NPEN++: AN ON-LINE HANDWRITING RECOGNITION SYSTEM

S. JAEGER, S. MANKE, A. WAIBEL
Interactive Systems Laboratories
University of Karlsruhe
Computer Science Department, 76128 Karlsruhe, Germany
and
Carnegie Mellon University
School of Computer Science, Pittsburgh, PA 15213-3890, USA
E-mail: {stefan.jaeger,waibel}@ira.uka.de

Abstract

This paper presents the on-line handwriting recognition system NPen++ developed at the University of Karlsruhe and the 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 5000 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 loosing too much recognition performance compared to an exhaustive search. This allows running the NPen++ recognizer in real-time with large dictionaries.

1 Introduction

This paper describes the preprocessing steps, the computation of features, the recognizer with training and testing, and the dictionary based search of the NPen++ handwriting recognition system. Section 2 begins with the description of the normalizing preprocessing steps in NPen++. Section 3 shows different features computed after preprocessing. The core of the NPen++ recognition engine, i.e. the Multi State Time Delay Neural Network (MSTDNN), is presented in Section 4. Training and recognizing are described in Section 5 and Section 7, respectively. Section 6 describes the search technique used in the NPen++ system. Section 8 contains a short summary that concludes this paper.

2 Preprocessing

Before NPen++ derives features from the handwritten word, the raw data recorded by the hardware goes through several preprocessing steps:

Computing baselines. NPen++ exploits four lines, including baseline and upper baseline, for normalizing sizes, correcting rotations, and deriving features. The baseline corresponds to the original writing line on which a word or text was written. The upper baseline goes through the top of lower case characters such as "n" or "w". In addition, NPen++ computes a third line that goes through ascenders (tops of characters such as "E" or "l") and a fourth line that goes through descenders (bottoms of characters such as "p" or "g"). In the NPen++ system, all four lines are defined as polynomials with Degree 2:

$$y = f_i(x) = k\,(x - x_0)^2 + s\,(x - x_0) + y_i \qquad i = 1, \ldots, 4 \tag{1}$$

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 using an Expectation-Maximization algorithm [1].

Normalizing size. NPen++ transforms every word to a given core height, where the core height is defined as the distance between baseline and upper baseline.

Normalizing rotations. Rotations of words, i.e. deviations of the computed baseline from the horizontal writing line, are corrected by means of a simple geometric rotation that is based on the Parameter s denoting the rotation of both baselines.

Interpolating missing points. If the distance between two neighboring points exceeds a certain threshold, we interpolate the trajectory between both points using a Bezier curve. This preprocessing step may be necessary in situations where the hardware is not able to capture all points of the trajectory or can not catch up with the speed of writing.

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

Normalizing inclination. To normalize the different slants of words, we shear every word according to its angle of inclination. This angle 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 the distance of every pair of successive points. A search for the maximum entry in the histogram provides us with the slant of the word [7, 10].

Computing equidistant points (Resampling). 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 used

hardware. NPen++ replaces the captured points with points having the same spatial distance.

Removing delayed strokes. Delayed strokes, e.g., the crossing of a "t" or the dot of an "i", introduce additional temporal variation and complicate on-line recognition because the writing order of delayed strokes is not fixed and varies between different writers. This is an important difference to off-line 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 that tried to exploit off-line data in order to improve on-line recognition rates [4, 7]. NPen++ exploits simple heuristics for detecting delayed strokes and removes them after setting a flag in the feature vector.

3 Computing Features

This section presents the features NPen++ computes from the normalized sequence of captured coordinates $(x(t), y(t))$:

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 core height of the word [7].

Writing direction. The local writing direction at a Point $(x(t), y(t))$ is described using the cosine and sine [2]:

$$\cos\ \alpha(t) = \frac{\Delta x(t)}{\Delta s(t)}, \tag{2}$$

$$\sin\ \alpha(t) = \frac{\Delta y(t)}{\Delta s(t)}, \tag{3}$$

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

$$\Delta s(t) = \sqrt{\Delta x^2(t) + \Delta y^2(t)}, \tag{4}$$

$$\Delta x(t) = x(t-1) - x(t+1), \tag{5}$$

$$\Delta y(t) = y(t-1) - y(t+1). \tag{6}$$

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 [2]:

$$(x(t-2), y(t-2)), (x(t), y(t)), (x(t+2), y(t+2)). \tag{7}$$

Cosine and sine can be computed using the values from the direction of writing:

$$\cos\beta(t) \;=\; \cos\alpha(t-1) \times \cos\alpha(t+1) \;+\; \sin\alpha(t-1) \times \sin\alpha(t+1), \qquad (8)$$

$$\sin\beta(t) \;=\; \cos\alpha(t-1) \times \sin\alpha(t+1) \;-\; \sin\alpha(t-1) \times \cos\alpha(t+1). \qquad (9)$$

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.

"Hat"-Feature. This is a simple, binary feature that indicates if the current position is below a delayed stroke, e.g. below a t-stroke [10].

Aspect. The aspect of the trajectory in the vicinity of a point $(x(t), y(t))$ is another local feature computed in NPen++ [10], which is described by a single value $A(t)$:

$$A(t) = \frac{2 \times \Delta y(t)}{\Delta x(t) + \Delta y(t)} - 1, \qquad (10)$$

which characterizes the ratio of height to width of the bounding box containing the preceding and succeeding points of $(x(t), y(t))$. Figure 1 illustrates the computation of the aspect. The vicinity of a point $(x(t), y(t))$ is also used to

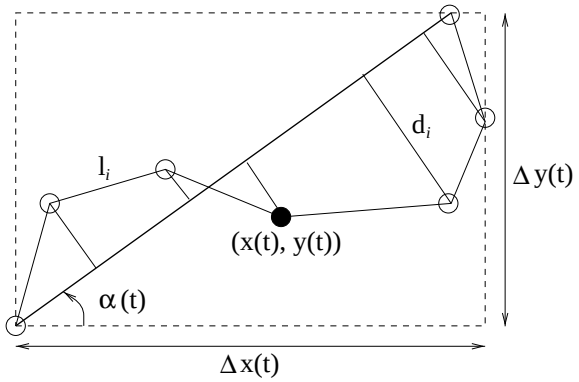

Figure 1: Aspect.

define the following three features: curliness, lineness, and slope.

Curliness. Curliness is a feature that describes the deviation from a straight line in the vicinity of $(x(t), y(t))$ (see Figure 1). It is defined as follows:

$$C(t) = \frac{L}{max(\Delta x, \Delta y)} - 2, \tag{11}$$

where L is the length of the trajectory in the vicinity of $(x(t), y(t))$, i.e. the sum of lengths of all line segments. Δx and Δy are the width and height of the bounding box containing all points in the vicinity of $(x(t), y(t))$ (see Figure 1).

Lineness. 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 Lineness, which is defined as follows:

$$L(t) = \frac{1}{N} * \sum_i d_i^2. \tag{12}$$

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 Figure 1).

Ascenders/Descenders. These are two global features that count the number of points above the upper baseline (ascenders) and the number of points below the baseline (descenders) in the off-line image at a given time instance t. Every point is weighted with its distance to the upper baseline (ascenders) or baseline (descenders). The main idea of these features is to help identifying specific characters, like "t" or "g", using a wider temporal context.

Context Bitmaps. A context bitmap is an off-line, 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 bitmap is a low resolution image with the pixel in the center containing $(x(t), y(t))$. The size of the context bitmap depends on the core height of the word. In our experiments, we have transformed the computed bitmap to a 3×3-bitmap before adding it to the feature vector [8]. Every pixel of a context bitmap is a feature and thus added to the set of features described above. The main idea of context bitmaps is to add features that consider a wider context. Context bitmaps 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 bitmaps are an example of combining off-line and on-line information in handwriting recognition [8]. This is an interesting research topic, which has been investigated by several researchers in the recent past [4, 5].

4 Multi-State Time Delay Neural Networks (MS-TDNN)

The core recognition engine of NPen++ is a Multi-State Time Delay Neural Network (MS-TDNN [3, 11]). A MS-TDNN is a connectionist recognizer that integrates recognition and segmentation into a single network architecture. The MS-TDNN is an extension of the Time Delay Neural Network (TDNN) [11], which has been applied successfully to on-line *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. A MS-TDNN recognizes words by integrating a dynamic time warping algorithm (DTW) into the TDNN architecture. 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 modelled 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 neural networks and hidden Markov models (HMMs). Figure 2 shows the basic architecture of the MS-TDNN. The first three layers, which

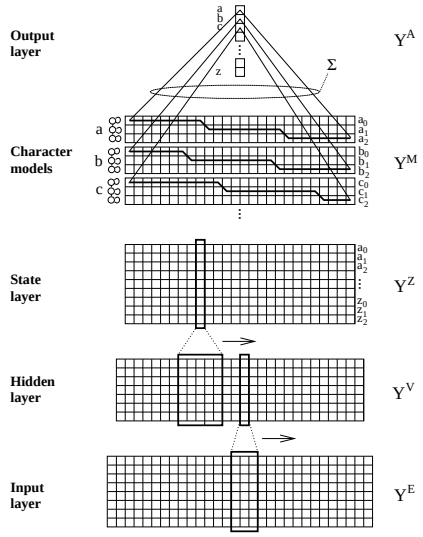

Figure 2: The architecture of a Multi-State Time Delay Neural Network.

are respectively called input layer, hidden layer, and state layer, constitute a standard TDNN. Every element ("neuron") in the state layer represents a state of a character in the alphabet. The Viterbi algorithm computes the score of

a word by finding an optimal alignment path through the states of characters composing the word and summing all activations along this path.

5 Training

The MS-TDNN is trained in three steps with standard back-propagation. The first two training steps are based on a forced alignment mode, during which the MS-TDNN is trained with hand-segmented training data. The first step assumes that the Viterbi path remains for the same time in each state of a word of the training set. The activations along this Viterbi path constitute the training data for the back-propagation procedure. In Step 2, the assumption that the Viterbi path remains for the same time in every state is abandoned in favor of computing the actual Viterbi path through a character model. The third step commences by replacing the forced alignment in Step 1 and Step 2 with the free alignment provided by the Viterbi algorithm. This has the advantage that training can be performed on unsegmented data. Thus, only a small part of the training data must be labeled manually with character boundaries to accomplish 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. The number of iterations in each step is optimized in practical experiments using cross validation sets, which are independent from the test sets.

In our experiments, we used the cross entropy for propagating the error E_{CE} back in the MS-TDNN [7]:

$$E_{CE} = -\sum_j [d_j \log(y_j) + (1 - d_j) \log(1 - y_j)], \tag{13}$$

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

6 Search Technique

NPen++ supplements the MS-TDNN approach with a tree-based search engine [9]. It combines a tree representation of the dictionary with efficient pruning techniques to reduce the search space without loosing much recognition performance compared to a flat exhaustive search through all words in the dictionary. NPen++ builds search trees for each character in the alphabet, where nodes in a tree are HMMs representing individual characters. Each 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 in the dictionary. Figure 3 shows a dictionary represented as a tree. In order to achieve

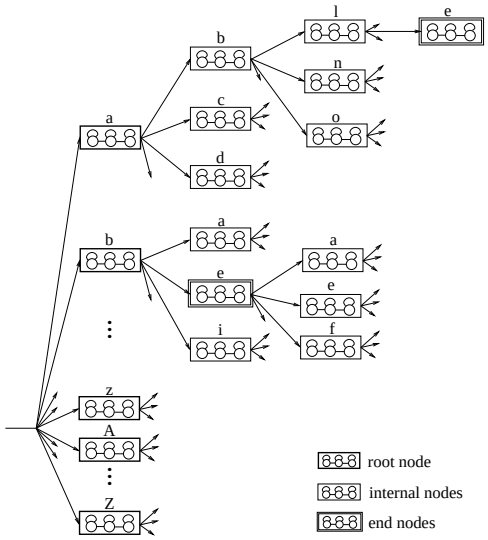

Figure 3: Tree representation of a dictionary.

real-time performance for very large dictionaries, NPen++ abandons applying the exact Viterbi algorithm. Instead of that, 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 supposed 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:

$$\hat{s}_i < \hat{s} - beam \tag{14}$$

  Thus, all nodes whose best accumulated score is below a certain threshold, called beam, will become inactive in the next frame.

- **Expansion:** For every node being active 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. This is shown in the next section.

7 Evaluation

Most of the design parameters in NPen++ were determined empirically during several training and test cycles, with only a few exceptions. The number of input units ("input neurons") is identical to the number of features described in Section 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 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.

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 20000 word dictionary contains the most likely 20000 words from the Wall Street Corpus and thus contains any smaller dictionary derived from this corpus.

We used three different databases for training and testing: the CMU database collected at the Carnegie Mellon University, the UKA database collected at the University of Karlsruhe, and the MIT database collected at the Massachusetts Institute of Technology [7]. While the CMU and the UKA databases contain printed and cursive handwriting data, the MIT database only contains printed data (mixed styles are considered cursive). The UKA database contains approximately 7000 words written by 114 writers and the CMU database contains approximately 12500 words from 215 writers. The database collected at the MIT contains about 8400 words from 159 writers [6]. A detailed description of all three databases is given in [7]. Since we need some hand-segmented data to start training (see Section 5), we explicitly segmented 4000 cursive words from the UKA database and 2000-cursive words from the CMU database by visual inspection.

Figure 4 shows the recognition rates of NPen++ on different data sets. As

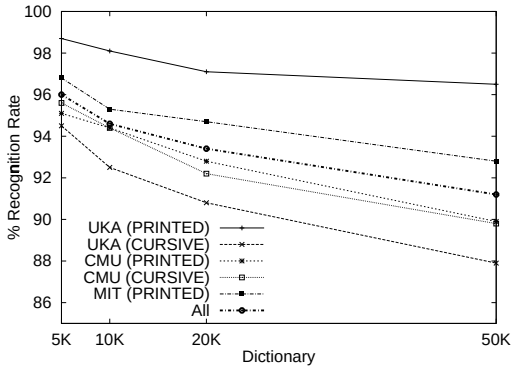

Figure 4: Recognition rates computed on different dictionaries.

one would expect, the recognition rates for printed data are higher than recognition rates for cursive data. NPen++ achieves recognition rates of 96% for a 5000 word dictionary when it is trained and tested with data taken from each database. The number of false classified samples approximately doubles when the ten times larger 50000 word dictionary is used instead. The recognition rate is 91.2% for this dictionary. In our experiments, recognition rates with no context bitmaps included in the feature vector are more than one percent less on the 20000 words dictionary, compared to the rate shown in Figure 4. Figure 5 illustrates the tradeoff between recognition time and recognition accuracy measured on the 20000 words dictionary using test and training data from

every data set. The left-hand graphic of Figure 5 shows the recognition times depending on the width of the beam b (see Section 6). The rightmost column

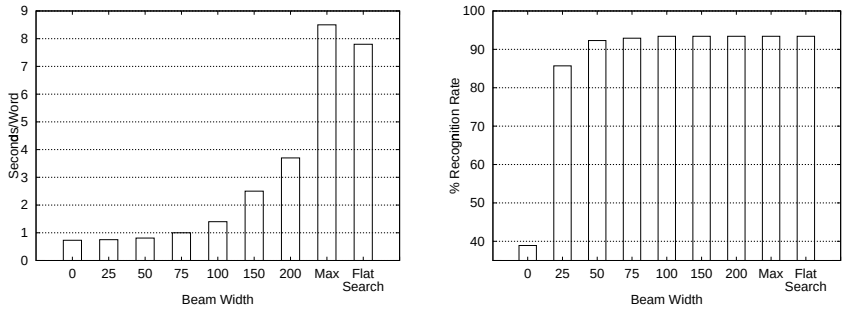

Figure 5: Recognition times and recognition rates on a 20k dictionary.

shows the recognition time for the flat search with no pruning. The column on its left 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 standard PC containing a Pentium Processor and 64 MB SDRAM.

The right-hand graphic in Figure 5 shows the recognition rates depending on the width of the beam. They remain almost constant for beam widths equal to or higher than 50. Hence, we can considerably reduce recognition time without loosing recognition performance by reducing the beam width.

8 Summary

NPen++ integrates local as well as global information about the trajectory of the pen into the feature vector. The core of NPen++ is a Multi-State Time Delay Neural Network, which is a hybrid architecture between neural networks and hidden Markov models. Training consists of a forced alignment and a free alignment mode, which requires only a small set of hand-segmented data. NPen++ uses an efficient tree search and pruning technique to ensure real-time performance for very large dictionary sizes. The recognition rates range from 96% for a 5000 word dictionary to 91.2% for a 50,000 word dictionary.

References

1. Y. Bengio and Y.L. Cun. Word Normalization for On-Line Handwritten Word Recognition. In *Proceedings of the International Conference on Pattern Recognition*, pages 409–413, 1994.
2. 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):105–119, 1991.
3. H. Hild and A. Waibel. Speaker-Independent Connected Letter Recognition with a Multi-State Time Delay Neural Network. In *3rd European Conference on Speech, Communication and Technology (EUROSPEECH)*, volume 2, pages 1481–1484, Berlin, 1993.
4. S. Jaeger. *Recovering Dynamic Information from Static, Handwritten Word Images.* PhD thesis, University of Freiburg, 1998. Foelbach Verlag.
5. S. Jaeger. On the Complexity of Cognition. In *7th International Workshop on Frontiers in Handwriting Recognition (IWFHR)*, Amsterdam, 2000.
6. R. H. Kassel. *A Comparison of Approaches to On-Line Handwritten Character Recognition.* PhD thesis, Massachusetts Institute of Technology, 1995.
7. S. Manke. *On-line Erkennung kursiver Handschrift bei grossen Vokabularen (in german).* PhD thesis, University of Karlsruhe, 1998. Shaker Verlag.
8. S. Manke, M. Finke, and A. Waibel. Combining Bitmaps with Dynamic Writing Information for On-Line Handwriting Recognition. In *Proc. of the 12th International Conference on Pattern Recognition*, pages 596–598, 1994.
9. S. Manke, M. Finke, and A. Waibel. A Fast Search Technique for Large Vocabulary On-Line Handwriting Recognition. In *International Workshop on Frontiers in Handwriting Recognition (IWFHR)*, Colchester, 1996.
10. M. E. Schenkel. Handwriting Recognition Using Neural Networks and Hidden Markov Models. *Series in Microelectronics*, volume 45, 1995.
11. A. Waibel, T. Hanazawa, G. Hinton, K. Shiano, and K. Lang. Phoneme Recognition Using Time-Delay Neural Networks. In *IEEE Transactions on Acoustics, Speech, and Signal Processing*, pages 328–339, 1989.