper, we reported results on the recognition of run-on hand-printed words [11]. In this paper we are extending our method to the problem of cursive handwriting. The present study shows the versatility of our system which is suitable both for hand-printed and cursive styles. The target for our system is to provide acceptable walk-up recognition accuracy without forcing the writers to go through a training session and without imposing restrictions to the writing style or stringent limitations on the size of the vocabulary.

1.1 Problem description and system overview

The difficulty of recognizing natural handwriting comes from the inherent ambiguity of some handwritten characters and the difficulty of separating connected characters for which multiple segmentation hypothesis

are equally likely (Figure 1). This is aggravated by the fact that handwriting is a means of communication between people who usually share the same language. Therefore, writers assume that readers will be able to guess missing or distorted characters from the context.

To read natural handwriting, human readers perform *recognition and segmentation simultaneously* using various *contextual information*, including their language knowledge. Our system is a combination of Neural Network and Hidden Markov Model [12, 13, 14] which mimics this strategy. We avoid making any premature decisions with respect to recognition or segmentation. Our preprocessing is very conservative and, in particular, it preserves the time ordered nature of the raw data. The Neural Network has several layers of feature detectors which progressively extract relevant features for segmentation and recognition. Using the scores of the Neural Network and the constraints of a dictionary, the Hidden Markov Model post-processor makes the final recognition and segmentation decisions.

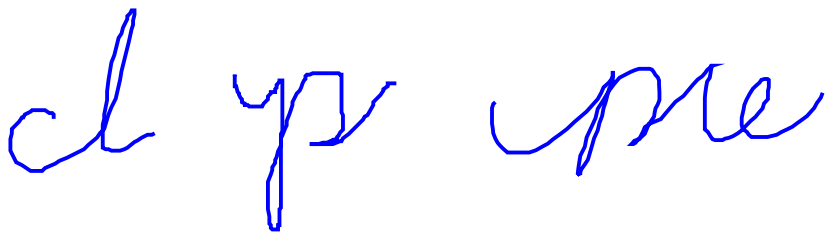

Figure 1: **Ambiguity of cursive handwriting:** Examples showing segmentation ambiguities that cannot be resolved without context: *cl/d*, *ys/p* and *pre/me*

Our Neural Network/Hidden Markov Model system has many parameters which are adjusted during a training session. The performance of this system relies on the availability of large training data sets. We are currently using data collected in our laboratory with a Wacom pad HD-648A (508 points per inch, 200 points per second).

For training data, we collected 26,931 words from 59 paid donors. For test data, we collected 607 word from 25 volunteers who had not participated in the training data collection. Neither data set was cleaned. Approximately 2.5% of the words cannot be recognized correctly by a human reader. Figure 2 shows some samples of the data used and the variety of handwriting quality.

2 Preprocessing

Our preprocessor generates a time ordered sequence of low level features, as described in our previous papers [15, 16]. We introduce novel features needed for cursive handwriting recognition.

2.1 Normalizations and filtering

The input signal, a sequence of x and y coordinates sampled at regular time intervals, is first normalized and filtered. Such operations are performed on entire words so that individual characters keep their relative position and size.

1. **Size normalization and baseline detection.** A word has three different areas. An *upper area* for ascenders, a *median area* (occupied by all characters) and a *lower area* for descenders. The lower limit of the median area gives us an estimate of the baseline and the height of the median area gives us an estimate of the scale. We subtract the baseline, correct the orientation and scale the word to a standard median size.

   First, we perform a linear regression through all the points (Figure 3-a) and correct the word orientation accordingly (Figure 3-b). This correction is usually beneficial for long words, but may have adverse effects on short words. Therefore, we skip this step for words shorter than 4 characters. The number of characters is estimated with the number of line crossings, which averages to 2.8 ± 0.6 crossings per character in our data.

   After detecting and removing the delayed strokes such as diacritical marks (see Section 2.2), the local minima and maxima of the y-coordinate get extracted. Two linear regressions through the minima

Figure 2: **Some typical examples of the data used:** The words are: indent, klaxon, cyclorama, illiterate, distribution, off, jacksonian, furnish, davison, choral, crux, wind, bottle, inquisition

and maxima are performed with the constraint of having a common slope. The procedure is repeated once more with the least fitting points removed. This results in a second orientation correction. The difference in the y-coordinate of the two regressions is our estimate of the size of the median area which is used to scale the word (Figure 3-c).

2. **Smoothing.** The pen trajectory is also slightly smoothed (Figure 3-b) to remove jitter and facilitate some preprocessing steps, including the extraction of local minima and maxima of the y-coordinate above described. The low-pass cut-off frequency depends on the estimated number of points per character to compensate for different writing speeds. As low-pass filter we use a moving-average filter of length 1/20 of the number of points per character (on average the length is 4 points).

3. **Resampling.** Because people write at different speeds and sometimes hesitate or stop writing, it is desirable to enforce some invariance under variations of the speed of the pen. Initially, points recorded by the pad are regularly spaced in time. Using linear interpolation, we replace them by points regularly spaced in arc length. Our sampling step is one fifth of the size of the middle area of the word. Consequently, all characters of similar shape have approximately the same number of points no matter at which speed they are written. Pen up segments are also linearly interpolated such that corresponding segments are approximately aligned, no matter whether the pen was lifted or segments were retraced [15, 11]. Since the speed might carry information which is useful for segmentation, we encode the speed information independently as an extra feature, as explained in the next section.

4. **Removal of average horizontal displacement.** The x-coordinate is high-pass filtered to remove its constant increase during the writing of a word. This keeps characters in within the same x-

coordinate range from the beginning to the end of the word. The high-pass filter subtracts a moving-average of the signal from the original data (filter length 25 points). This is equivalent to subtracting the position of a cursor moving at the average horizontal speed of the hand.

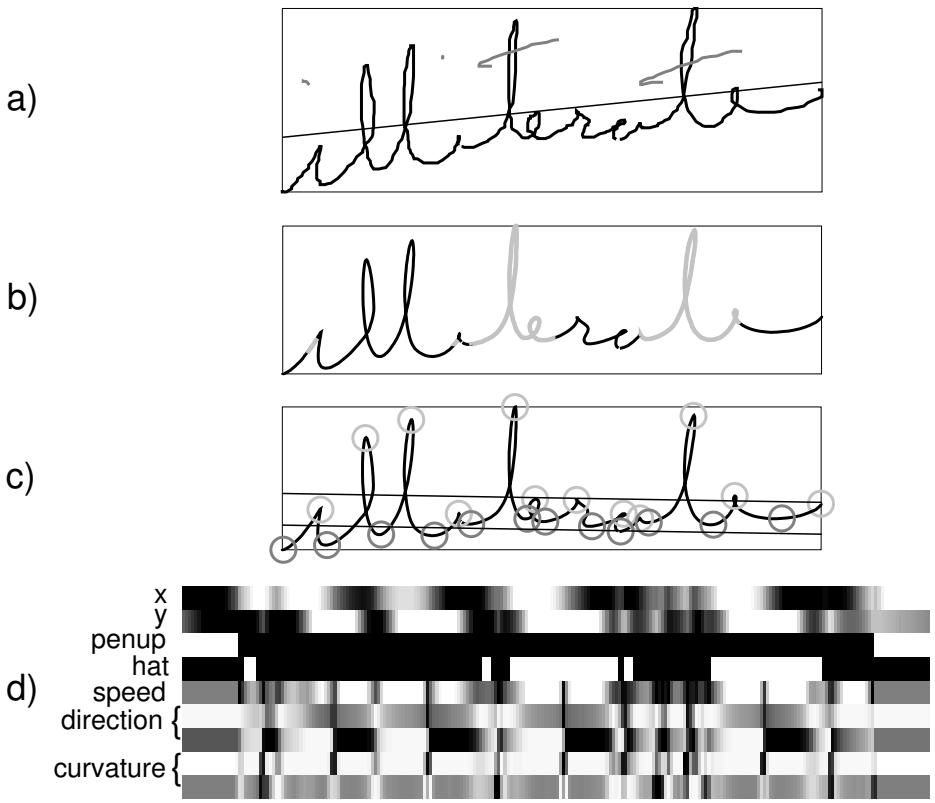

Figure 3: **The Preprocessing: a)** The original data with the linear regression through all points. **b)** After the subtraction of the linear regression and smoothing. The diacritical features are removed and the *hat* feature is set instead. The gray line indicates points that have the *hat* feature set. **c)** The constrained linear regressions through all local maxima and minima of the y-coordinate. **d)** The final representation: from left to right is the time axis, from top to bottom are the features.

2.2 Feature extraction

The second step of our preprocessing is the extraction of local topological features along the pen trajectory, yielding a sequence of time ordered feature vectors (Figure 3-d). The chronology of the preprocessing steps is important for the extraction of some features and will be mentioned where necessary. The features are not presented in chronological order.

1. x **coordinate.** The x position of the pen after normalizations and high-pass filtering. The high-pass filtering is the very last step of our preprocessing.

2. y **coordinate.** The y position of the pen after normalizations.

3. *penup* **feature.** A binary feature indicating whether the pen is touching the pad or not. The trajectory when the pen is in the up position is linearly interpolated.

4. *hat* **feature.** Diacritical marks such as i dots, t bars and x slashes are often written after the whole word was written. This is a potential problem for time ordered representations. Removing delayed diacritical marks [8] or trying to reinsert them at a better position in the stroke sequence [9] are two standard procedures to address that problem. We introduce here a new encoding of delayed diacritical marks which is more compatible with our representation: an additional binary feature called "hat" indicates whether a given point along the trajectory was later horizontally covered by a diacritical mark (see grey parts of the trajectory in Figure 3-b). The "hat" feature is extracted at the beginning of our preprocessing.

5. *speed* **coordinate.** The instantaneous speed of the pen measured by $\Delta s/\Delta t$ where Δs is the elementary arc length and $1/\Delta t$ is the sampling rate. The speed is calculated before resampling and then linearly interpolated during resampling.
6&7. Directional features. The direction of the pen trajectory is encoded with two features: $\Delta x/\Delta s$ and $\Delta y/\Delta s$ indicating the cosine and sine of the angle between the horizontal and the elementary displacement. Directional features are extracted after resampling.
8&9. Curvature features. The curvature of the pen trajectory is also encoded with two features which are the cosine and sine of the angle between two consecutive elementary segments. These can be simply derived from the directional features, using trigonometric formulas [15].

3 The Neural Network

Segmentation and recognition are performed with estimates of character interpretation probabilities provided by a Time Delay Neural Network (TDNN). We keep this section short because the TDNN and the training method used are essentially the same as the one we proposed in [11], where further details can be found.

3.1 Network Architecture

The TDNN (Figure 4) is a multi-layer feed-forward network, the layers of which perform successively higher-level feature extraction and produce scores for each class which can be interpreted as probabilities of presence of a given character in the input field. TDNNs, which were previously applied to speech recognition, are well suited to sequential signal processing [17, 18]. This allows us to use a representation which preserves the sequential nature of the data, in contrast with other approaches based on pixel-map representation.

A given neuron detects a particular local topological feature of the pen-trajectory. It performs a weighted sum of its inputs followed by a non-linear squashing function (a hyperbolic tangent). Its receptive field is restricted to a limited time window. The same neuron is reused along the time axis to detect the presence or absence of the same feature at different places along the trajectory ("weight sharing" [19]). Except for the neuron non-linearity, this operation is a convolution using the neuron weights as kernel coefficients. By using several neurons at each time step, the Neural Network performs the detection of different features.

The last layer of the TDNN consists of as many neurons as there are classes C_i. In our case we use 26 output features, providing scores for all the letters of the Latin alphabet (see Figure 5). After the training of the weights, the output activities corresponding to an input window X can be interpreted as *posterior* probabilities $P(C_i|X)$.

For isolated character recognition, the time contraction reduces the output to a single score vector. For word recognition the network is replicated in time and the output is a sequence of score vectors providing estimates of the *posterior* probabilities $P(C_i|X_t)$ for each character category C_i given the input window X_t at every time step t.

Because of the convolutional structure of the TDNN, there are many identical computations between two successive calls of the recognizer. Therefore, the TDNN is computationally efficient for this task. Only about one sixth of the network connections are reevaluated for each new input window.

The network used has 4 convolutional layers, an undersampling factor of 3 and a total of 8,008 independent weights. The detailed architecture is described in Table 1. Design criteria of the network are described in [15, 16].

input window length	klen / #kfeat 1^{st} layer	klen / #kfeat 2^{nd} layer	klen / #kfeat 3^{nd} layer	klen / #kfeat 4^{nd} layer	#indep. weights
50	14 / 9	8 / 15	4 / 20	3 / 23	8,008

Table 1: $TDNN$ **specifications.** Kernels have dimensions $klen$ by $\#kfeat$. Where $klen$ is the length in time and $\#kfeat$ is the number of features. (see Figure 4).

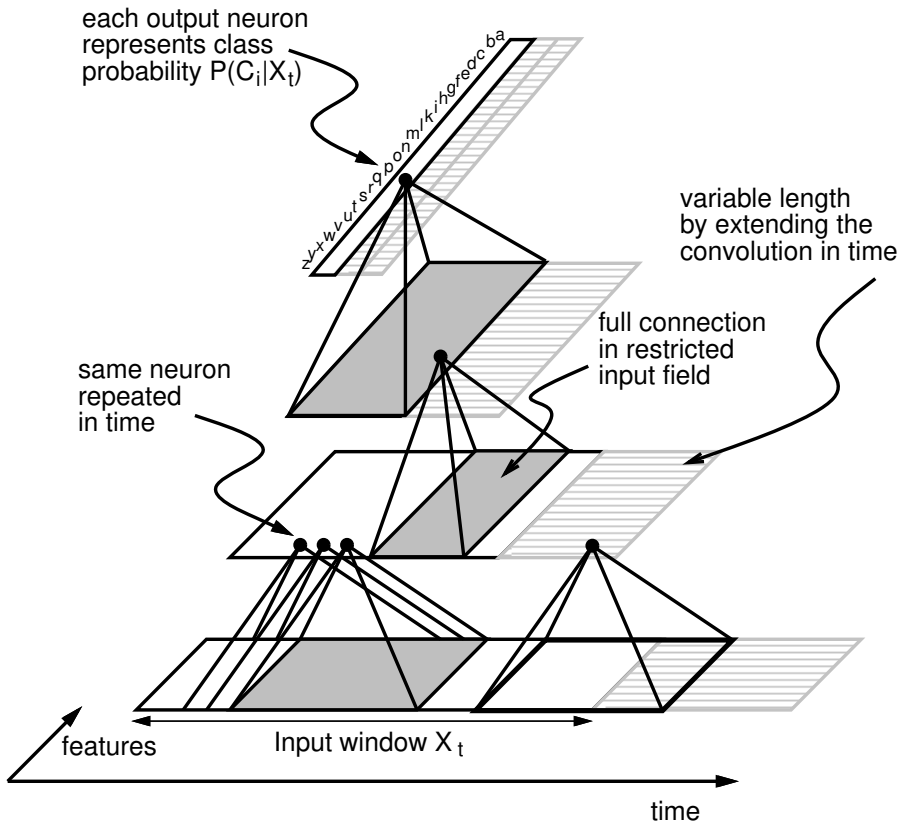

Figure 4: **Architecture of the Time Delay Neural Network (TDNN).**
The connections between layers obey the following rules (not all neurons are represented):
- neurons are feature detectors with restricted input fields, limited in the time direction,
- in each layer, a set of neurons scans the output of the previous layer along the time axis, and produces higher-level feature vectors,
- the sequence of feature vectors is subsampled in time at each layer to obtain time contraction.
The output is a sequence of vectors representing character posterior probabilities $P(C_i|X_t)$ at the corresponding time steps t.

3.2 Training procedure

The weights of the Neural Network are adjusted during a *supervised* training session, using the "back-propagation" algorithm [20] which performs gradient descent in weight space. As opposed to *unsupervised* training techniques [8, 9] which discover features independently of their relevance to the recognition problem, *supervised* training adjusts the neuron weights directly to achieve the overall goal of discriminating between character categories.

Our network is trained on entire unsegmented words with the position invariant training method proposed by Keeler et al. (1991). TDNNs trained solely on isolated characters perform poorly on unsegmented data because they give unpredictable scores for character transitions [11]. With the technique that we use, the network is trained both on valid characters and character transitions.

During training, we add a layer which transforms the outputs of the TDNN into a single score vector indicating presence or absence of a given character anywhere in the input field. The presence indicator of character C_i is computed as:

$$P(C_i) = 1 - \prod_t (1 - P(C_i|X_t))$$

which approximates the probability that character C_i appears *at least once* in the input field. This allows us to train the network with one target vector containing only informations about presence or absence of a character anywhere in the input field.

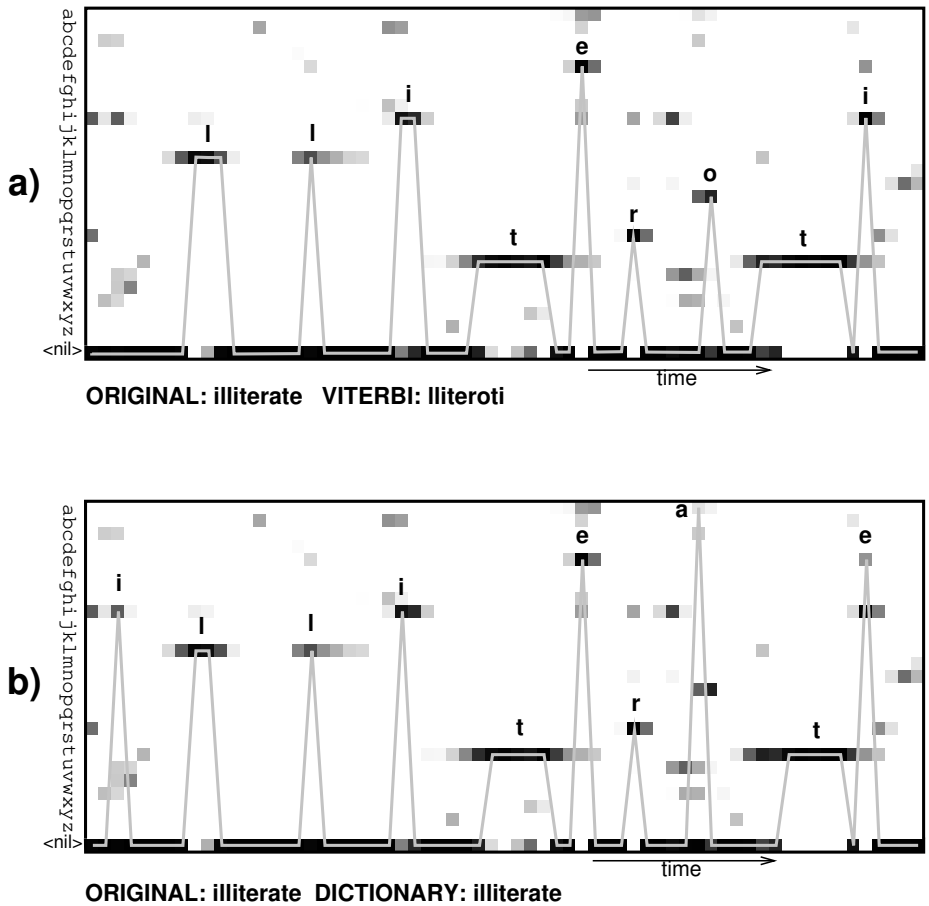

Figure 5: **Output character probability map of the TDNN:** Each box represents $P(C_i|X_t)$, the probability that character C_i is present in the window of input X_t (the darker the box, the higher the probability). The lowest row shows probabilities for the *nil*-class (absence of character). The light gray line indicates the best path of the HMM **a)** without dictionary. **b)** with dictionary constraints (the word "illiterate" is recognized).

Although this target vector contains no character order information anymore, position guessing is enforced by the architecture of the TDNN and the redundancy of the data. It self-organizes the activities of the units in a competitive fashion to infer the positional information and approximates the *posterior* probabilities $P(C_i|X_t)$ (see [21] for details). Therefore no manual segmentation is needed, word labels can be used directly.

Because the neurons have restricted input fields and are reused in time by "weight sharing", there are far fewer weights (i.e. free parameters) in this architecture than in fully connected networks. This has two desirable consequences: overfitting is prevented and less memory is needed to store the network. Weight sharing is enforced during training by averaging the weight updates of the various replica.

Training is stopped when the training character error rate is constant for several cycles.

4 Postprocessing

As postprocessor we use a set of Hidden Markov Models to model the character durations. We explain here just briefly the main idea of HMMs and discuss their use in our system in combination with a dictionary. For a good review on HMMs see [22].

4.1 Hidden Markov Models

Hidden Markov Models (HMM) have been widely used in speech recognition for almost ten years and since recently have also been applied to handwriting recognition [23, 24]. Their success is based on their ability to process sequential data with time variabilities in a statistical manner.

In the HMM framework it is assumed that the data observed are generated by a double random process. A HMM is a finite state machine for which: (i) each state S_k emits observations X according to a probability distribution $P(X|S_k)$; (ii) the machine changes states according to a state transition matrix $P(S_k|S_h)$.

In an HMM model of handwriting, the pen-trajectory is represented as a sequence of time ordered observations $X_1, \ldots, X_T$, e.g. the sequence of feature vectors we described in Section 2. States are concept classes such as sub-character shapes or characters.

Recognition consists in identifying the state sequence $S_{k_1}, \ldots, S_{k_T}$ which has produced the observation sequence $X_1, \ldots, X_T$ with highest probability. Under the assumptions of a first order Markov process and the independence of X_t on other inputs and states other than the current state S_{k_t}, the joint probability of the observation sequence and the state sequence is the product of all emission and transition probabilities:

$$P(X_1, \ldots, X_T; S_{k_1}, \ldots, S_{k_T}) = \prod_{t=1}^{T} P(S_{k_t}|S_{k_{t-1}})P(X_t|S_{k_t}) \tag{1}$$

with the convention $P(S_{k_1}|S_{k_0}) = P(S_{k_1})$. The maximum of this expression is easily found by the Viterbi algorithm [22]. By matching a sequence of observations and a sequence of states with this technique we simultaneously solve the segmentation and the recognition problem.

HMMs are organized in a hierarchical manner: characters are modeled by one or several states; words consist of several interconnected character models; sentences are built similarly from word models. Grammars impose rules upon the connections of the HMMs.

We introduce the *nil* category to model character transitions and to allow us to distinguish between letter-duration and letter-repetition. The observation probability for *nil* is:

$$P(C_{nil}|X_j) = 1 - \sum_{i=1}^{N} P(C_i|X_j)$$

which represents the probability to find no character at all at a certain time step t. Since we cannot normalize the posterior probability estimates obtained by the TDNN (otherwise $P(C_{nil}|X_j)$ would always be 0), we set the probability for the *nil*-class to zero when it would become negative.

In Figure 7-a, a simple grammar forces a word model to follow a predefined sequence of characters. In that case, the HMM provides only the segmentation and assigns a score to the corresponding word.

In Figure 7-b, the unconstrained grammar allows the recognition of any character sequence. The category *nil* represents character transitions. It is possible to change from any character to any other character via the *nil*-model.

Using several word models such as the one of Figure 7-a, a grammar for lexical checking can be derived as illustrated 7-c. In that case, the HMM chooses the word obtaining the best score in a set of dictionary words. Models for parsing entire sentences are derived along the same lines.

Training a HMM consists in first selecting a model for the distribution $P(X|S_k)$, then adjusting its parameters and calculating the coefficients of the transition matrix $P(S_k|S_h)$. This is often done with the Baum-Welsh algorithm or with segmental k-means [22]. In standard HMM, $P(X|S_k)$ is modeled by mixtures of multivariate Gaussian densities. Several authors report that better results can be obtained by using Neural Networks to estimate $P(S_k|X)$ because Neural Networks are good discriminant functions [25, 26]. $P(S_k|X)$ is substituted to $P(X|S_k)$ using Bayes' inversion rule.

In our hybrid system TDNN/HMM the observation probabilities $P(S_k|X)$ are estimated by the TDNN, trained with the method explained in Section 3.2 and the transition probabilities $P(S_k|S_h)$ are estimated independently as explained in the next section.

4.2 Duration modeling

The particular HMM we use is a model of character durations. In its simplest HMM implementation, a duration model has one state per character category and all transitions are possible, including self loops. The probability of remaining exactly d time steps in the same state i decreases exponentially with d.

In our experiments we use a more sophisticated model of duration. Each character category C_i is associated to several states: $S_{i,1}, \ldots, S_{i,N_i}$, where N_i is the number of states for category C_i (Figure 8). States associated with the same character have the same observation probability $P(C_i|X_t)$. The transition probabilities are calculated to reflect the desired distribution of durations. The probability of remaining exactly d time steps in the same character model i after entering it is, according to equation 1, proportional to:

$$\rho(d) = (1 - P(S_{i,(d+1)}|S_{i,d})) \prod_{t=1}^{d} P(S_{i,t}|S_{i,t-1}),$$

with the convention $P(S_{i,1}|S_{i,0}) = P(S_{i,1}) = 1$. With a simple recursion formula, one can determine $P(S_{i,t}|S_{i,t-1})$ from values of the distribution $\rho(d)$:

$$P(S_{i,2}|S_{i,1}) = 1 - \rho(1)$$

$$P(S_{i,d+1}|S_{i,d}) = 1 - \rho(d) / \prod_{t=1}^{d} P(S_{i,t}|S_{i,t-1})$$

We truncate the distribution by limiting the model to a finite number of states and add a self loop to the last state. This is equivalent to replacing the tail of the distribution by an exponential.

For $\rho(d)$ we chose a Poisson distribution, as it fitted best the experimentally found duration distributions. The parameter λ of the distribution (both mean and variance) was set by estimating statistically character durations from the training data. We use 3-state models with $\lambda = 2$ for all characters and a 6-state model with $\lambda = 4$ for the *nil* category. Experiments to fine tune the transition probabilities, by training them with the EM algorithm [22], did not result in any significant improvement.

In Figure 5 an example of the output of the TDNN is given with the result of the HMM segmentation with or without dictionary constraints. Figure 6 shows graphically the HMM trellis given by the HMM "unfolded in time" in connection with the *state observation* probabilities given by the outputs of the TDNN.

4.3 Dictionary search

As mentioned in the introduction, it is very difficult to recognize cursive handwriting without language knowledge. In the present study, we implement a grammar which forces the HMM to recognize only words from a given dictionary (Figure 7-c).

To keep the search for the best match computationally tractable, we use a fast-match algorithm (see e.g. [27]). We calculate the Levenshtein-distance [28, 29] (number of edit operations) between the word found without dictionary and all words in the dictionary. We then calculate the scores of the HMM *only* for the group of words that are within a certain range of Levenshtein-distance. As the calculation of the Levenshtein distance is much cheaper than the evaluation of the HMM, we easily obtain a speed-up in the order of 10^3.

Two variants of our fast-match dictionary search were tried.

- In a first variant, we selected all dictionary words that have the smallest edit-distance to the word recognized without dictionary constraint. With this method we selected only 4 words on the average in a 25k word dictionary. This corresponds approximately to a speed-up by a factor 6,000 compared to full search.
- In a second variant we selected the closest and second-closest group of dictionary words. There are, on the average, 57 words selected. This scheme is 14 times slower than the first one.

5 Results

Our system was trained with the 26'931 cursive word database described in Section 1.1 with the algorithms described in Section 3.2 and 4.1. Training took 30 back-propagation cycles through the training set which corresponds to a week on a SUN Sparc-10. We tested the system on 607 cursive words from a disjoint set of writers, with and without dictionary constraints.

Table 2 gives results for cursive words and, for comparison, results for the same words written in uppercase by the same subjects, using the same postprocessor with the uppercase network described

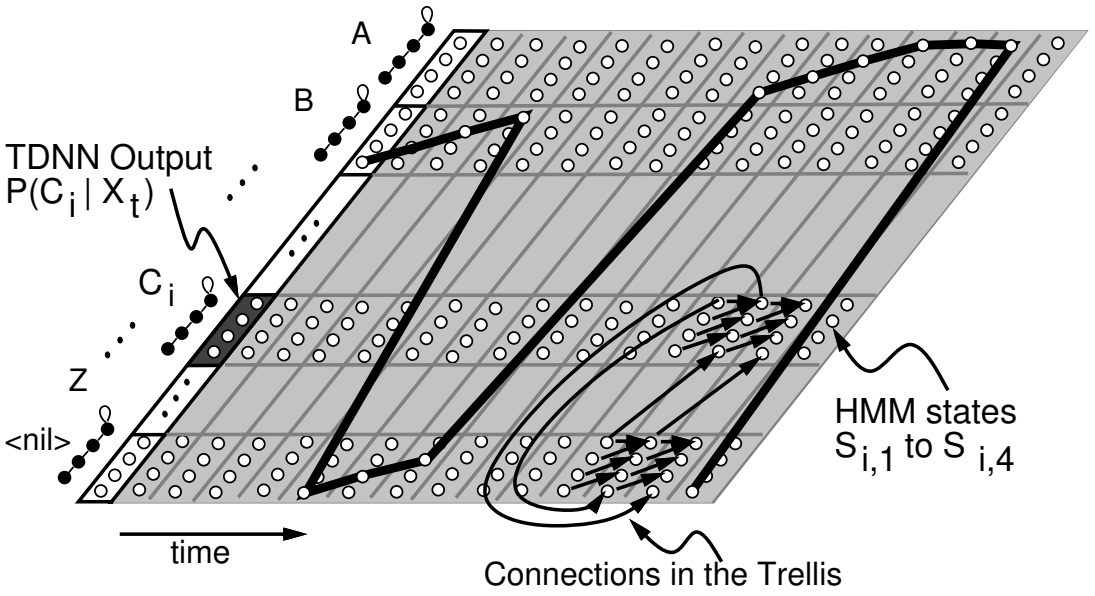

Figure 6: **Hidden Markov Model (HMM).**
Not all neurons and HMM states are drawn and only a few connections between neurons and between HMM states are represented. The output of the TDNN is a sequence of vectors representing character posterior probabilities $P(C_i|X_t)$. In the HMM layer, each character category has an HMM duration model. The connections between models, given by the grammar, are not represented. The HMM trellis has *state transition* probabilities given by the HMM "unfolded in time". The *state observation* probabilities are given by the outputs of the TDNN. Within one duration model, all the states have the same observation probability, as given by the output of the TDNN of the corresponding character category. The best path through the trellis is found by the Viterbi algorithm. The path shown by the thick black line on the figure corresponds the word "BA".

in [11]). As expected reading cursive is much harder than reading uppercase but the use of a dictionary is a significant help in both cases.

The character error-rate is the fraction of insertions, deletions and substitutions, each counting as 1 error, over the total number of characters. Error rates larger than 100% are possible with this scheme.

We compare the results obtained without dictionary, with our two fast-match dictionary search and with a lower bound on full dictionary search. The lower bound is obtained by discounting all errors for which the score for the correct word (which can be calculated using the true label) is better than the score of the best word found with the fast-match method.

The performances of the fast-match technique are very good compared to full search since by doing only a small fraction of the calculations, near optimal error-rates are obtained. Indeed, the maximal improvement in word error rate that can be obtained using the Viterbi search on the entire dictionary is less than one percent. The times to read a character on a SUN Sparc-10 are 0.8 seconds for smallest distance fast-match and 3.2 seconds for the second scheme. Of that time, the preprocessing and the TDNN need about 0.6 seconds per character. We estimated that our system could run 4 times faster if optimized and implemented in C. At present, most of the time is spent in routines written in Lisp.

The performance of our recognizer for various dictionary sizes is shown in Figure 9. The tests were performed with seven dictionaries of sizes between 521 and 25,104 words. Each dictionary consisted of the 521 unique words of the test set plus a random selection of additional words from the UNIX Spell dictionary (25,104 words). The solid lines represent the search scheme using the smallest-distance words, the dashed lines correspond to the errors using the first and second-closest words.

As expected the size of the dictionary is quite influential on the performance. We observe a factor of 2-3 difference in the word error rates between a 1k word and a 25k word dictionary.

Table 3 shows the influence of word length on word error rates. Without dictionary, the longer the word, the more likely it is that at least one character will be wrong. Thus error rates without dictionary

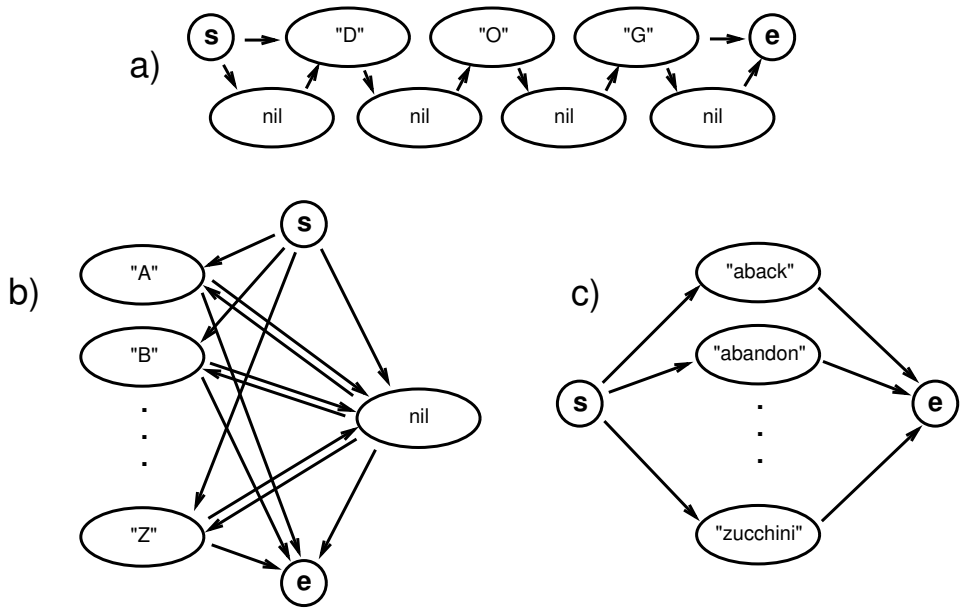

Figure 7: **Three types of grammars used:**
a) Grammar for a single word; the HMMs only determine the segmentation and the score of the given word. Each box corresponds to a character model. **b)** Free grammar for any character sequence; there are no restrictions on the character sequence except to visit the *nil*-class between characters. **c)** Grammar forcing to find a word of a given word list. Each box represents a word model of type a).

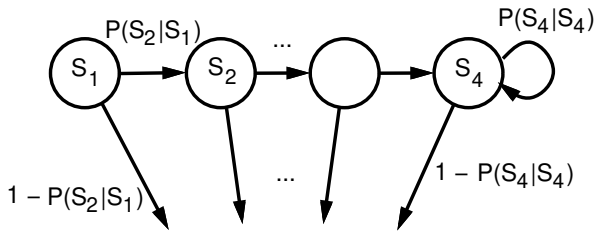

Figure 8: **Character HMM duration model** Example of a 4-state HMM used to model character duration. The category index i has been dropped. By adjusting the transition probabilities one can model different duration distributions $\rho(d)$. We used for $\rho(d)$ a Poisson distribution which was fitted to the empirical distribution obtained with the training data.

are higher for long words. The situation is reversed when using a dictionary. This is because there are fewer long English words than short ones. The postprocessing exploits well this redundancy in word coding provided by the language. In the experiments reported in Table 2, test words were randomly drawn from a dictionary, without respecting natural word frequencies. It is worth noticing that in grammatical English short words occur more frequently than in our experiments.

Finally we investigated the robustness of our system under writing style variability. When looking at the error rates of each writer in the training or test set we found word error rates between 5% for nice handwritings to 40% for sloppy writing. This shows how much the performance still depends on the writing style of the individual.

6 Discussion

Because of the highly ambiguous nature of cursive handwriting a statistical approach which can deal with conflicting data seems to be appropriate for the recognition. We found the combination Time delay Neural Network / Hidden Markov Model particularly suitable for on-line handwriting recognition of both

	Uppercase		*Cursive*	
Method	% char. error	% word error	% char. error	% word error
no dictionary	11.4	48.4	28.2	81.2
smallest dist	1.8	4.1	12.2	23.6
2nd smallest	1.4	3.9	10.9	20.3
lower bound	-	3.9	-	19.6

Table 2: **Error rates on 607 uppercase and 607 cursive test words from 25 new writers.** A 25,000 word dictionary is used for lexical checking. *Smallest dist.* refers to the fast-match dictionary search by exploring only the closest words and *2nd smallest* to closest and 2nd-closest exploration. *Lower bound* refers to a lower bound on the error rate obtainable with a full dictionary search.

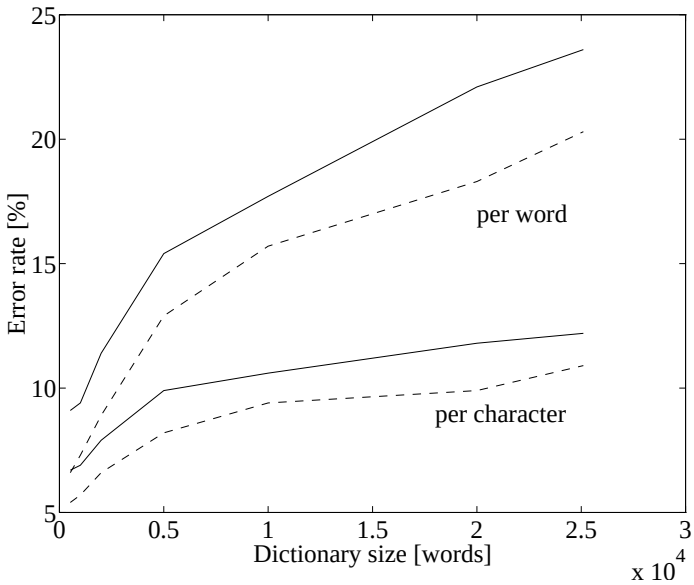

Figure 9: **Performance for different dictionary sizes on the test set.** The solid lines represent the search scheme using the smallest-distance words, the dashed lines correspond to the errors using the second-smallest-distance check.

hand-printed and cursive styles.

Although we trained our system with a large database of cursive words, our results indicate that there were too few writers (59) in the database to provide a truly writer independent system. In particular, most data donors had an American handwriting style. When tried by European writers the system was not recognizing well r's, t's and x's which are written in a different style. We are presently collecting more data to enhance our system.

The strength of the system presented here lies in its ability to deal with strongly distorted handwriting (as e.g. the word *cyclorama* in Figure 2). It is able to deal with words where strokes or even characters are missing thanks to the fact that no early decisions on the segmentation are made. The system is also able to deal with printed character within the cursive (as e.g. in the word *bottle* in Figure 2).

Although our results seem encouraging, it is clear that a real performance evaluation requires a controlled benchmark with a standard data set. We hope that in the near future on-line handwriting databases will be made publicly available and that open benchmarks will be organized. We are actively involved in such projects.

	Error in % word	
Word length	No dict.	dict.
short words (3-6 chars)	80.9	29.6
long words (7-15 chars)	95.4	19.2

Table 3: **Error rates for different word length on the test set.** The performance improvement due to the dictionary is much bigger for long words (7 - 15 character) than for short words (3 - 6 character).

Acknowledgments

We wish to thank all the Neural Network group at Bell Labs Holmdel for supportive discussions and in particular Yann LeCun, Yoshua Bengio, Bernhard Boser and Larry Jackel for many useful suggestions.

References

[1] M. Berthod. On-line analysis of cursive writing. In C. Y. Suen and R De Mori, editors, *Computer Analysis and Perception: Vol. 1 – Visual Signals*, pages 55–81. CRC Press, 1990.

[2] C. C. Tappert, C. Y. Suen, and T. Wakahara. The state of the art in on-line handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(8):787–808, 1990.

[3] E. Lecolinet and O. Baret. Cursive word recognition: Methods and strategies. In *Fundamentals in Handwriting Recognition*, Bonas, July 1993. NATO-ASI workshop.

[4] L. D. Harmon. Automatic reading of cursive script. In G. L. Fisher, editor, *Optical Character Recognition*, page 151, 1962.

[5] P. Mermelstein and M. Eden. Experiments on comuter recognition of connected handwritten words. *Information and Control*, 7:255, 1964.

[6] R. W. Ehrich and K. J. Koehler. Experiments in the contextual recognition of cursive script. *IEEE Transactions on Computers*, 24(2):182–194, 1975.

[7] K. M. Sayre. Machine recognition of handwritten words: A project report. *Pattern Recognition*, 5:213–228, 1973.

[8] L. Schomaker. Using stroke- or character-based self-organizing maps in the recognition of on-line, connected cursive script. *Pattern Recognition*, 26(3):443–450, 1993.

[9] P. Morasso, L. Barberis, S. Pagliano, and D. Vergano. Recognition experiments of cursive dynamic handwriting with self-organizing networks. *Pattern Recognition*, 26(3):451–460, 1993.

[10] R. Nag, K. H. Wong, and F. Fallside. Script recognition using hidden markov models. In *ICASSP*, pages 2071–2074, Tokyo, May 1986. IEEE.

[11] M. Schenkel, H. Weissman, I. Guyon, C. Nohl, and D. Henderson. Recognition-based segmentation of on-line hand-printed words. In S. J. Hanson et al., editor, *Advances in Neural Information Processing Systems 5*, pages 723–730, Denver, 1993. Morgan Kaufmann.

[12] H. Bourlard and N. Morgan. *Connectionist Speech Recognition.* Kluwer Academic, 1993.

[13] O. Matan, C. J. C. Burges, Y. Le Cun, and J. Denker. Multi-digit recognition using a Space Displacement Neural Network. In J. E. Moody et al., editor, *Advances in Neural Information Processing Systems 4*, Denver, 1992. Morgan Kaufmann.

[14] Y. Bengio, Y. Le Cun, and D. Henderson. Globally trained handwritten word recognition using spatioal representation, space displacement neural networks and hidden markov models. In *Advances in Neural Information Processing Systems 6*. Morgan Kauffman (to appear), 1994.

[15] 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.

[16] H. Weissman, M. Schenkel, I. Guyon, C. Nohl, and D. Henderson. Recognition-based segmentation of on-line handprinted words: Input vs. output segmentation. *Pattern Recognition*, 27(3):405–420, 1994.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] J. Keeler, D. E. Rumelhart, and W-K. Leow. Integrated segmentation and recognition of hand-printed numerals. In J. E. Moody et al., editor, *Advances in Neural Information Processing Systems 3*, pages 557–563, Denver, 1991. Morgan Kaufmann.

[22] L. R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77-2, February 1989.

[23] T. Fujisaki, K. Nathan, W. Cho, and H. Beigi. On-line unconstrained handwriting recognition by a probabilistic method. In *Int. Workshop on Frontiers in Handwriting Recognition III*, pages 235–241, Buffalo, May 1993. IAPR.

[24] J. Y. Ha, S. C. Oh, J. H. Kim, and Kwon Y. B. Unconstrained handwritten word recognition with interconnected hidden markov models. In *Int. Workshop on Frontiers in Handwriting Recognition III*, pages 455–460, Buffalo, May 1993. IAPR.

[25] H. Bourlard and C. Wellekens. Links between markov models and multilayer perceptron. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12-12:1167–1178, December 1990.

[26] W. Y. Huang and Lippmann R. P. Hmm speech recognition with neural net discrimintation. In *Advances in Neural Information Processing Systems 2*, pages 194–202, Denver, 1990. Morgan Kaufmann.

[27] V. N. Gupta, M. Lennig, and P. Mermelstein. Fast search strategy in a large vocabulary word recognizer. *J. Acoust. Soc. Amer.*, 84(6):2007–2017, 1988.

[28] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics-Doklady*, 10(8):707–710, 1966.

[29] T. Okuda, E. Tanaka, and K. Tamotsu. A method for the correction of garbled words based on the levenshtein metric. *IEEE Transactions on Computers*, c-25(2):172–177, 1976.