

 Open access • Proceedings Article • DOI:10.1109/CVPR.1994.323858

## An on-line cursive word recognition system — Source link

Seni, Nasrabadi, Srihari





**Institutions:** State University of New York System

**Published on:** 21 Jun 1994 - Computer Vision and Pattern Recognition

**Topics:** Intelligent word recognition, Intelligent character recognition, Time delay neural network, Feature (machine learning) and Cursive

Related papers:

- [An On-Line Cursive Word Recognition System](#)
- [Large vocabulary recognition of on-line handwritten cursive words](#)
- [A connectionist recognizer for on-line cursive handwriting recognition](#)
- [Connectionist architectural learning for high performance character and speech recognition](#)
- [Using constrained snakes for feature spotting in off-line cursive script](#)

Share this paper:    

View more about this paper here: <https://typeset.io/papers/an-on-line-cursive-word-recognition-system-z1xpfmss2l>

# An On-Line Cursive Word Recognition System

Giovanni Seni<sup>†</sup>, Nasser Nasrabadi<sup>‡</sup>, Rohini Srihari<sup>†</sup>

<sup>†</sup>Center of Excellence for Document Analysis and Recognition

<sup>‡</sup>Department of Electrical and Computer Engineering  
State University of New York at Buffalo  
Buffalo, NY 14260

## Abstract

*This paper presents a system for large vocabulary recognition of on-line handwritten cursive words. The system first uses a filtering module, based on simple letter features, to quickly reduce a large reference dictionary to a smaller number of candidates; the reduced lexicon along with the original input is subsequently fed to a recognition module. In order to exploit the sequential nature of the temporal data, we employ a TDNN-style network architecture which has been successfully used in the speech recognition domain. Explicit segmentation of the input words into characters is avoided by using a sliding window concept where the input word representation (a set of frames) is presented to the neural network-based recognizer sequentially. The outputs of the recognition module are collected and converted into a string of characters that can be matched with the candidate words. A description of the complete system and its components is given.*

## 1 Introduction

Computer recognition of on-line handwriting offers a new way of improving the human-computer interface. Since handwriting is one of the most familiar communication media, an automatic handwriting recognition system can offer a very easy and natural input method. However, most of the research effort in this area has been devoted to the recognition of isolated characters (particularly important for large-alphabet languages such as Chinese, with over 3000 different ideographs) [7, 12], or run-on hand-printed words [15, 5]. A significantly smaller number of recognition systems have been devised for cursive words [16, 13, 4], a more difficult task due to the presence of the letter segmentation problem (partitioning the word into letters), and generally larger variations at

the letter level. Most of these systems restrict the working dictionary sizes to less than few thousand words.

Two major approaches have traditionally been used in cursive handwriting recognition, whether online or offline: *segmentation-based* and *word-based* (also refer to as ‘holistic’). In the segmentation-based approach each word is segmented into its component letters and a recognition technique is then used to identify each letter. Unfortunately, the nature of cursive script is such that the letter segmentation points<sup>1</sup> can only be correctly identified when the correct letter sequence is known, and the recognition of characters can only be done successfully when the segmentation is correct. Therefore, a recognition engine that performs character recognition and segmentation in parallel is desirable. Segmentation-based systems also fail to account for handwriting variations due to co-articulation (the influence of one letter on another) and can suffer from combinatorial complexity when combining multiple decisions about individual characters.

In the word-based approach, individual letters are not recognized as such but a global feature vector is extracted from the input word which is matched against a stored dictionary of prototype words; a distance measure is used to choose the best candidate. This word recognition method has the advantage of speed, and avoids problems associated with segmentation. This method reflects the human reading process which is not character by character, but rather by words or even phrases. The main disadvantage of this method is the need to train the machine with samples of each word in the established dictionary, thereby constraining vocabulary size.

In our recognition system, we adopt an intermediate position between the above fundamentally different approaches, and attempt to incorporate the follow-

---

<sup>1</sup>Points where one letter ends and the succeeding one begins.

ing three concepts regarding the cognition of cursive handwriting. First, the perception of words by humans is a two step process: characteristic letters are found in the word image which are used to select candidate words; an attempt is then made to align these words with the input image. Second, the dynamic pattern of motion in cursive handwriting carries more information and less variability than the static geometric representation of a word. Third, separating a character from its background is not a necessary pre-processing step for identifying the character.

Accordingly, the system first uses a *filtering module* that extracts a structural description for a given input word and uses it to reduce quickly a large lexicon (i.e., more than 20,000 words) to a smaller set of *matchable* words, and then a neural network-based *recognition module* takes a temporal representation of the input word and identifies each of its letters (alphabets) without performing an explicit segmentation step. The predicted word is then compared with the possible matchable words. A narrowed-down lexicon will limit the amount of work to be done during this stage.

Section 2 shows the architecture of the system. Section 3 presents the filtering module. Section 4 describes the recognition module. The final section summarizes and discusses future work.

## 2 Proposed recognition system

The structure of the proposed cursive word recognition system is shown in Figure 1. Notice that both the filtering module and the recognition module works on data which is preprocessed. This is necessary because the output of the digitizing tablet is noisy (due to quantization effects and the shaking of the hand) and usually contains too many points. Data reduction and enhancement is achieved by a resampling algorithm which includes duplicated points removal, enforcing even spacing between points and smoothing (see [6] for an overview of these operations). Equalization of different writing orientations, writing slant, and writing sizes is also desirable in order to reduce writer dependent variability. A normalization algorithm is provided for this purpose based on the work of [1],[17].

The filtering module (Lexicon reduction module) starts by deriving a *description string*  $\alpha = \alpha_1 \alpha_2 \dots \alpha_n$  where  $\alpha_i$  corresponds to a key feature found in the input word. Such string is then passed to a procedure *search*( $\alpha$ ) which has knowledge about how to derive ASCII letters from the symbols  $\alpha_i$  and uses

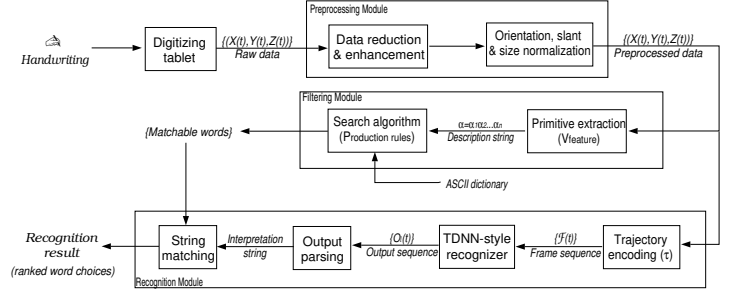


Figure 1: Overview of proposed model for large vocabulary recognition of on-line handwritten cursive words.

it to generate matchable words. Specifically, the problem is that of establishing a grammar  $G_{filter} = (V_{ascii}, V_{feature}, P, S)$ , where the set  $V_{ascii}$  of terminal symbols is the English alphabet, the set  $V_{feature}$  of non-terminal symbols is made of those basic elements in terms of which all letters can be described,  $P$  is the set of production rules which define the valid combinations of these elements to generate letters, and  $S$  is the starting (or root) symbol. The set of matchable words is the set of strings  $\beta$  which constitute valid English words and can be derived from  $\alpha$  (i.e.,  $\alpha \xRightarrow{*} \beta$ ).

The recognition module uses a temporal representation of the input, thereby preserving the sequential nature of the cursive data and enabling the use of a Time-Delay Neural Network (TDNN)-style architecture [18]. This recognizer is trained to classify the signal within its *fixed-size* input window as this window sequentially *scans* the input word representation, thus allowing us to build a system that no longer depends on a potentially errorful segmentation procedure during the recognition. By training and recognizing characters in context<sup>2</sup> (i.e., we define a large enough input window to include more than one character) the co-articulation phenomena is accounted for. Finally, the recognizer's outputs are collected and converted into an ASCII string that can be matched against the reduced lexicon.

## 3 Filtering module

According to the *harmonic oscillator* description of the muscle action involved in handwriting production [9], cursive handwriting generation can be viewed as a sequential modulation of two coupled oscillations; one in the vertical direction and one in the horizontal direction. In this context, it is natural to charac-

<sup>2</sup>By character context we mean a small portion of the word image that precedes and follows the given character.

terize the writing as an ordered sequence of upward and downward strokes. However, it has been previously suggested that downward strokes in the word are more important than upward strokes because they are always part of the letters while the former ones sometimes act as joining strokes [1]. Therefore, we choose to extract downward strokes only which are subsequently classified based on their relative heights and direction of movement. They constitute the set of key features which are going to be used to describe the structure of the words. Accordingly, grammar  $G_{filter}$  is currently defined as follows:

$$G_{filter} = (V_{ascii}, V_{feature}, P, S)$$

where

$$\begin{aligned} V_{feature} &= \{A, D, M, B, C, K, L, R, U\} \\ V_{ascii} &= \{a, b, c, \dots, z\} \\ P &= \{A \rightarrow b|d|f|h|k|l|t \\ &\quad D \rightarrow f|g|j|p|q|y|z \\ &\quad M \rightarrow a|c|e|i|m|n|o|r|s|u|v|w|x \\ &\quad \dots \\ &\quad AM \rightarrow b|h|k \\ &\quad RD \rightarrow g|q \\ &\quad \dots \\ &\quad MMM \rightarrow m|w \\ &\quad RDM \rightarrow q\} \end{aligned}$$

In total there are 64 production rules; therefore, only a few of them are shown. Referring to the lines illustrated in Figure 2 the following interpretation of the symbols in  $V_{feature}$  is used: ,

- A Ascender stroke (a stroke that substantially extends from the half-line to the upper region of the word);
- D Descender stroke (a stroke that substantially extends from the base-line to the lower region of the word);
- M Median stroke (a stroke that lies between the half-line and the base-line of the word);
- B Both stroke (a stroke that expands both the upper region and the lower region of the word);
- C Connection stroke (a stroke that lies above the center line between the half-line and the base-line);
- K K-stroke (the middle downward stroke in a letter 'k');
- L L-stroke (a retrograde stroke pointing leftwards);

- R R-stroke (a retrograde stroke pointing rightwards);
- U Unknown stroke (a stroke with an ambiguous classification).

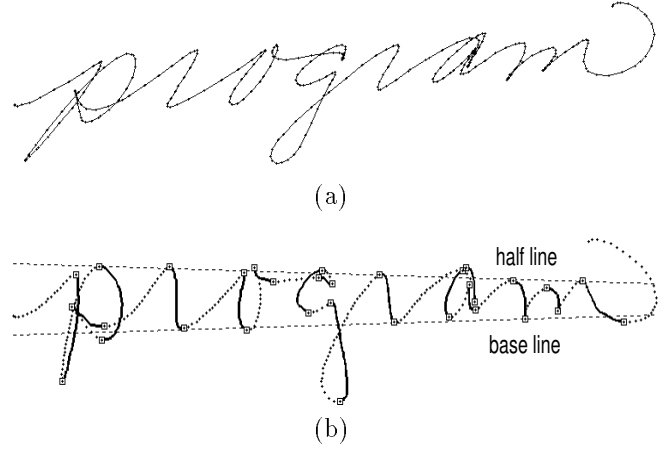


Figure 2: The filtering module: (a) a raw word image, and the (b) preprocessed image shown with base-line, half-line and extracted downward strokes. The associated code word is 'ULMMCRDMMMMM' and the reduced lexicon is {'program', 'programmer', 'programs'}.

The searching procedure uses a trie [10] representation of the dictionary to check if derived strings constitute valid English words. The final set of matchable words is further pruned if any diacritical mark (points on "i", "j", "t" bars, and "x" slash) is detected in the input image.

### 3.1 Discussion of filtering module

Care has been taken in the selection of features and derivation rules since exclusion of the correct word at this stage may deter effective recognition in the subsequent stage. Tested on a database of 750 cursive words (3 to 8 letters long) written by 10 writers, using a lexicon of 21,000 words, the reduced lexicon contains the correct word in 733 cases (i.e., 97.7% accuracy). The size of the correctly pruned lexicon is 112.8 words on average (i.e., 99.4% reduction efficacy) and 3103 words in the worst case. It took the filtering module, running on a Sun IPC (concurrently with other light processes), 9 secs in the worst case to output the reduced lexicon for an eight letter word.

## 4 Recognition module

On-line data represents text as a coordinate sequence  $\{P(t) = (X(t), Y(t), Z(t))\}$ , where  $X, Y$  are

the coordinates of the pen tip, and  $Z$  corresponds to the pen-up/pen-down indication. Although all the dynamic information about handwriting can be presumably inferred from this sequence, this data is too unconstrained so we must make a decision about how to encode it. However, we want to avoid the handcrafting involved in selecting features, a process which could result in the discarding of information essential for recognition. Therefore, we choose mainly to encode information about the local direction and curvature in the pen trajectory, and rely on the neural network-based recognizer for the selection of features relevant for performing the classification task.

Two parameters are used in the coding of direction: (i) sine of the angle between each segment  $(X(t), Y(t)) - (X(t-1), Y(t-1))$  of the trajectory and the Y-axis ( $\sin \theta_y(t)$ ), and (ii) sine of the angle between  $(X(t), Y(t)) - (X(t-1), Y(t-1))$  and the X-axis ( $\sin \theta_x(t)$ ). By restricting these angles to vary between  $-\pi/2$  and  $+\pi/2$  we make the parameters unambiguous; a negative value of  $\sin \theta_y(t)$  will indicate that point  $P(t) = (X(t), Y(t))$  is *before* point  $P(t-1) = (X(t-1), Y(t-1))$  (i.e., a backward pen movement was made in going from  $P(t-1)$  to  $P(t)$ ), and a positive value will indicate that point  $P(t)$  is *after* point  $P(t-1)$  (i.e., a forward pen movement was made). Similarly, the sign of  $\sin \theta_x(t)$  will indicate whether point  $P(t)$  is above or below point  $P(t-1)$  (i.e., if an upward or downward pen movement was made).

In addition to the directional information, the location of the points in the trajectory at which sharp changes in the direction of movement (i.e., cusps) take place is found. A very simple measurement of the ‘local’ curvature can be obtained by calculating the change between two consecutive directional angles. Guyon et al.[7] suggested to represent the angle  $\phi(t) = \theta_x(t+1) - \theta_x(t-1)$  by its sine and cosine values. However, we found that the values of  $\cos \phi(t)$  behave more smoothly than that of  $\sin \phi(t)$ ; the reason being that for small values of  $\phi(t)$  (i.e., little change in direction),  $\cos \phi(t)$  remains flat and at the high value of  $+1$  while  $\sin \phi(t)$  will oscillate around zero. We choose  $\cos \phi(t)$  as our only curvature descriptor: it will go down to  $-1$  for sharp cusps (independently of their orientation) and down to around 0 for more smoother turns.

#### 4.1 Varying duration and scaling

The trajectory representation is made scale invariant by normalizing the size of the word with respect to the vertical axis to a given height  $\mathcal{H}$ , while maintaining

the same aspect ratio. Thus, the height of small letters (those that fall between the base-line and the half-line) is approximately fixed for all words. Since the distance between points is kept constant, the above procedure effectively minimizes time distortions of letters.

#### 4.2 Time frames

Given a sequence  $\{(X(t), Y(t), Z(t))\}$  of on-line data, we define a time frame  $\mathcal{F}(t)$  to be a 4-dimensional feature vector consisting of four elements  $(\sin \theta_x(t), \sin \theta_y(t), \cos \phi(t), \text{zone}(Y(t)))$ , where the first three elements have already been described above. The fourth element,  $\text{zone}(Y(t))$ , is introduced to help distinguish pairs such as ‘e-l’ and ‘a-d’, which have similar temporal representations. These pairs can be differentiated by encoding their corresponding  $Y(t)$  values into the previously determined zones: the middle zone (between the base-line and the half-line), the ascender zone (above the half-line) and the descender zone (below the base-line). The frame sequence  $\{\mathcal{F}(t)\}$  constitutes an intermediate representation of the on-line data and is used as the input to the neural network recognizer.

#### 4.3 Neural network recognizer

Our current three layer TDNN-style network is shown in Figure 3. The input to the neural network is a window of  $L = 96$  frames (network receptive field) which are extracted from the input frame sequence  $\{\mathcal{F}(t)\}$ . The input window is continuously moved across the frame sequence  $\{\mathcal{F}(t)\}$  thus generating activation traces  $\mathcal{O}_l(t)$  at the output of the network where  $\mathcal{O}_l(t)$  corresponds to the network’s confidence in recognizing a letter  $l$  at time  $t$ . These output traces would be subsequently examined to determine the most likely interpretation of the input word. The input window is shifted by  $\mathcal{S}=3$  frames between successive generations of the output activation trace.

The weight connection in the network is arranged such that each hidden unit has a receptive field that is limited by a time delay; each neuron’s decision at time  $t$  is based on the previous frames  $\mathcal{F}(t), \mathcal{F}(t-1), \dots, \mathcal{F}(t-N-1)$  where  $N$  is the receptive field’s size or the number of delays. In the first hidden layer there are 30 frames  $\mathcal{F}_{L_1}(t)$  and each frame consists of 8 feature detecting units interconnected to the input layer with 9 delays. That is, each unit in the first hidden layer receives input from 9 consecutive frames in the input layer; therefore, there are 36 connections. The particular choice of 9 delays comes from the goal of detecting features with short duration at this level,

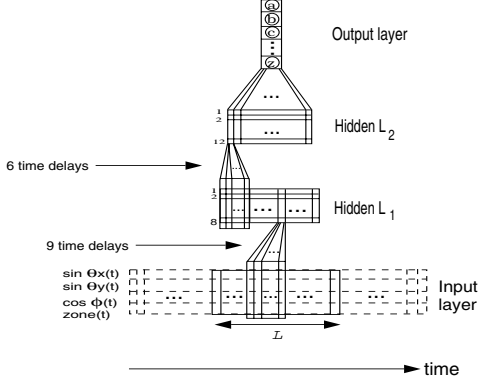


Figure 3: The architecture of a TDNN-style network for cursive word recognition.

but also long enough for each unit to detect a meaningful feature (e.g., a cusp). The receptive fields of two consecutive units in the first hidden layer overlap by 6 frames. In the second hidden layer, there are 9 frames  $\mathcal{F}_{L_2}(t)$  each with 12 feature detecting units and looking at a 6 frame window of activity levels in the first hidden layer. These units receive a larger time spans information from the input, and hence are expected to detect more complex and global features (i.e., longer in duration). The receptive fields of two consecutive units in the second hidden layer overlap by 3 frames. Finally, the second hidden layer is fully connected to the output layer which has 24 units (one for each of the English letters except ‘t’ and ‘x’ which we are not currently dealing with since they require special treatment of their crossing stroke). In the current network there are a total of 660 units.

The choice of  $L=96$  frames as the size of the input window to the network is related to  $\mathcal{H}$  (currently set at about 3mm), the normalization height.  $\mathcal{H}$  is selected such that  $L$  frames are enough to represent a character and, in most cases, include part of the characters on each side of it for contextual information.

Weights are also *shared*: the weights connecting one frame  $\mathcal{F}_{L_i}(t)$  in layer  $L_i$  to its corresponding receptive field (in the layer below) are the same as the weights connecting the next frame  $\mathcal{F}_{L_i}(t+1)$  in layer  $L_i$  to its corresponding receptive field. Therefore, in total there are only 3428 independent weights. Weight sharing is a general paradigm that allows us to build reduced size networks. Minimizing the number of free parameters in the network (i.e., weights that must be determined by the learning algorithm) is an effective way of increasing the likelihood of correct generalization [11]. Weight sharing also guarantees invariance under translation in time [14]; by constraining all the frames

to learn the same pattern of weights, each frame computes the same set of features over its receptive field as the neighboring frames do. Intuitively, if a particular feature detector is useful on one part of the sequence, it is likely to be useful on other parts of the sequence as well, particularly if such a feature appears in the input, displaced from its ideal position.

#### 4.4 Training data set

In order to perform training, with back-propagation algorithm, a training data set is created by labelling each word sample with the positions of each inter-character boundary (where roughly one character ends and the next one begins). This information is then used to pair each frame  $\mathcal{F}(t)$ , in the dynamic representation of the word, with an output vector. The goal is to generate a target signal that ramps up about halfway through the character and then quickly backs down afterwards, so that the network learns to recognize this character whenever its center is in the middle of the network’s receptive field. For every character in every word of the training data set, a target signal that ramps up at 30% of the character’s length, reaches its maximum between 45% and 55% of the character’s length, and backs down to its minimum afterwards was generated.

About 25 different words were randomly selected from a 60,000 words dictionary keeping the letter frequency ‘roughly’ uniform (the maximum letter frequency is about three times the minimum letter frequency in the set). Three different writers were asked to write these words 4 times, using a WACOM SD-311 digitizing tablet, resulting in 225 words for training and 75 words for testing. All 225 training words were concatenated together, after preprocessing and size-normalization, into a sequence of 64,596 pairs (frame,target).

#### 4.5 Network simulation

The activation range of the neurons is chosen to be between  $-1$  and  $+1$  with the following computationally efficient squashing function [2] :

$$f(u) = \frac{u}{1+|u|}, \quad f'(u) = \frac{1}{(1+|u|)^2} + offset$$

where  $|u|$  stands for the absolute value of the weighted sum and *offset* is a constant suggested by Fahlman [3] to kill flat spots. Weights are initialized with random numbers uniformly distributed between  $-0.1$  and  $+0.1$ . A single bias unit is used by all weight-shared units that are controlled by the same weight

kernel, as opposed to an independent bias per unit (we found no reason to have independent bias units in order to develop truly invariant feature detectors).

The use of an *error tolerance*<sup>3</sup> during training was very helpful in mediating the disproportion between the training samples with negative target values (negative evidence indicating that the network should not respond) and the training samples with positive target values (positive evidence indicating the network should respond). We start this parameter at 0.3 and subsequently gradually reduced it to 0.1. All our simulations have been performed with an in-house developed simulator written in ANSI C. A Mean Squared Error (MSE) of 0.17 was obtained after a training session of 140 iterations using an error tolerance of 0.3 during the first hundred iterations, and then 0.15 and 0.1 for the next thirty and ten iterations respectively.

## 4.6 Output trace parsing

In order to convert the output trace signal  $\mathcal{O}_i(t)$  generated by the network into an ASCII string, the sizes and widths of all activation ‘peaks’ for every output unit are determined. Figure 4b shows the output activation traces, for all the 24 output units, generated by the network when presented with the word ‘vainer’ from our testing data set. Eight different activation peaks are clearly visible; the six larger ones corresponding to a letter in the word.

The sizes of all activation peaks are computed by scanning the output activation traces, from left to right, looking for activation levels that exceed a given threshold. When the activation value of an output unit exceeds the threshold (currently set at  $-0.8$ ), a summing process begins for the unit, that ends when it’s activation value falls below the threshold. The width of a peak is defined as the maximum of the actual peak width ( $finish\_time - beginning\_time$ ) and the expected peak width. The expected peak width for a given letter is given by the average width of all the peaks in the training signal for that letter.

Activation peaks with a maximum value below  $-0.2$  are ignored (i.e., they are not considered confident enough). In order to compensate for smaller letters, which are shorter in the temporal domain, we normalize the size of a peak by its expected average size [8]. The expected average peak size is computed in a similar way to the expected peak width. The set  $\{P_i\}$  of

all selected activation peaks is ordered based on the beginning time of each peak  $P_i$ .

In order to handle overlapping peaks, we construct an interpretation directed graph from the ordered set of activation peaks as follows: there is a node  $N_i$  in the graph for every activation peak  $P_i$ , and there is an edge between nodes  $N_i$  and  $N_j$  ( $i < j$ ) if peaks  $P_i$  and  $P_j$  are adjacent and do not overlap; otherwise, nodes  $N_i$  and  $N_j$  will lie on parallel paths of the graph. In Figure 4b all activation peaks that reach a maximum value above  $-0.2$  are shown with their expected widths ( $\leftrightarrow$ ) centered around the middle of each peak. Figure 4c shows the associated interpretation graph: the number next to each node is the size of the corresponding activation peak. Word hypotheses are generated by traversing all possible paths in the graph from the root to all the ‘leaves’. We currently set the confidence of a word hypothesis as the product of the node’s values in the corresponding path.

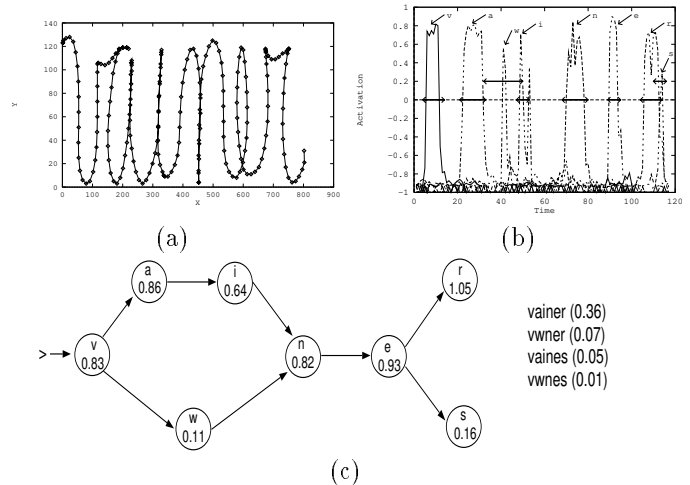


Figure 4: The operation of the output trace parsing algorithm: (a) the preprocessed image of a word ‘vainer’, (b) the plot of the corresponding network output traces (selected activation peaks are shown with their expected peak width), and (c) the associated interpretation graph and generated word hypotheses (nodes of the graph are shown with their corresponding peak’s sizes).

## 4.7 Discussion of recognition module

Without any additional postprocessing than what was described in the previous section, the top choice word recognition rate of the recognition module was 76% on the testing set. With elementary spell checking of the top choice returned by the recognizer, accuracy increases to 88%. The size of the returned word

<sup>3</sup>An error tolerance of, say, 0.3 means that any activation value of an output unit below  $-0.7$  is considered to be a  $-1.0$  and any value above  $+0.7$  is considered to be a  $+1.0$  (i.e., no error is feed back).

hypotheses set ranged between 1 and 8 words with an average of 1.6 words. There were in total 1,232 letters in our testing strings and the corresponding letter level recognition rate was 98.5%. This is an indication, which we manually confirmed, that when the top choice word was incorrect the recognizer had missed only a few characters (at most one in our testing set).

## 5 Summary and Conclusions

We have presented a system for writer independent large vocabulary recognition of on-line handwritten cursive words. The system is composed of two modules: a filtering module, based on features that are computationally very easy to compute, which quickly reduces a large reference dictionary to a much smaller number of string candidates, and a recognition module which uses a temporal representation of the input word, instead of a static 2-dimensional image of the word, which allows us to preserve the sequential nature of the data and enables us to use a TDNN-style network. The network recognizer avoids explicit segmentation of the input words by using a sliding window concept. Both modules were tested on a number of images and have been shown to be useful in this domain.

## Acknowledgements

The authors wish to thank several people at CEDAR. Dar-Shyang Lee for his many suggestions and assistance. Stayvis Ng implemented most of the lexicon reduction routines. Bobby Kleinberg implemented the slant correction algorithm. This work was supported in part by the USPS Office of Advanced Technology and by the National Science Foundation under research grant IRI9315006.

## References

- [1] E. Brocklehurst and P. Kenward. *Preprocessing for cursive script recognition*. NPL Report DITC 132/88, 1988.
- [2] D. Elliott. A better activation function for artificial neural networks. Tech. Report TR93-8, Institute for Systems Research, University of Maryland, 1993.
- [3] S. Fahlman. An empirical study of learning speed in back-propagation networks. Tech. Report CMU-CS-DD-88-162, Computer Science Department, Carnegie Mellon University, 1988.
- [4] D. Ford. *On-line recognition of connected handwriting*. PhD thesis, University of Nottingham, 1991.
- [5] T. Fujisaki, H. Beigi, C. Tappert, M. Ukelson, and C. Wolf. Online recognition of unconstrained handprinting: a stroke-based system and its evaluation. In *From Pixels to Features III*. Elsevier Science Publishers, 1992.
- [6] W. Guerfali and R. Plamondon. Normalizing and restoring on-line handwriting. *Pattern Recognition*, 26(3):419–431, 1993.
- [7] I. Guyon, P. Albrecht, Y. LeCun, J. Denker, and W. Hubbard. Design of a neural network character recognizer for a touch terminal. *Pattern Recognition*, 24(2):105–119, 1991.
- [8] N. Hakim, J. Kaufman, G. Cerf, and H. Meadows. Cursive script online character recognition with a recurrent neural network model. In *IJCNN*. IEEE, 1992.
- [9] J. Hollerbach. An oscillation theory of handwriting. *Biological Cybernetics*, 39:139–156, 1981.
- [10] D. Knuth. *The Art of Computer Programming: Sorting and Searching*, vol 3. Addison Wesley, 1973.
- [11] Y. LeCun. Generalization and network design strategies. In *Connectionism in Perspective*. Elsevier Science Publishers, 1989.
- [12] G. Martin and J. Pittman. Recognizing hand-printed letters and digits using backpropagation learning. *Neural Computation*, 3:258–267, 1991.
- [13] 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.
- [14] D. Rumelhart, G. Hinton, and R. Williams. *Learning internal representations by error propagation*, vol 1, p. 318–362. Bradford Books, 1986.
- [15] M. Schenkel, H. Weissman, I. Guyon, C. Nohl, and D. Henderson. Recognition-based segmentation of on-line hand-printed words. In *NIPS V*. Morgan Kaufmann, 1993.
- [16] 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.
- [17] Y. Singer and N. Tishby. A discrete dynamical approach to cursive handwriting analysis. Tech. Report CS93-4, Institute of Computer Science, The Hebrew University of Jerusalem, 1993.
- [18] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang. Phoneme recognition using time-delay neural networks. *IEEE Trans. on ASSP*, 37:328–339, 1989.