IJDAR (2001) 3: 169–180

International Journal on Document Analysis and Recognition

Online handwriting recognition: the NPen++ recognizer

S. Jaeger[1,2], **S. Manke**[1,2], **J. Reichert**[1,2], **A. Waibel**[1,2]

[1] Interactive Systems Laboratories, University of Karlsruhe, Computer Science Department, 76128 Karlsruhe, Germany; e-mail: stefan.jaegar@ira.uka.de
[2] Interactive Systems Laboratories, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA 15213-3890, USA

Received September 3, 2000 / Revised October 9, 2000

Abstract. This paper presents the online handwriting recognition system NPen++ developed at the University of Karlsruhe and Carnegie Mellon University. The NPen++ recognition engine is based on a multi-state time delay neural network and yields recognition rates from 96% for a 5,000 word dictionary to 93.4% on a 20,000 word dictionary and 91.2% for a 50,000 word dictionary. The proposed tree search and pruning technique reduces the search space considerably without losing too much recognition performance compared to an exhaustive search. This enables the NPen++ recognizer to be run in real-time with large dictionaries. Initial recognition rates for whole sentences are promising and show that the MS-TDNN architecture is suited to recognizing handwritten data ranging from single characters to whole sentences.

Key words: Online handwriting recognition – Neural networks – Pen-based computing – Pattern recognition – Human-computer interaction

1 Introduction

Pen-based interfaces are becoming increasingly popular and will play an important role in human-computer interaction in the near future. A clear indication for this is the advent of personal digital assistants, i.e., small handheld computers accepting pen-based input and combining agenda, address book, and telecommunication facilities. These palm computers are widely-used pen-based systems already.

Using a pen as an input device has several advantages: while a pen is much easier to carry around, which is a very important feature for small portable computers, a pen can also take over many functions of a conventional mouse. For instance, clicking and dragging are two basic operations that a pen can perform just as well. One major advantage of the pen over the mouse is the fact that a pen is a natural writing tool while the mouse is very cumbersome when used as a writing tool.

However, the reliable transformation of handwritten text into a coding that can be directly processed by a computer, e.g., ASCII, is essential for the pen to become a preferred input device. Handwriting needs reliable recognition rates to be accepted as an alternative input modality.

This paper presents the online handwriting recognition system called NPen++ that is based on a multi-state time delay neural network (MSTDNN). NPen++ yields recognition rates from 96% for a 5,000 word dictionary to 93.4% on a 20,000 word dictionary and 91.2% for a 50,000 word dictionary. The recognition times are virtually independent of the size of the dictionary. In this paper, the preprocessing steps, the computation of features, the recognizer with training and testing, and the dictionary-based search are described in detail. Section 2 begins with the description of the preprocessing steps in NPen++ that normalize the handwritten raw data. Section 3 shows different features computed from the normalized raw data. The core of the NPen++ recognition engine; i.e., the MSTDNN, is presented in Sect. 4. Training and recognizing are described in Sect. 5 and results for both word recognition and sentence recognition are presented in Sect. 7. Section 6 describes the search technique used in the NPen++ system for words and sentences. Section 8 contains a summary that concludes this paper.

2 Preprocessing

This section describes the preprocessing steps of NPen++. Before features are derived from the handwritten word, the raw data recorded by the hardware goes through several preprocessing steps. Raw data for the NPen++ system was collected with a sampling rate of 200 points/s using Wacom graphic tablets (the opaque SD-421E Tablet and the HD-648A Tablet with integrated LCD screen), where only the pen-down segments were recorded by the hardware.

Correspondence to: S. Jaeger

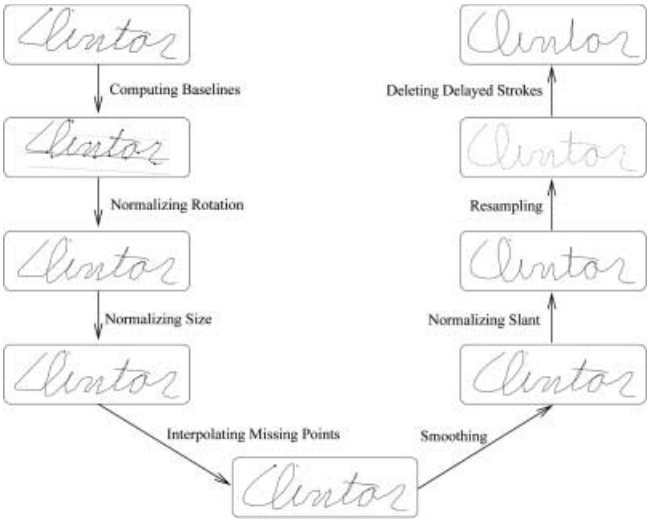

Fig. 1. Preprocessing (overview)

The main objective of the preprocessing steps is to normalize words and remove variations that would otherwise complicate recognition. Typical variations are size, slant, and rotation. These variations can provide important hints in signature verification and writer identification tasks, but they complicate handwriting recognition in general.

While several preprocessing steps in NPen++ are modeled on existing online handwriting recognition systems [4,7,20–22], some steps, such as context maps, are unique in NPen++.

Figure 1 gives an overview of the preprocessing steps of NPen++.

2.1 Computing baselines

Computing baselines is a main technique in handwriting recognition and is implemented in several online as well as offline (OCR) systems (e.g., [4,22]). Baselines are utilized for many reasons, e.g., normalizing sizes, correcting rotations, or deriving features. Many systems, including NPen++, compute the following two lines:

- **Baseline:** The baseline corresponds to the original writing line on which a word or text was written.
- **Corpus line:** The corpus line goes through the top of lower case characters such as "n" or "w".

In the approach described in this paper, however, we compute two additional lines as described in [2]. The first additional line goes through ascenders (tops of characters such as "E" or "l") and the second line goes through descenders (bottoms of characters such as "p" or "g"). Figure 2 shows all four lines for the word "writing." Baselines are often computed using linear regression lines that approximate the local minima (baseline) or the local maxima (corpus line) of the trajectory [4,21,22]. In the NPen++ system, however, all four lines are defined as polynomials with degree 2:

$$Y = f_i(x) = k\,(x - x_0)^2 + s\,(x - x_0) + y_i \quad i = 1, \ldots, 4$$

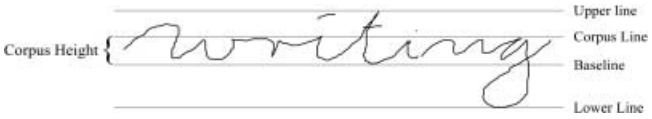

Fig. 2. Baselines

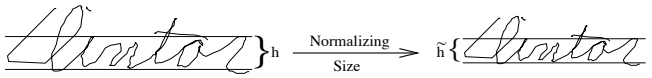

Fig. 3. Normalizing size

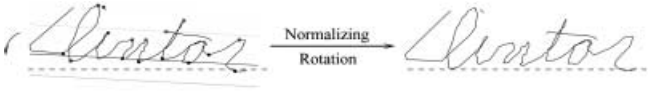

Fig. 4. Normalizing rotation

The parameters k, s, and x_0 are shared among all four curves, whereas each curve has its own vertical translation parameter y_i. Parameter k denotes the curvature and parameter s the rotation of all lines. All parameters are determined by fitting a geometrical model to the pen trajectory. This is accomplished using the Expectation-Maximization algorithm (EM) [2,5]. Before applying the EM step, parameters are initialized with values derived from a simple linear regression to obtain a good seed for the EM procedure. This technique is an advantage over other methods, where baselines and corpus lines are computed separately. In particular, computing corpus lines is facilitated here by coupling them with baselines.

The next two preprocessing steps normalize size and rotation by exploiting information about baselines and corpus lines.

2.2 Normalizing size

The main goal of this preprocessing step is to ensure that the same characters have approximately the same height for every handwritten word. This is accomplished by transforming every word to a given corpus height, where the corpus height is the distance between the baseline and the corpus line. Note that the total height of words may still vary after normalizing size, even for words having the same corpus height. Figure 3 shows the normalization of the word "Clinton."

2.3 Normalizing rotations

Rotations of words, i.e., deviations of the computed baseline from the horizontal writing line, are corrected by means of a Cartesian rotation that is based on the parameter s denoting the rotation of both lines. Figure 4 again shows the normalization of the word "Clinton."

2.4 Interpolating missing points

If the distance between two neighboring points exceeds a certain threshold, we interpolate the trajectory between

Fig. 5. Interpolating missing points using Bezier curves

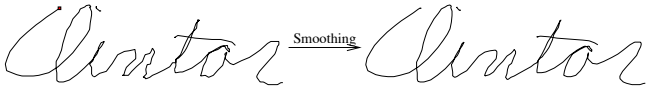

Fig. 6. Smoothing

both points using a Bezier curve [1]. In order to do so, we compute two additional points between both neighboring points and approximate all four points using a Bezier curve [1]. The position of the additional points depends on how the trajectory enters the first and leaves the second neighbor. This preprocessing step may be necessary in situations where the window-manager software is not able to capture all points of the trajectory or cannot catch up with the speed of writing. This preprocessing step has only a minor impact on the recognition rates reported in this paper. However, we have reduced error rates by more than 15% using this preprocessing step for some small subsets of data collected on a specific pressure-sensitive hardware platform. Figure 5 presents some examples before and after interpolating missing points.

2.5 Smoothing

To remove jitter from the handwritten text, we replace every point $(x(t), y(t))$ in the trajectory by the mean value of its neighbors:

$$x'(t) = \frac{x(t-N) + \ldots + x(t-1) + \alpha x(t) + x(t+1) + \ldots + x(t+N)}{2N + \alpha}$$

and

$$y'(t) = \frac{y(t-N) + \ldots + y(t-1) + \alpha y(t) + y(t+1) + \ldots + y(t+N)}{2N + \alpha}.$$

The parameter α is based on the angle subtended by the preceding and succeeding curve segment of $(x(t), y(t))$ and is empirically optimized. This helps to avoid the smoothing of sharp edges, which provide important information when there is a sudden change in direction. The smoothed image of the word "Clinton" is depicted in Fig. 6. In our experiments, smoothing has improved our overall recognition rate by about 0.5%.

2.6 Normalizing slant

To normalize the different slants of words, we shear every word according to its slant. The slant is determined by a histogram over all angles subtended by the lines connecting two successive points of the trajectory and the horizontal line. The computed angles are weighted with

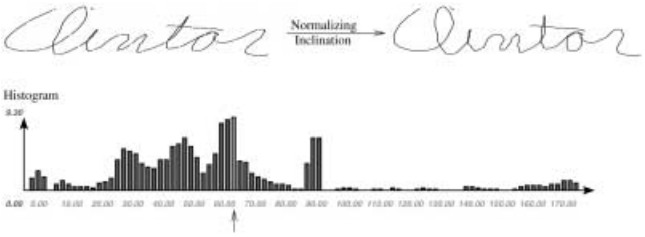

Fig. 7. Normalizing slant

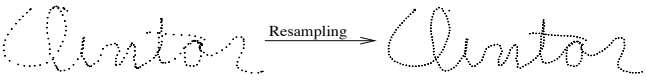

Fig. 8. Resampling

the distance of every pair of successive points. A simple search for the maximum entry in the histogram provides us with the slant of the word [14, 21]. Figure 7 shows the normalization of slant for the word "Clinton." Both the histogram containing all computed angles and the normalized word are shown in Fig. 7. Normalizing the slant has improved our overall recognition rate by just under 1%. However, we have observed major improvements for left-handed writers.

2.7 Computing equidistant points (Resampling)

This processing step is implemented in almost every online handwriting recognition system. In general, the points captured during writing are equidistant in time but not in space. Hence, the number of captured points varies depending on the velocity of writing and the hardware used. To normalize the number of points, the sequence of captured points is replaced with a sequence of points having the same spatial distance. In NPen++, this distance is some fraction of the corpus height, which has been empirically optimized. The optimal value found in our experiments is $\frac{h}{13}$, where h denotes the corpus height. Figure 8 illustrates the resampling of points. We have achieved major improvements of more than 5% applying this preprocessing step.

2.8 Removing delayed strokes

Delayed strokes, e.g., the crossing of a "t" or the dot of an "i", are a well-known problem in online handwriting recognition. These strokes introduce additional temporal variation and complicate online recognition because the writing order of delayed strokes is not fixed and varies between different writers. This stands in contrast to offline recognition since delayed strokes do not occur in static images. It is one of the reasons why there have been approaches in the recent past trying to exploit offline data in order to improve online recognition rates [11, 14]. Many online recognizers apply a simple technique to cope with delayed strokes: they use heuristics for detecting such strokes and remove them before proceeding with feature computation and recognition. NPen++ handles

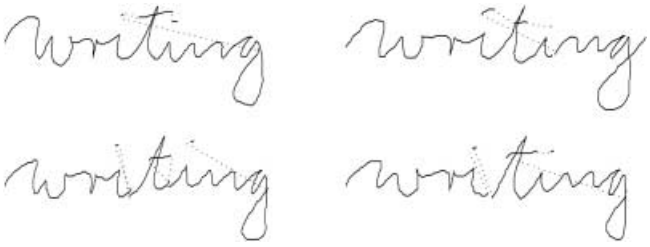

Fig. 9. Delayed strokes

delayed strokes the same way. A delayed stroke is usually a short sequence written in the upper region of the writing pad, above already written parts of a word, and accompanied by a pen movement to the left. NPen++ uses some simple threshold values for characterizing these features of delayed strokes. Figure 9 illustrates some examples of delayed strokes and the corresponding trajectories of the pen. Removing delayed strokes has improved our overall recognition rate about 0.5%.

3 Computing features

There is no doubt that computing features is a very important processing step in every online recognition system. However, neither a standard method for computing features nor a widely accepted feature set currently exists. In the NPen++ system, feature computation takes the normalized sequence of captured coordinates $(x(t), y(t))$ as input and computes a sequence of features along this trajectory, which is then directly put into the recognizer. This section presents the features of NPen++, some of which can already be found in the literature and others which have been implemented in NPen++ for the first time.

3.1 Vertical position

The vertical position of a point $(x(t), y(t))$ is the vertical distance between $y(t)$ and $b(x(t))$, where $b(x(t))$ is the y-value of the baseline at time t. The vertical distance is positive if $(x(t), y(t))$ is above the baseline and negative if it is below. All distances are normalized with the corpus height of the word [14].

3.2 Writing direction

The local writing direction at a point $(x(t), y(t))$ is described using the cosine and sine [8]:

$$\cos\ \alpha(t) = \frac{\Delta x(t)}{\Delta s(t)},$$

$$\sin\ \alpha(t) = \frac{\Delta y(t)}{\Delta s(t)},$$

where $\Delta s(t), \Delta x(t)$, and $\Delta y(t)$ are defined as follows:

$$\Delta s(t) = \sqrt{\Delta x^2(t) + \Delta y^2(t)},$$

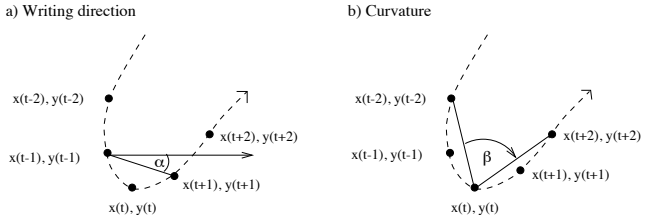

Fig. 10. Writing direction and curvature

$$\Delta x(t) = x(t-1) - x(t+1),$$

$$\Delta y(t) = y(t-1) - y(t+1).$$

The angles involved in this computation are illustrated in Fig. 10.

3.3 Curvature

The curvature at a point $(x(t), y(t))$ is represented by the cosine and sine of the angle defined by the following sequence of points [8]:

$$(x(t-2), y(t-2)), \quad (x(t), y(t)), \quad (x(t+2), y(t+2)).$$

This is again illustrated in Fig. 10. Strictly speaking, this signal does not represent curvature but the angular difference signal. Curvature would be $\frac{1}{r}$, of a circle touching and partially fitting the curve, with radius r. Cosine and sine can be computed using the values from the previous Sect. 3.2:

$$\begin{aligned}\cos\beta(t) &= \cos\alpha(t-1) * \cos\alpha(t+1) \\ &\quad + \sin\alpha(t-1) * \sin\alpha(t+1), \\ \sin\beta(t) &= \cos\alpha(t-1) * \sin\alpha(t+1) \\ &\quad - \sin\alpha(t-1) * \cos\alpha(t+1).\end{aligned}$$

3.4 Pen-up/pen-down

The pen-up/pen-down feature is a binary feature indicating whether the pen has contact with the writing pad at time t or not. Invisible parts of the trajectory, where the pen has no contact with the pad, are linearly interpolated in NPen++, i.e., a pen-up is connected with the next pen-down by a straight line. Note that the pen-up/pen-down information is dependent on the position-sensing device. Electromagnetic systems are able to return approximate planar positions while the pen is in the air, whereas pressure-sensitive technologies cannot. Since the NPen++ system does not exploit information about pen-up segments, this distinction has no effect on its features.

3.5 "Hat"-feature

This is a binary feature that indicates whether the current position is below a delayed stroke, e.g., below a t-stroke [21]. It is the only information about delayed strokes for the NPen++ system. Note that NPen++ detects and deletes delayed strokes using a simple heuristic (see Sect. 2).

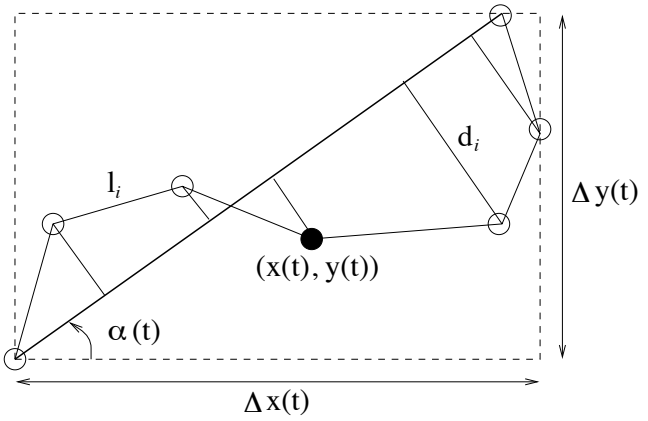

Fig. 11. Aspect

3.6 Aspect

The aspect of the trajectory in the vicinity of a point $(x(t), y(t))$ is described by a single value $A(t)$ [21]:

$$A(t) = \frac{\Delta y(t) - \Delta x(t)}{\Delta y(t) + \Delta x(t)}$$

It characterizes the height-to-width ratio of the bounding box containing the preceding and succeeding points of $(x(t), y(t))$. Figure 11 illustrates the computation of the aspect. The vicinity of a point $(x(t), y(t))$ shown in Fig. 11 is also used to define the following three features: curliness, linearity, and slope, which describe the shape of the trajectory in the vicinity of $(x(t), y(t))$.

3.7 Curliness

Curliness $C(t)$ is a feature that describes the deviation from a straight line in the vicinity of $(x(t), y(t))$ (Fig. 11). It is based on the ratio of the length of the trajectory and the maximum side of the bounding box:

$$C(t) = \frac{L(t)}{\max(\Delta x, \Delta y)} - 2,$$

where $L(t)$ denotes the length of the trajectory in the vicinity of $(x(t), y(t))$, i.e., the sum of lengths of all line segments. $\Delta x$and $\Delta y$are the width and height of the bounding box containing all points in the vicinity of $(x(t), y(t))$ (see Fig. 11). According to this definition, the values of curliness are in the range $[-1; N-3]$. However, values greater than 1 are rare in practice.

3.8 Linearity

The average square distance between every point in the vicinity of $(x(t), y(t))$ and the straight line joining the first and last point in this vicinity is called linearity, which is defined as follows:

$$LN(t) = \frac{1}{N} * \sum_i d_i^2.$$

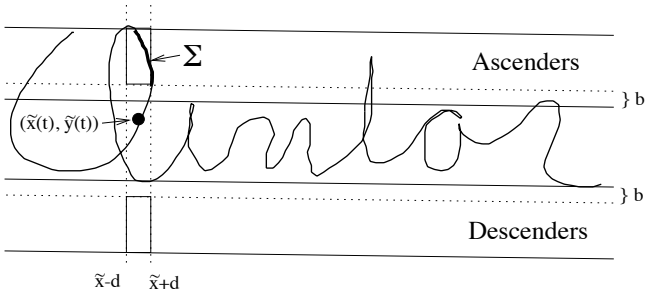

Fig. 12. Ascenders and descenders

3.9 Slope

The slope of the straight line joining the first and last point in the vicinity of $(x(t), y(t))$ is described by the cosine of its angle α (see Fig. 11).

The next two features are global features that consider a wider temporal context, i.e., not only the vicinity of a point $(x(t), y(t))$. These features can be seen as an approach to incorporate additional information about the offline image into the feature vector.

3.10 Ascenders/descenders

These two features count the number of points above the corpus line (ascenders) and the number of points below the baseline (descenders) in the word image at a given time instance t. Only points falling into a local region determined by the x-coordinate of the current point $(x(t), y(t))$, i.e., only points in the word image that have an x-coordinate x satisfying $x(t) - d < x < x(t) + d$, are considered to be part of ascenders or descenders at time t. Additionally, points must have a minimum distance to both lines to be considered as part of ascenders or descenders. This ensures that inaccurately computed baselines have only a minor impact on the computation of these features. Both features are computed by simply counting the number of these points, where every point is weighted with its distance to the corpus line (ascenders) or baseline (descenders). Figure 12 illustrates the computation of both features. The main idea of these features is to help identifying specific characters, like "t" or "g".

3.11 Context maps

A context map is an offline, gray-scale image $B = b(i, j)$ of the vicinity of a point $(x(t), y(t))$, where $b(i, j)$ is the number of points of the trajectory falling into pixel (i, j). In particular, a context map is a low-resolution image with the pixel in the center containing $(x(t), y(t))$. The size of the context map depends on the corpus height of the word. In our experiments, we have transformed the computed map into a 3×3 map before adding it to the feature vector [15]. Figure 13 shows the computation of context maps on a gray-scale representation computed for the word "Clinton" and the transformation to 3×3

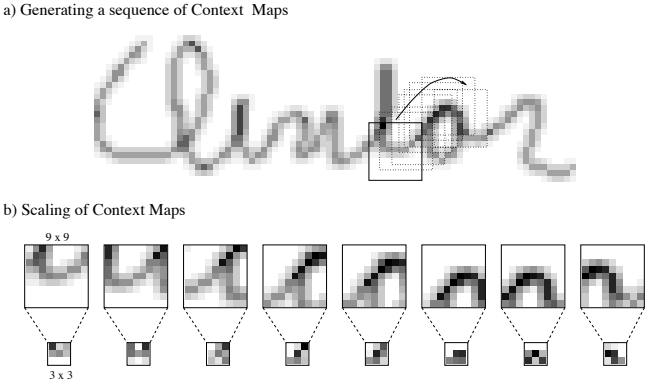

Fig. 13. Context maps

maps. Every pixel of a context map is a feature and thus added to the set of features described above. The main idea of context maps is to add features that consider a wider context. Context maps allow us to capture information about parts of the trajectory that are in the spatial vicinity of a point $(x(t), y(t))$ but have a long time distance to this point. Context maps are an example of combining offline and online information in handwriting recognition. This is an interesting research topic, which has been investigated by several researchers in the recent past (see [3,6,11,13–15] for more details).

As to the relevance of local and global features, we have made the following experiments for a 20,000 word dictionary: We have achieved recognition rates about 90% when taking only the following small subset of local features for training: vertical position, writing direction, curvature, and pen-up/pen-down information. The overall recognition rate of a system trained only with local features is 2% higher compared to a system trained with global features only. The performance of global features without any local features involved in the training, however, is close to the best overall performance. Nevertheless, optimal performance has been achieved by combining both local and global features into a single feature vector.

4 Multi-state time delay neural networks (MS-TDNN)

The core recognition engine of NPen++ is a MS-TDNN. A MS-TDNN is a connectionist recognizer that integrates recognition and segmentation into a single network architecture. This architecture was originally proposed for continuous-speech recognition tasks [10,16,23]. We adopted this technique, including the training procedure described in the next section, and have applied it to the handwriting recognition problem. The MS-TDNN is an extension of the time delay neural network (TDNN) [23], which has been applied successfully to online *single* character recognition tasks. The main feature of a TDNN is its time-shift invariant architecture, i.e., a TDNN is able to recognize a pattern independently of its position in time. The MS-TDNN combines the high accuracy character recognition capabilities of a TDNN with

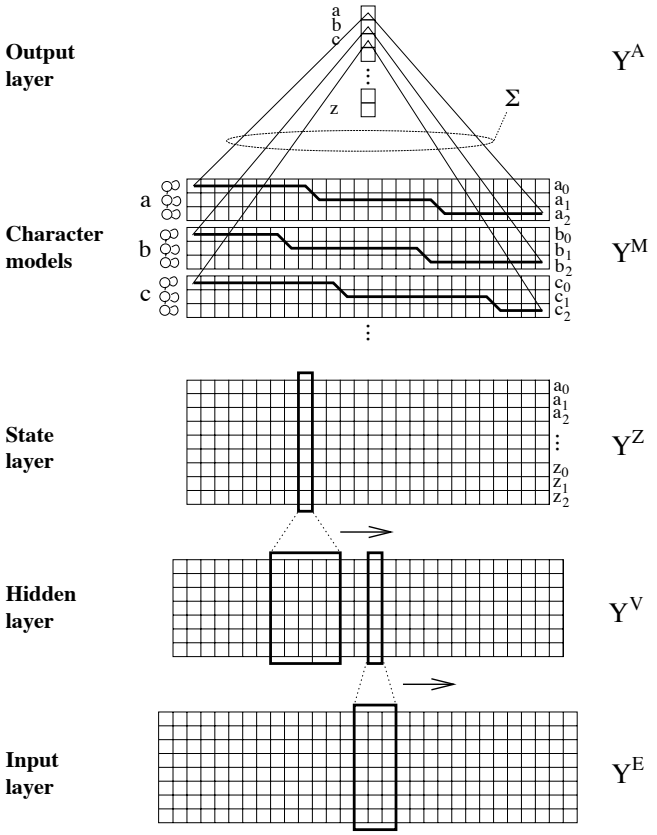

Fig. 14. The architecture of a multi-state time delay neural network

a nonlinear time alignment procedure to find an optimal alignment between strokes and characters in handwritten words [9,10,14]. Thus, the MS-TDNN is capable of recognizing words by integrating a dynamic time-warping algorithm (DTW) into the TDNN architecture. In MS-TDNNs, words are represented as a sequence of characters with each character being modeled by one or more states. In the experiments reported in this paper, each character is modeled with three states representing the first, middle, and last part of the character. Hence, the MS-TDNN can be regarded as a hybrid recognizer that combines features of both neural networks and hidden Markov models (HMMs). Figure 14 shows the basic architecture of the MS-TDNN. The first three layers, which are respectively called input layer, hidden layer, and state layer, constitute a TDNN. The input layer consists of the set of feature vectors computed for every time instance t. A sliding window moving from left to right integrates several neighboring input vectors into activation vectors in the hidden layer, which in turn are integrated by a sliding window into activation vectors in the state layer. The width of the sliding windows is a parameter of the MS-TDNN architecture. Every element ("neuron") in the state layer represents a state of a character in the alphabet (see Fig. 14). In single character recognition, the score for a character is computed by finding an optimal alignment path through its states and summing the activations along this path. In word recognition, the score of a word is computed by finding an optimal align-

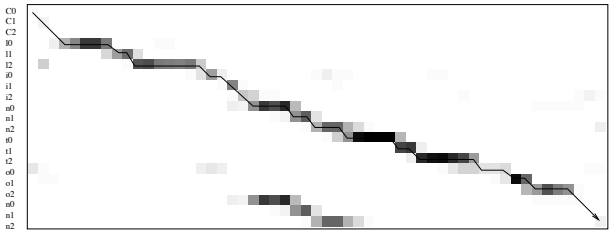

Fig. 15. Optimal alignment path through the word "Clinton" (1)

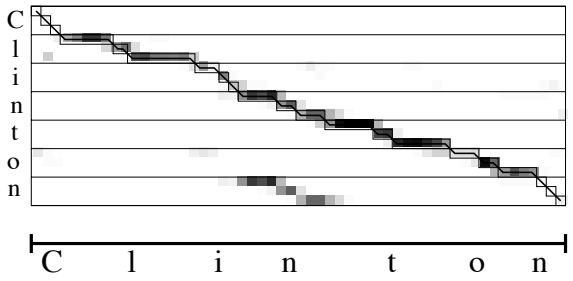

Fig. 16. Optimal alignment path through the word "Clinton" (2)

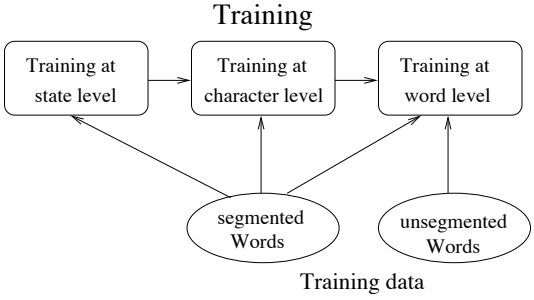

Fig. 17. Training of a multi-state time delay neural network

ment path through the states of characters composing the word. The final score is again derived by summing all activations along this path. Figure 15 and 16 illustrate the computation of the optimal alignment path through the word "Clinton", where dark squares indicate highly activated states.

5 Training

The MS-TDNN is trained in three steps with back-propagation. Figure 17 shows a schematic diagram of the training procedure. The first and second training step operate in a forced alignment mode, during which the MS-TDNN is trained with hand-segmented training data, i.e., the character boundaries are known for these words. In the first step (training at state level), it is assumed that the Viterbi path remains for the same duration in every state of a character belonging to a word of the training set. The states along this Viterbi path constitute the training data for the back-propagation procedure starting at the state layer of the MS-TDNN. Figure 18 shows this first step of training. After some iterations, the assumption that the Viterbi path remains for the same duration in every state is abandoned in favor of computing the actual Viterbi path through a character model, which marks the beginning of Step 2 (training at character level). This is shown in Fig. 19. Then, after some iterations, the third step commences by replacing the forced alignment in Step 1 and Step 2 with the free alignment provided by the Viterbi algorithm (training at word level). This has the advantage that training can now be performed on unsegmented data. Thus, only a small part of the training data needs to be labeled manually with character boundaries, as required by the first and second step. When the network has successfully learned character boundaries on the small segmented training set, the forced alignment is replaced by a free alignment and training can be performed on large databases containing unsegmented training data.

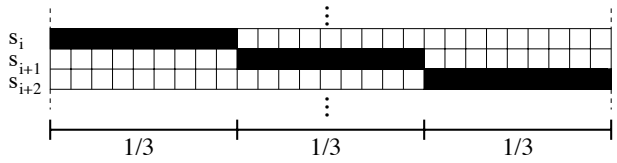

Fig. 18. Forced alignment with equal durations for each state

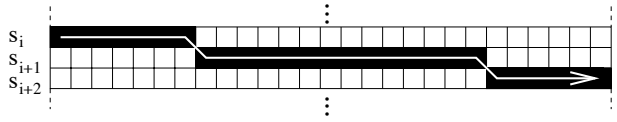

Fig. 19. Forced alignment together with the Viterbi algorithm

This mechanism also works for the next level in the hierarchy: whole sentences. We have trained unsegmented sentences with a MS-TDNN that was already pre-trained with single words. The Viterbi algorithm finds the best segmentation of all unsegmented sentences based on the information the neural net already learned on the word level. Our first results with this technique on the sentence level are presented in Sect.7.

In our experiments, we used the cross entropy for propagating the error E_{CE} back in the MS-TDNN [14]:

$$E_{CE} = -\sum_j [d_j \log(y_j) + (1 - d_j) \log(1 - y_j)],$$

where y_j is the output of unit j and d_j is the teaching input for unit j.

6 Search technique

The NPen++ online handwriting recognition system is based on a dictionary determining the set of recognizable words. In general, the size of the dictionary influences both the recognition performance and response time of a dictionary-based recognizer, and thus has an effect on the degree of user acceptance. NPen++ supplements the MS-TDNN approach with a tree-based search engine [17]. It combines a tree representation of the dictionary

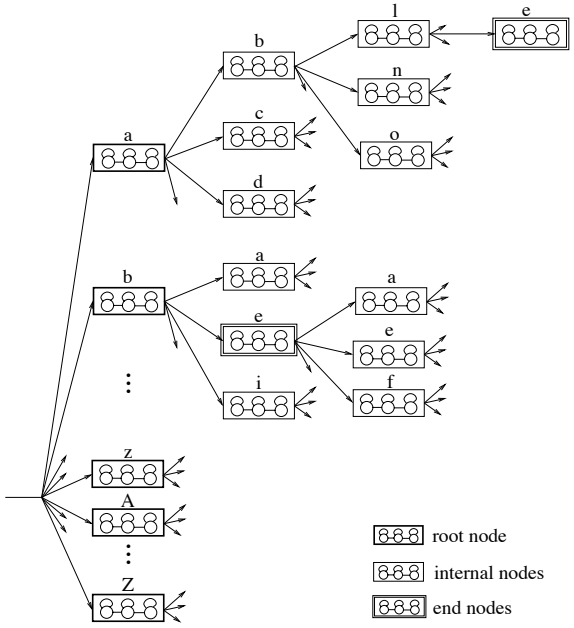

Fig. 20. Tree representation of a dictionary

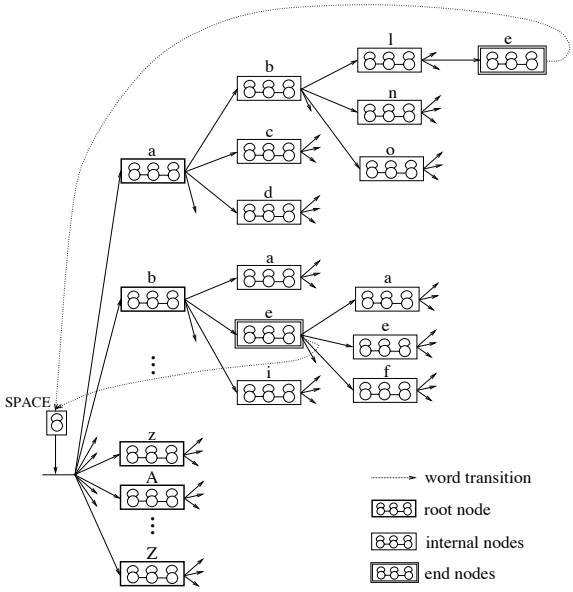

Fig. 21. Tree representation for recognizing sentences

with efficient pruning techniques to reduce the search space without losing much recognition performance compared to a flat exhaustive search through all words in the dictionary. A search tree is built for every character representing all words starting with this specific character. The nodes in every tree are HMMs representing individual characters. Every tree contains distinguished end nodes. A path from the root of a tree to a distinguished end node is a sequence of HMMs describing an entry of the dictionary. Figure 20 shows a dictionary represented as a tree. In order to achieve real-time performance for very large dictionaries, NPen++ does not attempt to apply the exact Viterbi algorithm. Instead, NPen++ introduces the concept of active and inactive HMMs and defines a set of pruning rules which specify when to turn on an inactive HMM and when to turn off an active one. Every HMM-node in the tree can be marked as being either active or inactive. When the search is initialized only the roots of the trees are turned on whereas all other nodes are set to be inactive. There are two lists whose elements are pointing to active nodes: the first list points to the nodes active in the current frame and the second list is used to gather pointers to those nodes which are expected to be active in the next frame. Based on these two lists, the search algorithm goes for each frame, i.e., feature vector, through the following three steps:

- **Evaluation:** For every active hidden Markov model a Viterbi step is computed to find the accumulated scores s_{i_j} for the next frame, where s_{i_j} is the score at state j in node i. The best state scores $\hat{s}_i$ within every node and the best score $\hat{s} = \max \hat{s}_i$ over all evaluated models are computed.
- **Pruning:** Turn off all currently active nodes in the search tree where the following pruning criterion is fulfilled:

$$\hat{s}_i < \hat{s} - beam$$

That means that all nodes whose best accumulated score is below a certain threshold, called beam, will become inactive in the next frame.
- **Expansion:** For every active node in the current frame, test whether a transition from the last state of the model i to the first state of any child HMM j leads to a higher accumulated score s_{j_0} in the first state of that model. If that holds and the new score is above the pruning threshold, the HMM j is marked to be active in the next frame. Go to step 1.

The tree search with pruning is about 15 times faster than a flat search and allows us to run the recognizer in real-time with large dictionary sizes. Moreover, the run-time is virtually independent of the dictionary size. In practical applications, we can adjust the beam size to find a good compromise between recognition accuracy and speed, as will be shown in the next section.

Figure 21 shows how we can easily extend the tree structure in Fig. 20 in order to recognize whole sentences. We insert an additional node that represents the white space between two words within a sentence. Every end node of the tree is connected with this node, which in turn is connected with every root node. Thus, the traversal of this node marks the beginning of a new word. Realizing this feedback by connecting end nodes directly with root nodes did not perform well in our experiments, so we decided to represent white spaces explicitly using this node. Recognizing sentences requires only minor modifications of the search algorithm and the training of the MS-TDNN. In particular, a new state representing the new node in the search tree is added to the state layer of the MS-TDNN, i.e., the representation of spaces consists of a single state in the state layer. In addition, we have added an additional feature to the feature set described in Sect. 3 that helps training this state. We compute this feature using a simple heuristic that detects white spaces between words: a space between words is said to be accompanied by a pen-up and moving the pen to the right.

For a sequence of pen-downs, the new feature is always 0. However, for the first pen-down encountered after a pen-up, it is higher the more the pen has moved to the right.

The next section also presents our first results for recognizing sentences.

7 Evaluation

This section describes the practical experiments we carried out and presents the recognition rates and recognition times of our system [14]. Section 7.1 describes the chosen design parameters of the MS-TDNN and the data we used for testing. Section 7.4 provides the recognition rates achieved.

7.1 Network architecture

The number of input units ("input neurons") is identical to the number of features described in Sect. 3. Hence, the input layer of the NPen++ recognizer contains 22 input units, one for each feature. The number of units in the state layer is determined by the number of characters in the alphabet and the number of states in the statistical model of each character. Since we want to recognize the Latin characters (small and capital ones) and model each character using three states, we have $26 * 2 * 3 = 156$ states in the state layer. The number of hidden units was empirically set to 120. The widths of the sliding windows in the input layer and the hidden layer are set to 7. While the time shift of the window in the hidden layer is 1, the window in the input layer is moved two frames to the right at every cycle. This ensures that the hidden layer is about half as long as the input layer, which reduces computational costs.

7.2 Dictionaries

In our experiments, dictionaries were compiled by extracting the most likely words from the Wall Street Journal Continuous Speech Recognition Corpus of the Linguistic Data Consortium. For instance, a $20,000$word dictionary would contain the most likely $20,000$words from the Wall Street Corpus and would contain any smaller dictionary derived from this corpus in that way.

7.3 Datasets

We used three different databases for training and testing: the CMU database collected at Carnegie Mellon University, the UKA database compiled at the University of Karlsruhe, and the MIT database from the Massachusetts Institute of Technology [14]. While the CMU and the UKA databases contain printed and cursive handwriting data, the MIT database only contains printed data, where mixed styles are considered cursive. In addition, the CMU database contains whole sentences.

Table 1. Data sets

Data set	Printed		Cursive	
	# Writer	#Words	# Writer	#Words
UKA - training	18	2313	70	3230
UKA - test	5	314	21	1085
CMU - training	39	2020	126	9308
CMU - test	15	347	35	900
MIT - training	119	6307	0	0
MIT - test	40	2120	0	0

Table 2. Sentences of the CMU database

	# Writer	# Sentences	# Words
CMU - training	163	1757	13221
CMU - test	50	161	2322

The major part of these data were written by students at their respective institutes.

The UKA and CMU databases were collected using a Wacom graphic tablet(SD-421E) with a sampling rate of 200 points/s. While part of this data was written directly on the tablet, a substantial part was written with an ink pen on paper lying on the tablet. Collecting the former data requires writers to look at the monitor while writing on the tablet to see the graphically emulated inking. Thus, the latter data can be considered more natural. Instructions to the writers were reduced to a minimum to guarantee the most natural way of writing. This also ensures that the number of collected printed and cursive styles reflects their true proportion in practice. The MIT databases was compiled using a Wacom tablet(HD-648A) with an integrated LCD screen and the same sampling rate as in the UKA and CMU databases. Hence, this data can also be considered very natural.

In brief, the collected data covers a wide range of national peculiarities collected by several position-sensing devices. Almost 500 writers wrote just under 30,000 words. Table 1 gives an overview over all three databases, the data they contain, and the division into training sets and test sets. Table 2 shows the number of sentences contained in the CMU database and their division into training and test set. A detailed description of these data sets is given in [14].

Since we need some hand-segmented data to start training (see Sect. 5), we have explicitly segmented $4,000$ cursive words from the UKA database and $2,000$-cursive words from the CMU database by visual inspection.

7.4 Recognition results

Figure 22 shows the recognition rates of NPen++ on different data sets using several dictionaries. NPen++ achieves recognition rates of 96% for a 5,000 word dictionary when it is trained with printed as well as cursive data from all three databases. The error rate for this completely trained system approximately doubles when the ten-times larger $50,000$word dictionary is used in-

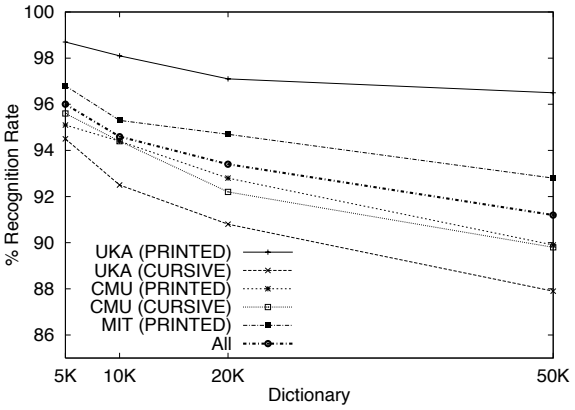

Fig. 22. Recognition rates achieved by NPen++

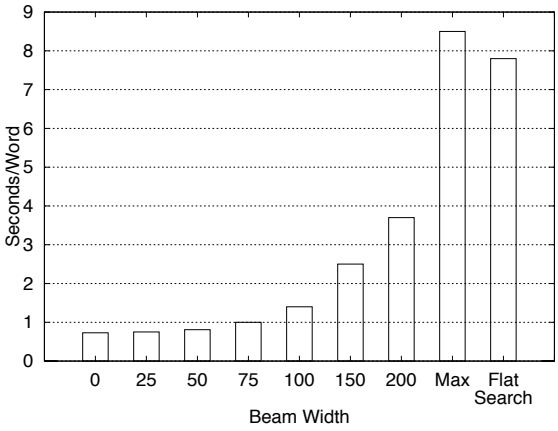

Fig. 23. Recognition times

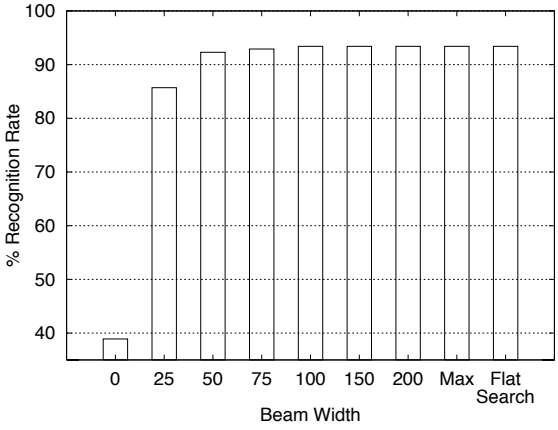

Fig. 24. Recognition rates

stead. The recognition rate is 91.2% for a dictionary of this size.

Figure 22 shows the recognition rates when NPen++ is trained with training sets from all databases and tested separately for each database and each writing style using the test subsets given in Table 1. As one would expect, the recognition rates for printed data are higher than the recognition rates for cursive data.

In our experiments, we have observed that the error rate for the completely trained system increases by more than 1% on the 20,000word dictionary when context maps are excluded from the feature set, as compared to the rate shown in Fig. 22.

In order to evaluate the performance of NPen++ for sentences, we have computed the word accuracy for every test sentence. Word accuracy (WA) is based on the edit distance E, i.e., the smallest number of insertions, deletions, and replacements necessary to transform a word sequence into another:

$$WA = \frac{N - E}{N},$$

where N is the number of words in the test sentence. The average word accuracy measured on the 20,000 word dictionary is more than 81% for recognizing sentences without any language model. In our first experiments with language models, which show great promise, we have been able to increase this word accuracy to more than 86% using delayed bigrams as they are used for continuous speech recognition [18]. Note that NPen++ is among the first online handwriting recognition systems that allow recognition of whole sentences with a neural network architecture and are not exclusively based on hidden Markov models [19].

Figures 23 and 24 illustrate the tradeoff between recognition time and recognition accuracy measured on the 20,000word dictionary for the completely trained word recognizer. Figure 23 shows the recognition times depending on the width of the beam b (see Sect. 6). The rightmost column shows the recognition time for the flat search with no pruning. The column on its left-hand side shows the recognition time for the tree search with an indefinite beam width, i.e., no pruning. Since tree search requires some bookkeeping during its computation, the recognition time is slightly higher than the time for the flat search. With shrinking beam size, however, recognition times rapidly decrease. All recognition times were measured on a PC containing a 200MHz Pentium processor and 64MB SDRAM.

Figure 24 shows the recognition rates depending on the width of the beam. It can be seen that the recognition rates remain almost constant for beam widths equal to or higher than 50. Hence, we can considerably reduce recognition time without losing recognition performance by simply reducing the beam width. A beam width of $b = 75$ or $b = 50$ would be an appropriate choice for interactive applications of NPen++.

8 Summary and conclusion

In this paper, we have presented the online handwriting recognition system NPen++ based on a MS-TDNN, a hybrid architecture combining features of neural networks and hidden Markov models. Two main features of a MS-TDNN are its time-shift invariant architecture and the nonlinear time alignment procedure.

Preprocessing in NPen++ is guided by the application of well-known techniques in handwriting recognition. In particular, the following preprocessing steps are accomplished in NPen++: computing baselines, normalizing size, correcting rotations, interpolating missing

points, smoothing, normalizing slant, computing equidistant points (resampling), and removing delayed strokes.

The set of features computed in NPen++ integrates local as well as global information about the trajectory of the pen. While local features describe the shape of the trajectory at a certain time instance, global features deal with spatial information covering a wider time context. NPen++ considers both types of information important. The following local and global features are computed in NPen++: vertical position, writing direction, curvature, pen-up/pen-down, "hat"-feature, aspect, curliness, linearity, slope, ascenders/descenders, and context maps.

Context maps are a new approach combining global offline features with local online information. They provide additional spatial information about parts of the trajectory that can have a long time distance to each other. Context maps increase overall recognition rates about 1%. However, one disadvantage of context maps is that they increase the size of feature vectors by more than 50%, which in turn increases the complexity of the neural network. Nevertheless, we think that context maps are an important approach for incorporating online as well as offline recognition, which goes beyond combining two separate online and offline recognizers.

NPen++ uses an efficient tree search and pruning technique to ensure real-time performance for very large dictionary sizes. The recognition rates reported in this paper range from 96% for a 5,000 word dictionary to 91.2% for a 50,000 word dictionary. Hence, it is justified to say that NPen++ is a successful example of transfering the MS-TDNN from the speech-recognition domain to the handwriting-recognition task. Moreover, the MS-TDNN is a powerful architecture that allows training and recognition of the whole hierarchy of handwritten data ranging from single characters to whole sentences.

We suppose that transfering the main techniques of NPen++ to other languages, such as Russian, Greek, or Arabic, requires only slight modifications within the NPen++ system. For Asian languages, however, some major changes will be necessary due to the different structure of these languages. For instance, the baselines computed by NPen++ will not work for languages such as Chinese or Japanese. Since an Asian character resembles a Latin word in terms of complexity, the optimal number of states representing a character will presumably be different from the number of states in western writings. In practical experiments, we have achieved the optimal performance of the completely trained NPen++ system when we used three states. However, we have observed some slight improvements with four or even five states for some smaller subsets of our three databases.

A possible improvement that we have not utilized in NPen++ are context-dependent states, which are already used in speech recognition [12]. Representing different writing styles, such as printed and cursive styles, using separate models could also be a potential improvement of NPen++, which deserves future research.

In principle, the basic architecture of NPen++ allows run-on recognition, i.e., recognizing parts of the trajectory during writing. However, run-on recognition may require modifications within some preprocessing steps. In particular, baselines computed during the beginning of the trajectory will be unreliable due to the insufficient number of local maxima or minima.

The NPen++ system presented can be easily extended to enable recognition of complete sentences. In order to so, we have embedded the tree search in a feedback loop joining the output of the tree structure with the root nodes via an additional state representing spaces between words. Our first experiments with sentence recognition have been very encouraging. Nevertheless, these experiments are still in an early stage and there is much room for improvement. Most importantly, the investigation of more elaborate language models, such as trigrams, should be the next major step.

References

1. S. Abramowski, H. Müller.: Geometrisches Modellieren (in German), vol. 75 of Reihe Informatik. BI, Mannheim Wien Zürich, 1991
2. Y. Bengio, Y.L. Cun.: Word Normalization for On-Line Handwritten Word Recognition. In: Proc. Int. Conf. on Pattern Recognition, pp. 409–413, 1994
3. G. Boccignone, A. Chianese, L.P. Cordella, A. Marcelli.: Recovering Dynamic Information from Static Handwriting. Pattern Recognition, 26(3): 409–418, 1993
4. T. Caesar, J.M. Gloger, E. Mandler.: Preprocessing and Feature Extraction for a Handwriting Recognition System. In: 2nd IAPR Int. Conf. on Document Analysis and Recognition, pp. 408–411, Tsukuba, Japan, 1993
5. A.P. Dempster, N.M. Laird, D.B. Rubin.: Maximum Likelihood from Incomplete Data via the EM Algorithm. J. R. Stat. Soc., 39: 1–38, 1977
6. D.S. Doermann, A. Rosenfeld.: Recovery of Temporal Information from Static Images of Handwriting. Int. J. Comput. Vision, 15: 143–164, 1995
7. J.G.A. Dolfing, R. Haeb-Umbach.: Signal Representations for Hidden Markov Model Based On-Line Handwriting Recognition. In: Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, Munich, 1997
8. I. Guyon, P. Albrecht, Y. Le Cun, J. Denker, W. Hubbard.: Design of a Neural Network Character Recognizer for a Touch Terminal. Pattern Recognition, 24(2): 105–119, 1991
9. H. Hild.: Buchstabiererkennung mit neuronalen Netzen in Auskunftssystemen (in German). PhD thesis, University of Karlsruhe, 1997. Shaker, Germany
10. H. Hild, A. Waibel.: Speaker-Independent Connected Letter Recognition with a Multi-State Time Delay Neural Network. In: 3rd European Conf. on Speech, Communication and Technology (EUROSPEECH), vol. 2, pp. 1481–1484, Berlin, 1993
11. S. Jaeger.: Recovering Dynamic Information from Static, Handwritten Word Images. PhD thesis, University of Freiburg, 1998. Foelbach, Germany
12. A. Kosmala, J. Rottland, G. Rigoll.: An Investigation of the Use of Trigraphs for Large Vocabulary Cursive Handwriting Recognition. In: Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, Munich, 1997
13. P.M. Lallican, C. Viard-Gaudin.: A Kalman Approach for Stroke Order Recovering from Off-line Handwriting. In: 4th Int. Conf. on Document Analysis and Recognition (ICDAR), pp. 519–522, 1997

14. S. Manke.: On-line Erkennung kursiver Handschrift bei grossen Vokabularen (in German). PhD thesis, University of Karlsruhe, 1998. Shaker, Germany
15. S. Manke, M. Finke, A. Waibel.: Combining Bitmaps with Dynamic Writing Information for On-Line Handwriting Recognition. In: Proc. 12th Int. Conf. on Pattern Recognition, pp. 596–598, 1994
16. S. Manke, M. Finke, A. Waibel.: *NPen*$^{++}$: A Writer Independent, Large Vocabulary On-Line Cursive Handwriting Recognition System. In: Proc. Int. Conf. on Document Analysis and Recognition (Montreal/Canada), 1995
17. S. Manke, M. Finke, A. Waibel.: A Fast Search Technique for Large Vocabulary On-Line Handwriting Recognition. In: Int. Workshop on Frontiers in Handwriting Recognition (IWFHR), Colchester, 1996
18. S. Ortmanns, H. Ney, X. Aubert.: A Word Graph Algorithm for Large Vocabulary Continuous Speech Recognition. Comput. Speech Lang., 11: 43–72, 1997
19. E.H. Ratzlaff, K.S. Nathan, H. Maruyama.: Search Issues in the IBM Large Vocabulary Unconstrained Handwriting Recognizer. In: Proc. Int. Workshop on Frontiers in Handwriting Recognition, Colchester, UK, 1996
20. M. Schenkel, I. Guyon, D. Henderson.: On-Line Cursive Script Recognition Using Time Delay Neural Networks and Hidden Markov Models. In: Proc. Int. Conf. on Acoustics, Speech, and Signal Processing, Adelaide, 1994
21. M.E. Schenkel.: Handwriting Recognition Using Neural Networks and Hidden Markov Models, volume 45 of Series in Microelectronics. Hartung-Gorre, Konstanz, 1995
22. G. Seni.: Large Vocabulary Recognition of On-Line Handwritten Cursive Words. PhD thesis, Department of Computer Science of the State University of New York at Buffalo, N.Y., USA, 1995
23. A. Waibel, T. Hanazawa, G. Hinton, K. Shiano, K. Lang.: Phoneme Recognition Using Time-Delay Neural Networks. In: IEEE Trans. on Acoustics, Speech, and Signal Processing, pp. 328–339, 1989

Stefan Jaeger was born in Trier, Germany, on June 23, 1967. He graduated from the University of Kaiserslautern, Germany, in Computer Science in 1994 and received his Ph.D. degree in Computer Science from Albert-Ludwigs University, Freiburg, Germany, in 1998. From 1994 to 1998 he worked as a Ph.D. student at the Daimler-Benz Research Center Ulm, Germany, where he was engaged in off-line cursive handwriting recognition for postal mail sorting. In 1998 he joined the Interactive Systems Laboratories located at Carnegie Mellon University, Pittsburgh, USA, and at the University of Karlsruhe, Germany, where he was a member of the Computer Science Research Staff and responsible for on-line handwriting recognition and pen-computing. He is currently working as an Invited Research Scientist at the Department of Computer, Information, and Communication Sciences at Tokyo University of A&T, Japan. His Ph.D. thesis addressed the problem of recovering dynamic information from static, handwritten word images, and was awarded the Dissertation Prize from the German Research Centers for Artificial Intelligence in 1999. His current research interests include handwriting recognition, pen-based interfaces, architectures for cognition and perception, and machine learning.

Stefan Manke was born in Germany in 1966. He received the diploma and Ph.D. degree in Computer Science in 1991 and 1998 from the University of Karlsruhe, Germany. His Ph.D. thesis laid the foundations of the NPen++ system and dealt with recognizing handwritten cursive script for large vocabularies. Since 1998 he has been with the TLC Company, the systems house of the German Federal Railway, where he is currently Systems Engineer for Inter- and Intranet Applications.

Juergen Reichert received his diploma in computer science from the University of Karlsruhe, Germany, in 1998. He is currently working as a Ph.D. candidate at the Interactive Systems Laboratories located at Carnegie Mellon University in Pittsburgh and at the University of Karlsruhe in Germany. His research interests include handwriting recognition, speech recognition, machine translation, multi modal user interfaces, and artificial intelligence.

Alex Waibel is a Professor of Computer Science at Carnegie Mellon University, Pittsburgh and at the University of Karlsruhe (Germany). He directs the Interactive Systems Laboratories at both Universities with research emphasis in speech recognition, handwriting recognition, language processing, speech translation, machine learning and multimodal and multimedia interfaces. At Carnegie Mellon, he also serves as Associate Director of the Language Technology Institute and as Director of the Language Technology PhD program. He was also one of the founding members of the CMU's Human Computer Interaction Institute (HCII) and continues on its steering committee. Dr. Waibel was one of the founders of C-STAR, the international consortium for speech translation research and currently serves as its chairman. He codirects Verbmobil, the German national speech translation initiative. His work on the Time Delay Neural Networks was awarded the IEEE best paper award in 1990, and his work on speech translation systems the Alcatel SEL research prize for technical communication in 1994. Dr. Waibel received his B.S. in Electrical Engineering from the Massachusetts Institute of Technology in 1979, and his M.S. and Ph.D. degrees in Computer Science from Carnegie Mellon University in 1980 and 1986.