Recognition-based Segmentation of On-line Run-on Handprinted Words: Input vs. Output Segmentation.

M. Schenkel*, H. Weissman, I. Guyon, C. Nohl, D. Henderson
AT&T Bell Laboratories, Holmdel, NJ 07733
* ETH-Zürich, CH-8092 Zürich

Abstract

This paper reports on performance of two methods for recognition-based segmentation of strings of on-line handprinted capital Latin characters. The input strings consist of a time-ordered sequence of X-Y coordinates, punctuated by pen-lifts. The methods were designed to work in "run-on mode" where there is no constraint on the spacing between characters. While both methods use a neural network recognition engine and a graph-algorithmic post-processor, their approaches to segmentation are quite different. The first method, which we call *INSEG* (for input segmentation), uses a combination of heuristics to identify particular pen-lifts as tentative segmentation points. The second method, which we call *OUTSEG* (for output segmentation), relies on the empirically trained recognition engine for both recognizing characters and identifying relevant segmentation points. Our best results are obtained with the *INSEG* method: 11% error on handprinted words from an 80,000 word dictionary.

1 INTRODUCTION

Our research is motivated by the desire to improve man-machine communications so that computers become accessible to an ever wider range of users. If replacing mouse and keyboard by "electronic paper" and "electronic pencil" is not to remain a dream, significant advances need to be made in handwriting recognition. To maximize our chances of success, we address in this paper a task of real practical interest but of intermediate difficulty: the writer independent recognition of handprinted words from an 80,000-word English dictionary.

Several levels of difficulty in the recognition of printed words are illustrated in the

samples of figure ??, extracted from our databases (table ??). Except in the cases of boxed or clearly spaced characters, trying to segment characters independently of the recognition process yields poor recognition performance. This motivates us to explore recognition-based segmentation techniques.

uppercase database	data nature	pad used	training set size	test set size	approx. # of donnors
DB1	boxed letters	AT&T	9000	1500	250
DB2	short words	Grid	8000	1000	400
DB3	English words	Wacom	-	600	25

Table 1: **Databases used for training and testing.** *DB2* contains words one to five letters long, but only four and five letter words are constrained to be legal English words. *DB3* contains legal English words of any length. English words are from an 80,000 word dictionary.

The basic principle of recognition-based segmentation is to present to the recognizer many "tentative characters". The recognition scores ultimately determine the string segmentation. We have investigated two different ways of achieving this. The first method uses tentative characters delineated by heuristic segmentation points. It is expected to be most appropriate for handprinted capital letters since nearly all writers separate these letters by pen-lifts. The second method expects the recognition engine to learn empirically (learn by examples) both to recognize characters and to identify relevant segmentation points.

2 METHODOLOGY

2.1 PREPROCESSOR

The data collection device provides pen trajectory information as a sequence of (x, y) coordinates at regular time intervals (10-15 ms). Our preprocessing preserves this pen trajectory information (figure ??). In contrast with other preprocessings that encode entire parts of characters (or allographs) [?, ?, ?], our preprocessing results in a finely sampled sequence of feature vectors providing local information along the pen trajectory. Features include: pen coordinates, pen lifts, speed, direction and curvature [?]. Pen up segments are linearly interpolated. We obtain a crude and redundant representation from which the neural network recognizer can extract higher order features.

To introduce invariance with respect to scale and fluctuations in pen speed, pen coordinates are normalized and resampled to points regularly spaced in arc length. We use two variants of our normalization process:

1. **Individual character normalization.** The character is put into a box of normal height. The resampling step Δs is a fraction of the total arc length L: $\Delta s = L/n_1$. Resampled characters have a fixed number n_1 of points per character. This preprocessing can only be used when tentative isolated characters are processed, as in the INSEG method.

2. **Entire word normalization.** Words are deslanted and put into a box of normal height. The origin is vertically centered but horizontally moving slowly from left to right, following a local average of the x-coordinates. The resampling step Δs is a fraction of the word height H: $\Delta s = H/n_2$. Resampled characters have a fixed number n_2 of points per unit arc length and therefore a variable number of points per character. This preprocessing is necessary when entire words are processed without trying to first isolate characters, as in the OUTSEG method.

2.2 NEURAL NETWORK RECOGNIZER

As a recognizer we chose a Time Delay Nerual Network ($TDNN$) as described in [?].

The $TDNN$ (figure ??) is a multi-layer feed-forward network, the layers of which perform successively higher-level feature extraction and produce scores for each class which can be interpreted as probabilities of presence of a given character in the input field. There is one output per class, in this case 26 outputs, providing a scores for all the letters of the Latin alphabet. $TDNN$'s, which were previously applied to speech recognition, are well suited to sequential signal processing [?, ?].

A neuron has an input field restricted in time. Weight sharing is used in time direction to enforce time invariant recognition features. The neuron weights are trained by example with the standard error-back-propagation algorithm.

2.3 POSTPROCESSOR

The critical step in the segmentation process is the postprocessing which disentangles various word hypotheses using the character recognition scores provided by the $TDNN$. For this purpose, we use conventional graph algorithms [?].

2.3.1 Word Recognition without Lexicon

A word interpretation is associated to a path in a graph whose nodes contain character recognition scores. In figure ??, the recognition scores for the word "LOOP" are gathered into an interpretation graph. Possible transitions between tentative character interpretations are indicated by the arrows. The bold-faced line shows the best path.

We use the following notations:

- $X\{i\}$: node corresponding to character interpretation X ($X \in \{A, B, C, ..., Z\}$), for tentative character number i ($i \in \{1, 2, ..., m\}$).
- $P(X\{i\}|input)$: score associated to node $X\{i\}$, the probability that tentative character number i is interpreted as character X given the input pattern (Scores are normalized between 0 and 1, if necessary, to behave as proper probabilities).
- $nil\{i\}$: node corresponding to no character interpretation (meaningless character):
  $P(nil\{i\}|input) = 1 - (P(A\{i\}|input) + P(B\{i\}|input) + ... + P(Z\{i\}|input))$
- $P(Y\{j\}|X\{i\})$: transition probability from node $X\{i\}$ to node $Y\{j\}$.

Transitions enforce temporal or geometrical constraints: all transitions go forward along the temporal direction to preserve letter ordering. Their weights indicate transition probabilities $P(Y\{j\}|X\{i\})$. In this paper transition weights are not trained, they are set by design (see sections ?? and ??).

The probability of a given path, is approximated by the product of all its node and arrow values. The best path (which corresponds to the most probable segmentation-recognition combination) is picked up with the help of the Viterbi algorithm.

Figure ?? also illustrates how the *nil* node can be used if forward jumping connections are not allowed to skip meaningless tentative characters (dashed line). In section ??, we authorize forward jumping connections. In section ??, we use the *nil* node to identify meaningless characters.

2.3.2 Word Recognition with Lexicon

There are many ambiguities inherent to our writing system which does not provide a fully deciphrable code when proper spacing is not respected. Consider the case of letter "K" in the word "CLOCK" of figure ?? (d). Without context it could be read as the two letters "I" and "C". Our knowledge of the English lexicon is necessary in this case to remove the ambiguity.

We use the k-best path algorithm [?] to provide a few likely word hypothesis. The *ispell* program [?] then checks these hypothesis against an 80,000 English word dictionary and returns a list with legal words closest to the presented one. The best path, subject to one of these interpretations, is chosen using a constrained Viterbi algorithm. Using this method not only do we improve performance but we also use our recognizer as a spell checker.

2.4 TRAINING DRIVEN SEGMENTATION

The description of our system so far assumes three independent modules: preprocessor, $TDNN$ and postprocessor. The only trainable module, the $TDNN$, could be trained with isolated characters, independently of segmentation issues.

Our initial recognition performances obtained with $TDNN$'s trained on isolated characters from $DB1$ were not satisfactory. Many meaningless characters received good scores for arbitrary interpretations. Moreover, embedded characters caused

many mistakes. For example, the vertical bar was recognized with too high confidence as an "I", resulting in many wrong interpretations, including "R"'s or "K"'s, read as "IZ" and "IC".

These problems arrive from not training the network on the same task as it is supposed to solve in the end. It has been trained only on correctly segmented isolated characters but is asked to classify examples from a different set (e.g. meaningless and embedded characters which should receive the *nil* interpretation). This suggests that training cannot be performed independently of segmentation.

To perform the "training driven segmentation" sessions, we generate tentative characters from the unsegmented word database $DB2$. These tentative characters come unlabeled, the only information available in the database being entire word labels. To avoid hiring an operator for the tedious labeling task, many approaches have been suggested in the literature [?, ?, ?, ?]. Two of them are used in this paper:

1. **Labeling with the postprocessor.** In section ??, the best path through the interpretation graph, subject to the correct word interpretation, is obtained with the constrained Viterbi algorithm. Tentative characters along that path are given their corresponding best interpretation. The other ones, which do not lie on the path, are given the *nil* interpretation. This procedure assumes that there is already a recognizer which can provide scores for the interpretation graph. The recognizer is first trained with presegmented data (for instance isolated characters from $DB1$). It is then used to label tentative characters from an unsegmented word database ($DB2$). The recognizer is retrained with the newly produced examples. These two last steps can be iterated many times until convergence.
2. **Position invariant learning.** In section ??, the labeling problem is circumvented. During training, we use an additional layer in the neural network structure which transforms the location specific information on where a character appears into a simple presence indicator [?]. This allows us to train the network with one target vector containing only information about presence or absence of a character anywhere in the input field. Due to the convolutional nature of the $TDNN$ and the redundancy of the data, the corresponding positions are enforced.

In the following sections, we enter into the specifics of both segmentation methods under study.

3 SEGMENTING IN INPUT SPACE

3.1 TENTATIVE CHARACTERS AND INTERPRETATION GRAPH

In figure ?? we represent the different steps of the $INSEG$ process. Module number 1 is used to define "tentative characters" delimited by "tentative cuts" (spaces or pen-lifts). The tentative characters are then handed to module number 2 which performs the preprocessing and the scoring of the characters with a $TDNN$. The recognition results are then gathered into an interpretation graph. In module number 3 the best path through that graph is searched with an optimization algorithm.

In figure ?? we show a simplified representation of an interpretation graph built by our system. We have shown only nodes corresponding to the highest score interpretation and no arrows are drawn.

Each tentative character has a double index: the tentative cut i at the character starting point and the tentative cut j at the character end point. With our previous notations, $X\{i,j\}$ is the node associated to the score of letter X for the tentative character $\{i,j\}$. A path through the graph starts at a node $X\{0,.\}$ and ends at a node $Y\{.,m\}$, where 0 is the word starting point and m the last pen-lift. In between, only transitions of the kind $X\{.,i\} \rightarrow Y\{i,.\}$ are allowed, to avoid character overlapping. The transition probability is simply set to $1/fout$, where $fout$ is the fan-out of node $X\{.,i\}$.

To select tentative cuts, the simplest strategy is to retain all pen-lifts. Tentative characters are thus obtained by combining the strokes delineated by these pen-lifts. Usually combinations of one to four strokes include all possible legal characters present in the word to be recognized (and many more!).

To avoid searching through too complex a graph, we need to perform some pruning. The spatial relationship between strokes is used to discard unlikely tentative cuts. For instance, strokes with a large horizontal overlap are bundled. The remaining cuts define a set of "tentative characters" which are then grouped in different ways to form more tentative characters. Tentative characters separated by a large horizontal spatial interval are never considered for grouping.

We checked the robustness of our pruning technique and found that less than 3% of recognition errors can be attributed to improper choices of tentative cuts. Besides reducing the recognition time by more than a factor of 10, graph pruning cut down the error rate by a factor of two.

3.2 NETWORK ARCHITECTURES AND TRAINING

The network specifications are the same as in [?]. with 4 convolutional layers and 6252 weights. It uses a fixed dimension input and the individual-character normalisation as a preprocessing.

The network is initially trained with the isolated characters of the $DB1$ training set. A first iteration of "training driven segmentation" is then performed, using the bootstrap method described in section ??. Tentative characters from the $DB2$ training set are thus labeled to obtain 20,000 new valid-examples and 2000 *nil*-examples that are merged with the $DB1$ training set to retrain the network. *Nil*-examples are trained with target zero for all classes. The results reported in table ?? show clear performance improvement with the training driven segmentation.

We found that retraining with *nil*-examples did not degrade the recognition rate of valid isolated characters.

The input space segmentation method does not presume a particular type of recognizer. For comparison, we substituted two other neural network recognizers to the $TDNN$. These networks use alternative input representations. The $OCR-net$ was designed for Optical Character Recognition [?] and uses pixel map inputs. Its first layer performs local line orientation detection. The $orientation-net$ has

an architecture similar to that of the $OCR-net$, but its first layer is removed and local line orientation information, directly extracted from the pen trajectory, is transmitted to the second layer [?]. Without dictionary, the $OCR-net$ has an error rate more than twice that of the $TDNN$ but the $orientation-net$ performs similarly. With dictionary the $orientation-net$ is better than the $TDNN$: the error rate per character is 6.5% on the $DB2$ test set and the word error rate 11%. This improvement is attributed to better second and third best recognition choices, which facilitate the lexical checking.

	Method	
Trained on	$INSEG$	$OUTSEG$
$DB1$	19 (14)	55 (59)
$+\ DB2$	9 (8.5)	10 (8)

Table 2: **Performance improvement due to training driven segmentation.** The error rate is given on the test set of $DB2$ on a per character bases as measured by the Levenshtein distance using the $TDNN$ architectures. The values in parantesis are obtained by use of the dictionary.

4 SEGMENTING IN OUTPUT SPACE

4.1 TENTATIVE CHARACTERS AND INTERPRETATION GRAPH

In contrast with the $INSEG$ system, the $OUTSEG$ method does not rely on human designed segmentation hints: the neural network learns both recognition and segmentation features from examples.

The production of tentative characters is trivial: A window is swept in small steps over the input sequence. At each step the content of the window is taken to be a tentative character. Successive characters usually overlap considerably.

In figure ??, we show the outputs of our $TDNN$ recognizer when the word "LOOP" is processed. The main matrix is a simplified representation of our interpretation graph where no arrow is represented. Tentative character numbers i ($i \in \{1, 2, ..., m\}$), run along the time direction. Each column contains the scores of all possible interpretations X ($X \in \{A, B, C, ..., Z, nil\}$) of a given tentative character. The bottom line is the nil interpretation score which complements the sum of all other scores in the same column (see section ??).

The connections between nodes reflect a modelization of character durations. A simple way of enforcing duration is to allow only the following transitions:

$$\begin{aligned} X\{i\} &\rightarrow X\{i+1\}, \\ nil\{i\} &\rightarrow nil\{i+1\}, \\ X\{i\} &\rightarrow nil\{i+1\}, \\ nil\{i\} &\rightarrow X\{i+1\}, \end{aligned}$$

where X stands for a certain letter. A character interpretation can be followed by the same interpretation but cannot be followed immediately by another character interpretation: they must be separated by nil. This permits distinguishing between letter duration and letter repetition (such as the double "O" in our example). The best path (see figure ??) in the graph is found by the Viterbi algorithm.

In fact, this simple pattern of connections corresponds to a Makov model of duration, with exponential decay. We implemented a slightly fancier model allowing to map any duration distribution (figure ??). In our experiments, we selected a Poisson duration distribution with parameter $\lambda = 2$. We used a four state model ended by a self loop. This truncates the distribution and replaces its tail by an exponential tail.

As a consequence, there are many more nodes in our graph than figure ?? indicates. Each node $X\{i\}$ is replicated as many times as there are states in the duration model of the corresponding letter X. The transition probabilities within the model, $P_{\alpha\beta}, P_{\beta\gamma}, etc.$, are determined by the duration distribution.

4.2 NETWORK ARCHITECTURE AND TRAINING

As a network we use a $TDNN$ with 3 layers and 10817 weights. The sequence of recognition scores is obtained by sweeping the neural network over the input. Because of the convolutional structure of the $TDNN$, there are many identical computations between two successive calls of the recognizer. Therefore, the $TDNN$ is computationally very efficient for this task. Only about one sixth of the network connections are reevaluated for each new tentative character. As a consequence, although the $OUTSEG$ system processes many more tentative characters than the $INSEG$ system does, the overall computation time is about the same.

The network is trained with the position invariant training method explained in section ??. The presence indicator of character X is computed as $X\{\} = 1 - \prod_i (1 - X\{i\})$, which approximates the probability that character X appears at least once in the input field. We minimize the cross-entropy cost function between outputs $X\{\}$ and target values [?].

To prove the efficiency of the position invariant training, we tested first a network trained only with isolated characters from $DB1$ (table ??) and compared it to a network using position invariant training with $DB2$ words. As shown in table ?? the position invariant training method improves very significantly the results over just training on isolated characters.

5 COMPARISON RESULTS AND CONCLUSIONS

The paper presented a methodological synthesis and the analysis of various experiments. As it is very hard to make a fair comparison, our conclusions should not be extrapolated beyond their limits. Both methods work and, while the paper was in preparation, their performance improved and varied with respect to one another. We tried here to oppose the two methods, but we think of them as two complementary ways of addressing the segmentation problem and our final system will possibly incorporate features of both.

	Error without lexicon		*Error with lexicon*	
on $DB2$	% char.	% word	% char.	% word
$INSEG$	9	18	8.5	15
$OUTSEG$	10	21	8	17
on $DB3$	% char.	% word	% char.	% word
$INSEG$	8	33	5	13
$OUTSEG$	11	48	7	21

Table 3: **Performance on short handprinted words ($DB2$ test set), with our best $TDNN$ recognizers.**

To complement the results obtained with database $DB2$ we tested (without retraining) with database $DB3$ containing words of any length from the English dictionary (table **??**). In our current versions, $INSEG$ performs better than $OUTSEG$. Although there is not definite evidence that one system is superior to the other, another advantage favors the $INSEG$ method: it can easily be used with other recognizers than the $TDNN$, whereas the $OUTSEG$ method relies heavily on the convolutional structure of the $TDNN$ for computational efficiency. Our best results so far (tables **??**) were obtained with the $INSEG$ method, by combining with a voting scheme two recognizers: the $TDNN$ and the $orientation-net$ (see section **??**). We mention for comparison the results obtained by a commercial recognizer on the same data. One should notice that our lexicon is the same as the one from which the data was drawn and is probably a larger lexicon than the one used by the commercial system. Our results are substantially better than those of the commercial system. On an absolute scale they are quite satisfactory if we take into account that the test data was not cleaned at all and that more than 20% of the errors have been identified to be patterns written in cursive, misspelled or totally illegible.

We expect the $OUTSEG$ method to work best for cursive handwriting, which does not exhibit trivial segmentation hints, but we do not have direct evidence. In reference [**?**] the authors had success with a version of $OUTSEG$. Work is in progress to extend the capabilities of our systems to cursive.

	Error without lexicon		*Error with lexicon*	
Method	% char.	% word	% char.	% word
$DB2$	7 (18)	13 (29)	7 (17*)	10 (32*)
$DB3$	6 (20)	23 (61)	5 (18*)	11 (49*)

Table 4: **Performance of our best system: $INSEG$ with a combination $TDNN+orientation-net$.** For comparison, we mention in parenthesis the performances obtained by a commercial recognizer on the same data. The performance of the commercial system with lexicon (marqued with a *) are penalized because $DB2$ and $DB3$ include words not contained in their lexicon.

Acknowledgments

We wish to thank all the Neural Network group at Bell Labs Holmdel for supportive discussions. We are grateful to Anne Weisbuch, Yann Le Cun and Jan Ben for giving us their Neural Networks to try on our INSEG method. We are indebted to Howard Page for providing comparison figures with the commercial recognizer.