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

H. Weissman, M. Schenkel*, I. Guyon, C. Nohl, D. Henderson

AT&T Bell Laboratories, Holmdel, NJ 07733

* also 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.

Keywords: Character Recognition, On-line Character Recognition, Neural Networks, Time-Delay Neural Networks, Segmentation, Run-on handwriting.

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 1, extracted from our databases (table 1). Except in the cases of boxed or clearly spaced characters, trying

uppercase database	data nature	AT&T location	pad used	resolution (pts/inch)	sampling (pts/sec.)	training set size	test set size	approx. num. of donnors
DB1	boxed letters	Columbus, OH	AT&T	70	80	9000	1500	250
DB2	short words	Holmdel, NJ	Grid	75	60	8000	1000	400
DB3	English words	Holmdel, NJ	Wacom	500	200	-	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.

to segment characters independently of the recognition process yields poor recognition performance. This motivates us to explore recognition-based segmentation techniques.

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.

The two methods are schematically illustrated in figure 2:

- **INSEG.** We call our first method *INSEG* (for input segmentation). It uses spatial information to identify particular pen-lifts as tentative segmentation points or "tentative cuts". In figure 2 (a) we indicate by a feedback loop that "definite cuts" are selected among the "tentative cuts" in the recognizer *input space*, after examination of the recognition results of various "tentative characters". This approach, introduced in [1] is the basis for the technique described in section 3.
- **OUTSEG.** We call our second method *OUTSEG* (for output segmentation). The empirical recognizer is passed continuously over the input stream, generating a stream of probability estimates for particular characters being present in the recognizer's receptive field (figure 2 (b)). The segmentation *per se* is carried out in the recognizer *output space*. Similar techniques were applied to Optical Character Recognition [2, 3] and to cursive handwriting recognition [4]. This method is described in further detail in section 4.

We begin by discussing the characteristics common to both methods.

2 Methodology

In this section, we describe the common features of our two segmentation methods with respect to preprocessing, recognition and postprocessing. The recognition step is performed by a neural network,

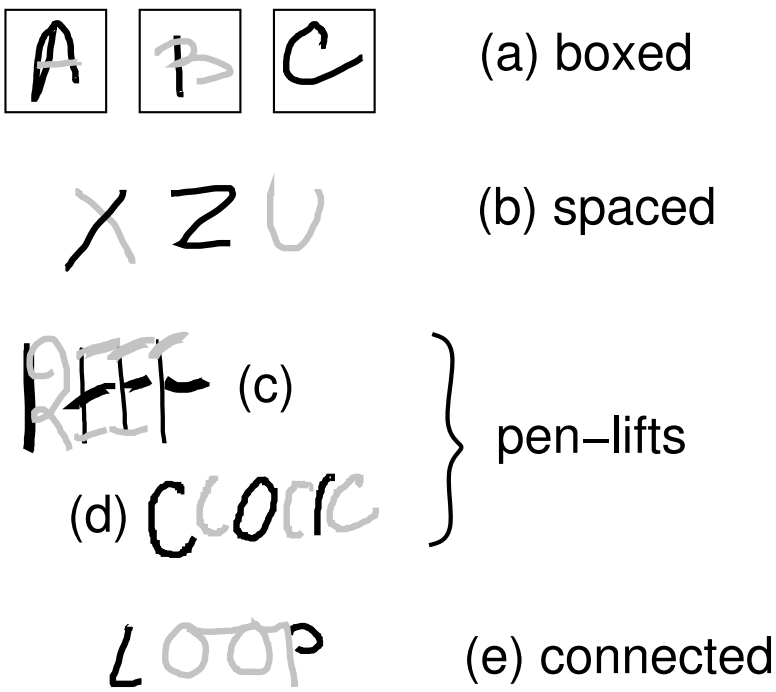

Figure 1: **Examples of styles that can be found in our databases:**
(a) $DB1$; (b) $DB2$; (c), (d), (e) $DB2$ and $DB3$. The line thickness or darkness is alternated at each pen-lifts.

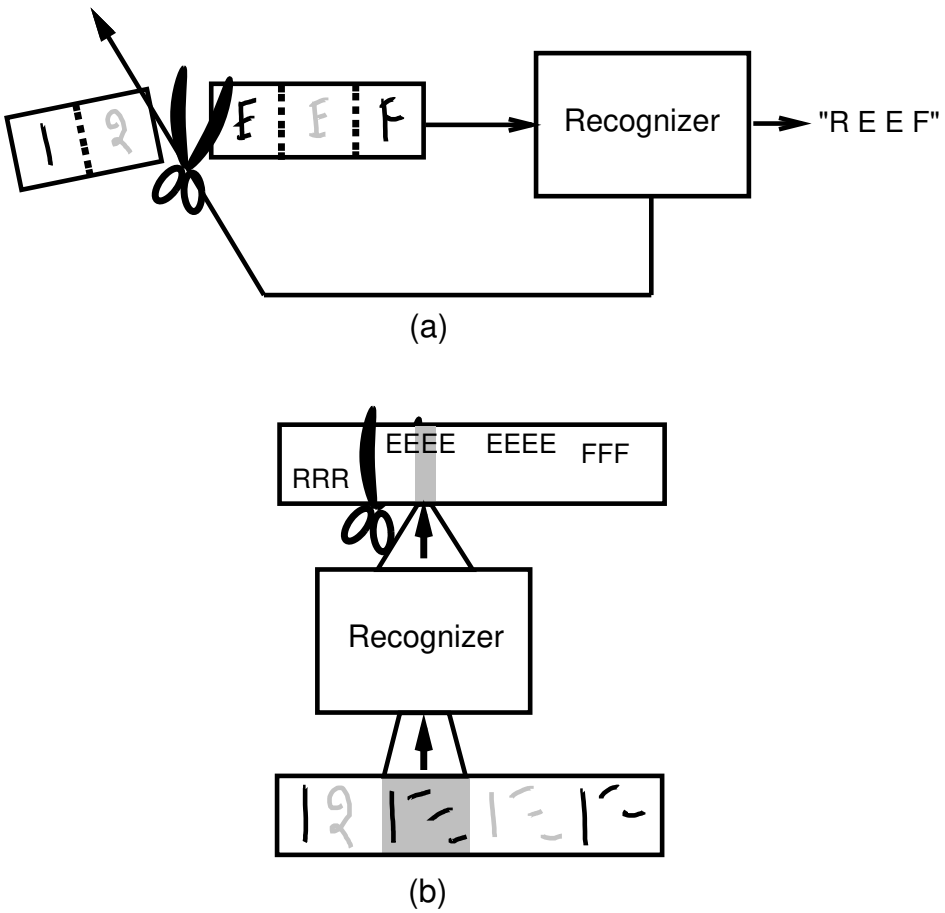

Figure 2: **Schematic illustration of the two segmentation techniques.**
(a) Input space segmentation ($INSEG$). (b) Output space segmentation ($OUTSEG$).

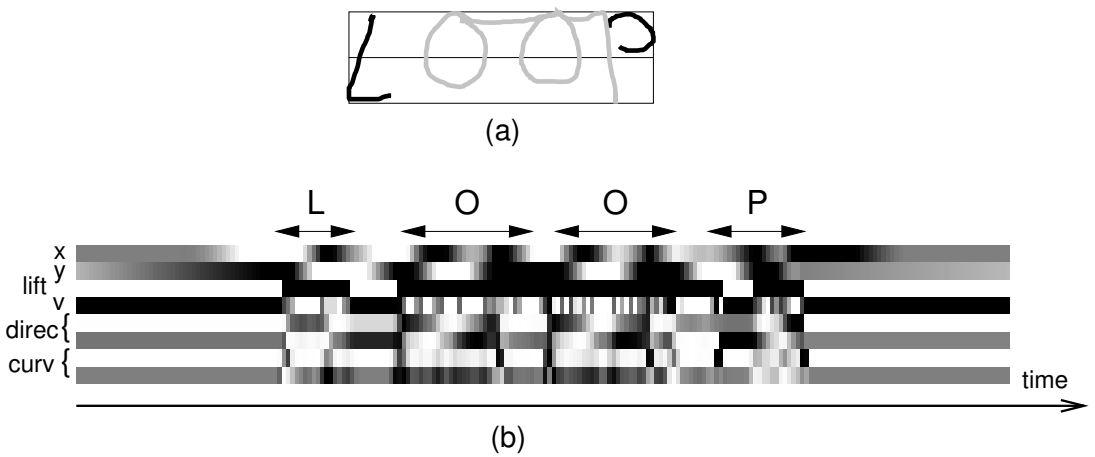

Figure 3: **Preprocessing.**
(a) The original word. After each pen lift the color is switched between black and grey to show the individual strokes.
(b) The data as presented to the network.

trained by examples, and the rest of the process is handled by conventional techniques. We introduce the notion of training driven segmentation.

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 3). In contrast with other preprocessings that encode entire parts of characters (or allographs) [5, 6, 4], 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 [7]. 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 (figure 4), 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.** Characters are put into a box of normal height. The origin is at the center of the box. 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, providing a fixed size input to the neural network.

2. **Entire word normalization.** Words are deslanted and put into a box of normal height. The origin

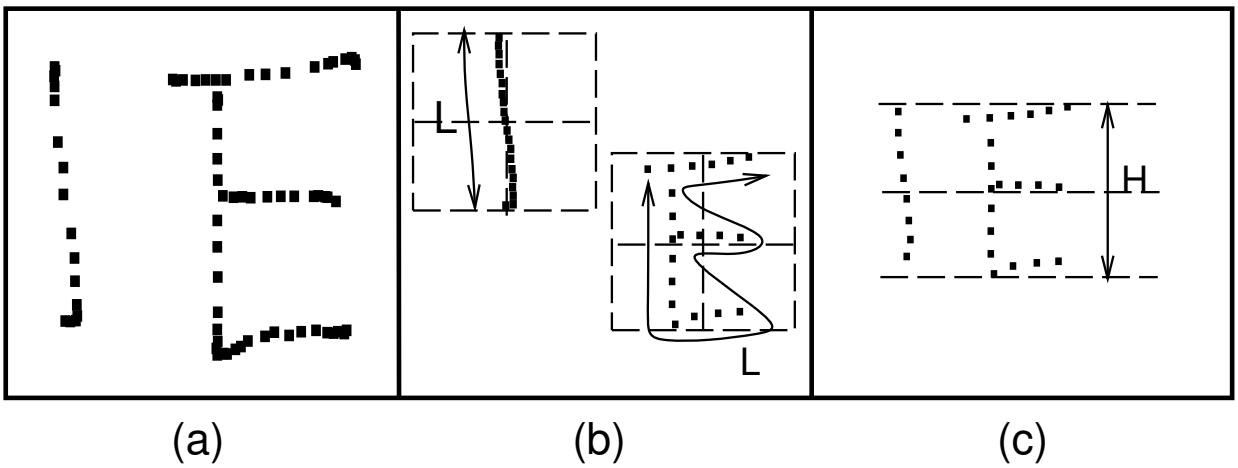

Figure 4: **Normalization and resampling.**
(a) The original data: The points are sampled at a constant sampling rate. Areas of high point density correspond to low speed.
(b) Individual character normalization. The number of points per character is constant.
(c) Entire word normalization. The sampling step is constant.

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. The relative character proportions are respected.

The *entire word normalization* is designed for recognizers that process entire words at a time such as $OUTSEG$ (section 4). The *individual character normalization* is adequate for the $INSEG$ method (section 3) which processes isolated "tentative characters" already segmented in input space.

2.2 Neural network recognizer

Segmentation and recognition are performed with estimates of character interpretation posterior probabilities. Such estimates can be obtained with various statistical techniques. The considerations developed in this section justify the choice of a neural network, the Time Delay Neural Network ($TDNN$) for the task at hand.

The $TDNN$ (figure 5) 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. $TDNN$'s, which were previously applied to speech recognition, are well suited to sequential signal processing [8, 9]. This allows us to use a representation which preserves the sequential nature of the data, in contrast with other approaches based on pixel-map

representation.

Each unit (a "neuron") performs a weighted sum of its inputs followed by a non-linear squashing function (a hyperbolic tangent). A neuron has an input field restricted in time. It behaves like a matched filter which extracts a local topological feature by scanning the output of the previous layer along the time dimension. In short, each layer of the network consists of three operations: convolution with a set of kernels (each kernel representing the weights of one neuron), squashing and subsampling. By repeating these operations, we extract progressively more complex features, sensitive to progressively wider portions of the input field. The output layer has the ultimate global features: the character interpretation scores. There is one output per class, in this case 26 outputs, providing a scores for all the letters of the Latin alphabet.

The neuron weights are determined by training. $TDNN$'s are trained by example with gradient descent techniques such as the back-propagation algorithm [10]. Because neurons have restricted input fields and are reused in time, there are far fewer weights (i. e. free parameters) in this architecture than in fully connected networks. This has two desirable consequences: overfitting is prevented and less memory is needed to store the network.

Internal feature representations of the network are tailored by thousands of examples from a large number of writers. This greatly facilitates adaptation to a particular user's style. Better writer dependent recognition accuracies can rapidly be obtained by retraining only the last layer of the network with a few examples [11].

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 [12]. To introduce the common notations used in sections 3 and 4, we produce here a schematic example.

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 6, 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

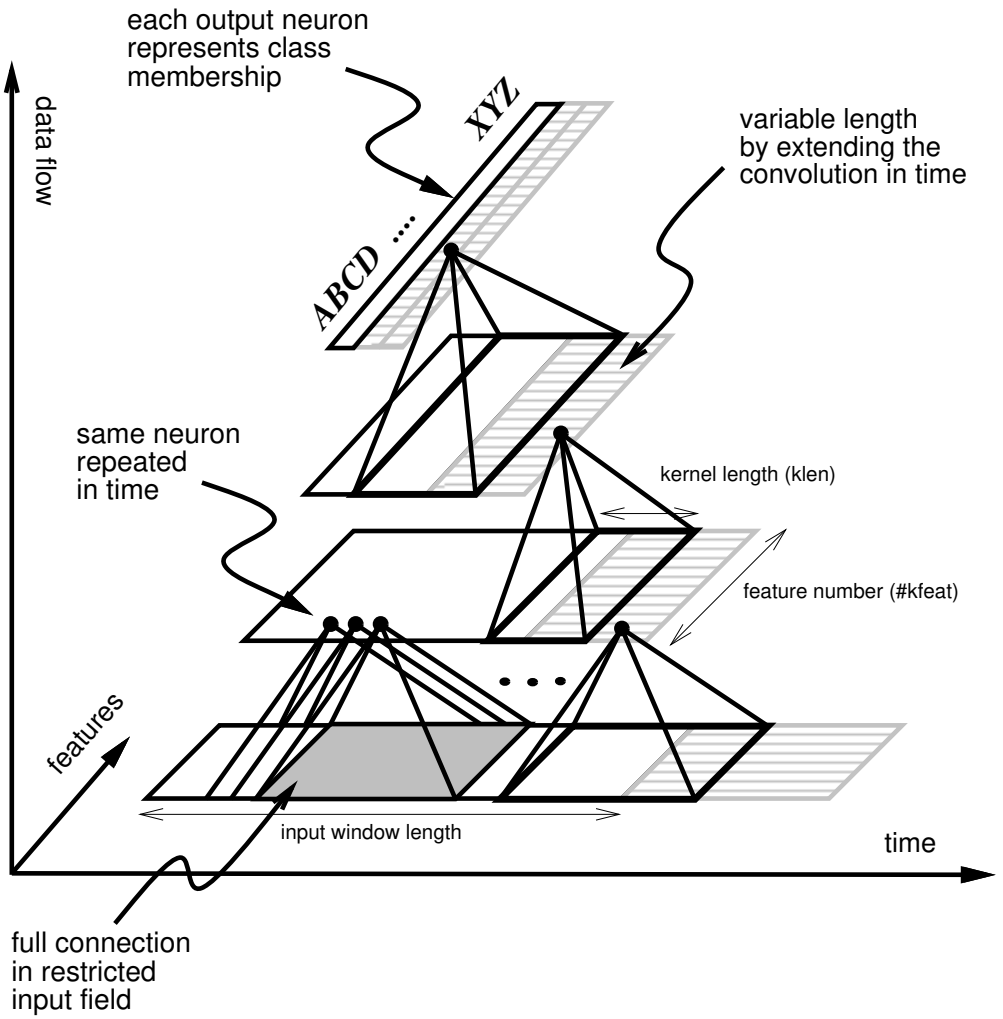

Figure 5: **Architecture of the Time Delay Neural Network ($TDNN$).**

The connections between layers obey the following rules (not all neurons are represented):

- neurons are feature detectors with restricted input fields, limited in the time direction,

- in each layer, a set of neurons scans the output of the previous layer along the time axis, every Δn time step (Δn being the subsampling rate), and produces higher-level feature vectors.

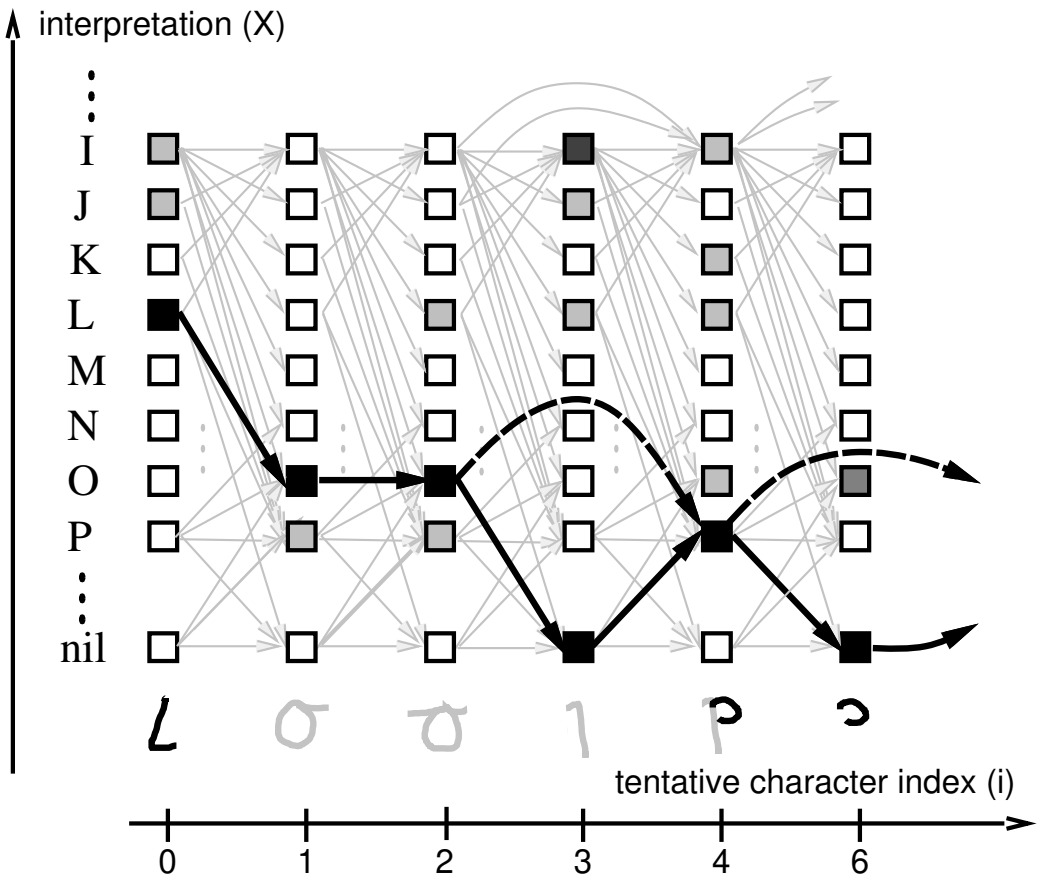

Figure 6: **Schematic example of interpretation graph.**
Each node is associated with an interpretation score, $P(X\{i\}|input)$. Each arrow represents a transition probability between two interpretations, $P(Y\{j\}|X\{i\})$. The best path through the graph (bold arrows) gives the final word interpretation.

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; many forward transitions are pruned to prevent, for instance, character overlapping. 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 3 and 4).

There are as many possible word interpretations as paths in the graph, starting from a node of the fisrt column and ending on a terminal node (not represented on the figure). The probability of a given path, is approximated by the product of all its node and arrow values. For instance, for the word "LOOP" in figure 6, the score of the best path (bold-faced line) approximates:

$$P(L\{0\}|input) \cdot P(O\{1\}|L\{0\}) \cdot P(O\{1\}|input) \cdot P(O\{2\}|O\{1\}) \cdot P(O\{2\}|input) \cdot P(P\{4\}|O\{2\}) \cdot P(P\{4\}|input).$$

A simple decoding scheme consists in picking up the word interpretation with highest score. This is achieved by searching for the best path with the Viterbi algorithm [13, 14, 15].

Figure 6 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 3, we authorize forward jumping connections. In section 4, we use the *nil* node to identify meaningless characters.

2.3.2 Word recognition with lexicon

Without using a dictionary (or lexicon), can we solve one of the main difficulty in segmentation: the problem of embedded characters?

The vertical bar of the "P", in our example of figure 6, may have been interpreted as letter "I" with a very high score, but as the second part of the "P" does not stand by itself for any Latin character. Therefore, thanks to the temporal and geometrical constraints that are implemented in the graph, this hypothesis is discarded and the path including the right interpretation "P" retained. The letters "E" in the word "REEF" of figure 1 (c) is an even more complicated example of embedded characters ($I \subset F \subset E$) which can be resolved similarly.

But many ambiguities are 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 1 (d). Without context it could be read as the two letters "I" and "C". Our knowledge of the English lexicon is here necessary to remove the ambiguity.

We use the k-best path algorithm [12] to provide a few likely word hypothesis. The *ispell* program [16] then checks these hypothesis against an 80,000 English word dictionary. It produces new hypothetical

interpretations, corresponding to legal dictionary words, and differing from the initial hypothesis at most by an insertion, a substitution, a deletion or a transposition of two adjacent characters. The best path, subject to one of these interpretations, is chosen using the Viterbi algorithm [13, 14, 15].

Using this method not only do we improve performance but we also use our recognizer as a spell checker. The efficiency of *ispell* could be improved for our specific use. Candidates proposed by the program are based on frequencies of spelling mistakes obtained by typing on a keyboard. These should be replaced by frequencies of mistakes made by the $TDNN$.

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. We explain in this section why this simplistic approach bears some limitations, and sketch the principles of training driven segmentation further developed in section 3 and 4.

Several conditions must be fulfilled for our methods to work properly:

- Tentative characters corresponding to actual characters should obtain a large score for the right interpretation and small scores for the other ones.
- Meaningless tentative characters should result in a small score for all character interpretations (or, the largest score for the *nil* interpretation).
- The right interpretation should at least be in the few first best scored interpretations.

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 are significantly reduced when the recognizer is trained not only with valid characters but also with "counter-examples". Counter-examples include meaningless characters which should receive the *nil* interpretation, but also valid characters such as "I" which induce errors in the overall recognition of the word when their score is too high. Valid-examples and counter-examples should therefore preferably correspond to tentative characters which the system failed to interpret properly. 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 [17, 18, 19, 20]. Two of them are used in this paper:

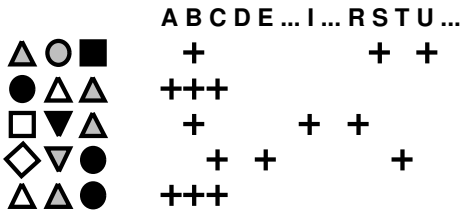

Figure 7: **Position invariant learning.**
Try this quiz to convince yourself that the $TDNN$ can be trained by being only told which characters appear in the input field, not where they appear. Symbols on the left side represent one letter of the alphabet. On the right side, you are told which letter appears in the corresponding line. Can you guess which symbol stands for B?

1. **Labeling with the postprocessor.** In section 3, 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 4, 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 [3]. 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. This idea seems *a priori* poorly founded. After all, we are throwing away useful character order information available from the word labels. In practice, position guessing is enforced by the architecture of the $TDNN$ and the redundancy of the data. One can get a feeling for how this semi-supervised learning works by solving the quiz of figure 7.[1]

In the following sections, we enter into the specifics of both segmentation methods under study, the description of their interpretation graphs, their neural network architecture and training methods.

[1] Answer to the quiz: The grey triangle pointing up stands for "B".

3 Segmenting in input space

In this section, we present the $INSEG$ recognition system which relies on segmentation information provided by spaces and pen lifts. We will first define tentative characters and interpretation graph, then explain the specifics of the neural network recognizer and the training procedure.

3.1 Tentative characters and interpretation graph

In figure 8 we represent the different steps of the $INSEG$ process. This block diagram can be thought of as an unfolded version of figure 2 (a). 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. The final segmentation retains a subset of the tentative cuts referred to as "definite cuts".

In figure 9 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!). But the graph obtained with such a simple scheme is unnecessarily big, which results in slow search and poor recognition rate. This last point comes from two reasons: Almost all single strokes stand by themselves as perfectly valid characters. Among all three and four stroke tentative characters, very few are valid characters.

To avoid searching through too complex a graph, we need to perform some pruning. Pen-lifts are first used to define a set of "tentative cuts" which delimit the "strokes". 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 that 3% of recognition errors

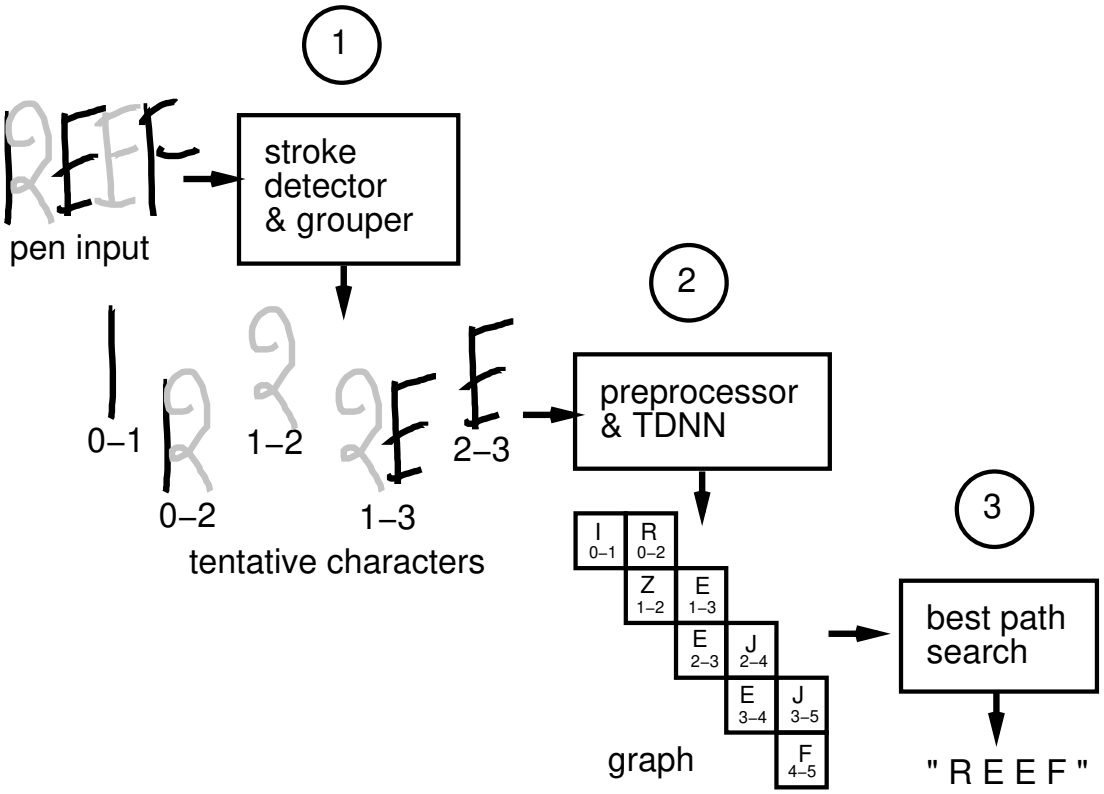

Figure 8: **Segmentation in input space.** (1) Pen-lifts and spaces are used as segmentation possibilities, or "tentative cuts", to construct "tentative characters". The two numbers separated by a dash indicate the order numbers of the "tentative cuts" that were retained to delineate the "tentative characters". (2) These characters, preprocessed individually and processed by the $TDNN$, obtain recognition scores. (3) The best path in the graph of recognition scores is searched, with or without the help of a dictionary.

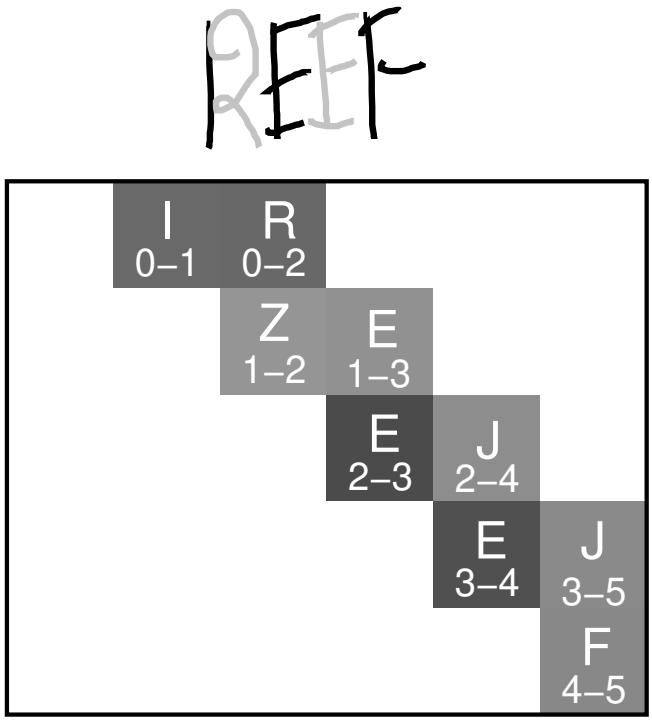

Figure 9: **Graph obtained with the input segmentation method for the word REEF.** The grey shading indicates the recognition scores (the darker, the stronger the recognition score and the higher the recognition confidence). For this example, the correct recognition is obtained without the help of a dictionary.

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.

We illustrate in figure 10 the incorporation of lexical checking. Without the help of dictionary lookup, the word "REEF" was found by the Viterbi algorithm in the graph of figure 9. In the other example of figure 10, however, the word "CCOCIC' was found instead of "CLOCK". We represent in this figure the second and third best character interpretations used by the k-best path algorithm. These were used to produce more word hypothesis that were checked against the English dictionary, as explained in section 2.3. The correct interpretation could thus be found.

3.2 Network architectures and training

We give in this section the specifications of the $TDNN$ recognizer and the training modalities. We mention comparison results obtained with the same segmentation technique, but with other neural network classifiers.

The network specifications are shown in table 2. The $TDNN$ has a fixed dimension input and a single output vector delivering the 26 scores for the letters of the alphabet. To provide a fixed number of feature

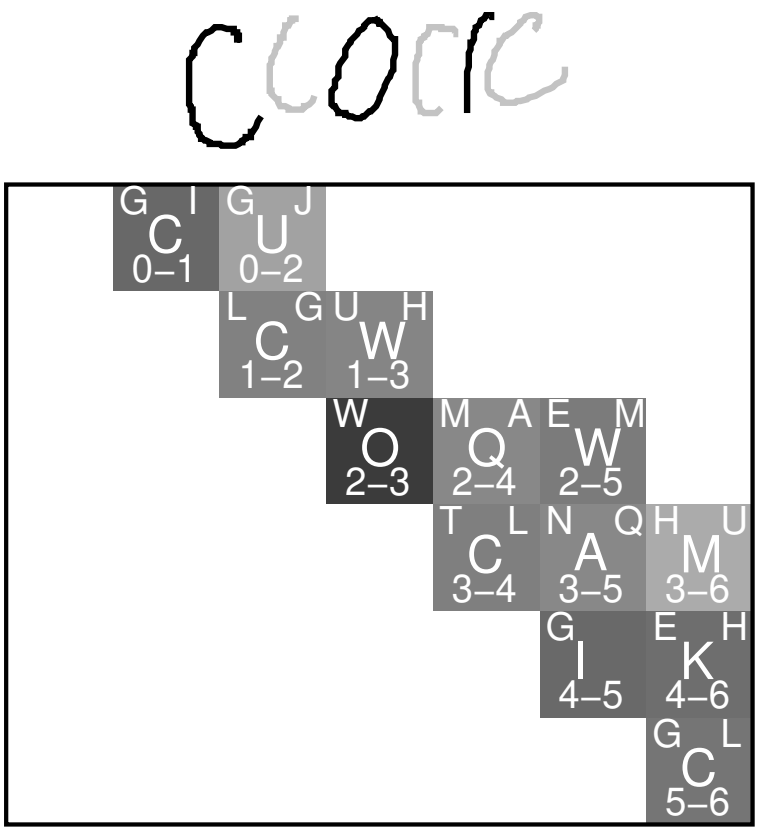

Figure 10: **Graph obtained with the input segmentation method for the word CLOCK.** Second and third highest scores are shown in the upper corners. The word "CCOCIC" is obtained without lexical checking, but the correct recognition "CLOCK" is obtained with the help of a dictionary.

input window length	subsampl. steps Δn	klen / #kfeat 1^{st} layer	klen / #kfeat 2^{nd} layer	klen / #kfeat 3^{nd} layer	klen / #kfeat 4^{nd} layer	#indep. weights
90	3.3.2	8 / 7	6 / 10	4 / 16	5 / 24	6,252

Table 2: $TDNN$ **specifications for the** $INSEG$ **system.** Kernels have dimensions klen by #kfeat (see figure 5).

	without lexicon		*with lexicon*	
Method	% char. error	% word error	% char. error	% word error
Training with $DB1$	19	29	14	23
Retraining with $DB2$	9	18	8.5	15

Table 3: **Performance of the** $INSEG$ **system on short handprinted words (**$DB2$ **test set), using a** $TDNN$ **recognizer.**
The error rate per character includes insertions substitutions and deletions, as measured by the Levenshtein distance between target word and recognized word [21]. $DB1$ contains only isolated characters and $DB2$ short words.

vectors to the $TDNN$ input, we use individual character normalization in the preprocessing described in section 2. Only seven features are used, the speed being omitted. For more detailed justifications of this particular choice of $TDNN$ architecture, see reference [7].

The sum of the squared errors between $TDNN$ outputs and target values is minimized during training. The target is 1 for the correct interpretation and 0 for all other classes. 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 2.4. Tentative characters from the $DB2$ training set are thus labeled to obtain 20,000 new valid-examples and 2000 counter-examples that are merged with the $DB1$ training set to retrain the network. Counter-examples are trained with target zero for all classes. The results reported in table 3 show clear performance improvement with the training driven segmentation.

The performance improvements were due to a reduction in the scores of both meaningless tentative characters and tentative characters that may be embedded in another character such as the vertical bar. We found that retraining with counter-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 [22] and uses pixel

map inputs. Its first layer performs local line orientation detection. The $orientation - net$ has an architecture similar 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 [23]. 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.

4 Segmenting in output space

In this section, we present the $OUTSEG$ recognition system. In contrast with the $INSEG$ system, it does not rely on human designed segmentation hints: the neural network learns both recognition and segmentation features by examples.

4.1 Tentative characters and interpretation graph

In the output space segmentation method, the production of tentative characters is trivial. A window is swept continuously over the input sequence. The consecutive tentative characters can be seen through the window at regular sampling intervals (figure 2). They usually overlap considerably.

In figure 11, 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 2.3.1).

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

$$X\{i\} \rightarrow X\{i+1\}, nil\{i\} \rightarrow nil\{i+1\}, X\{i\} \rightarrow nil\{i+1\}, nil\{i\} \rightarrow X\{i+1\},$$

where X stands for any 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 11) in the graph is found by the Viterbi algorithm [13, 14, 15].

In fact, this simple pattern of connections corresponds to a Makov model of duration, with exponential decay (figure 12). We implemented a slightly fancier model allowing to map any duration distribution (figure 13). In our experiments, we selected a Poisson duration distribution with parameter $\lambda = 2$. We

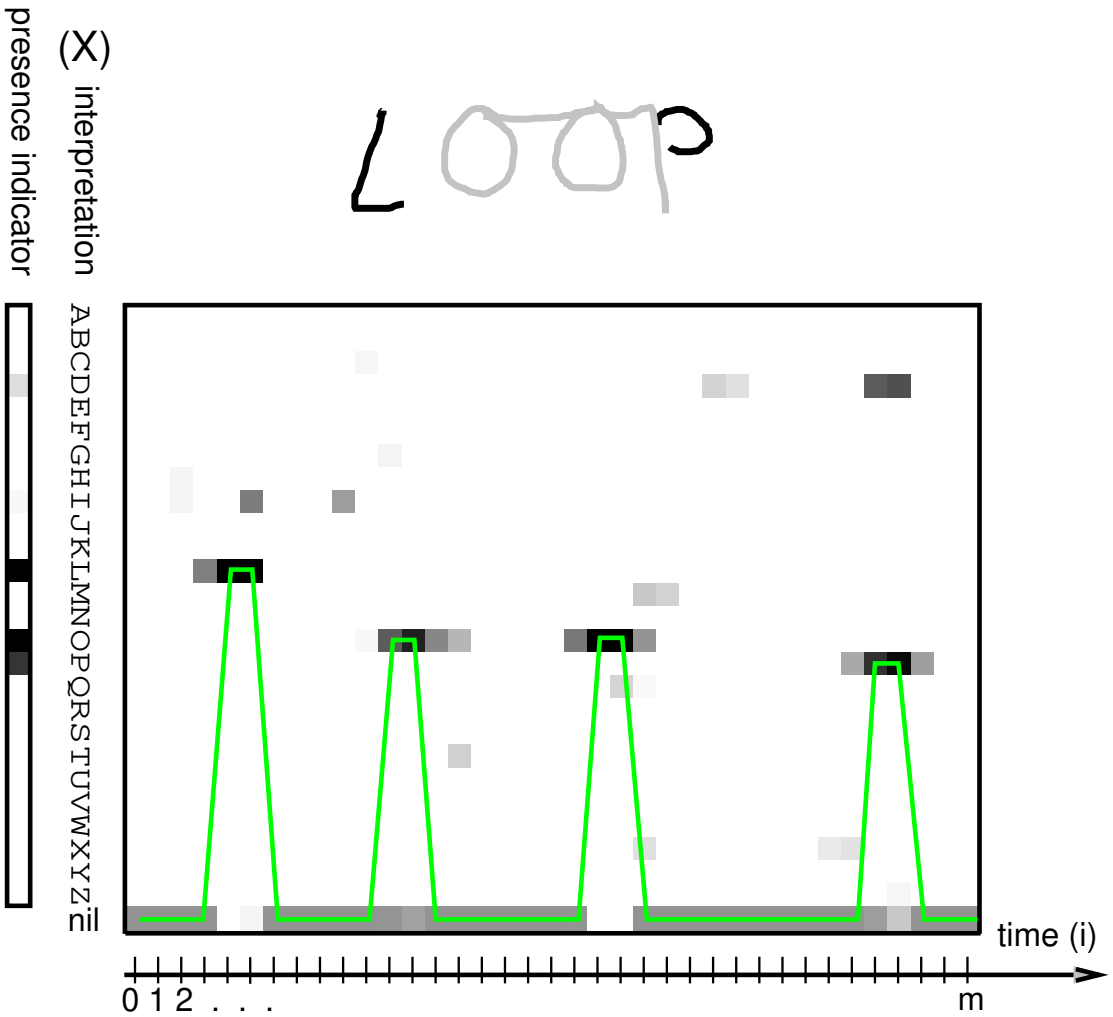

Figure 11: *TDNN* **outputs of the** *OUTSEG* **system.**

The vertical column on the left side is the probability of the presence of a given character anywhere in the input field. The main matrix represents the sequence of outputs of the *TDNN*. The grey curve indicates the best path through the graph, using duration modeling. The word "LOOP" was correctly recognized in spite of the ligatures which prevent segmentation on the basis of pen-lifts.

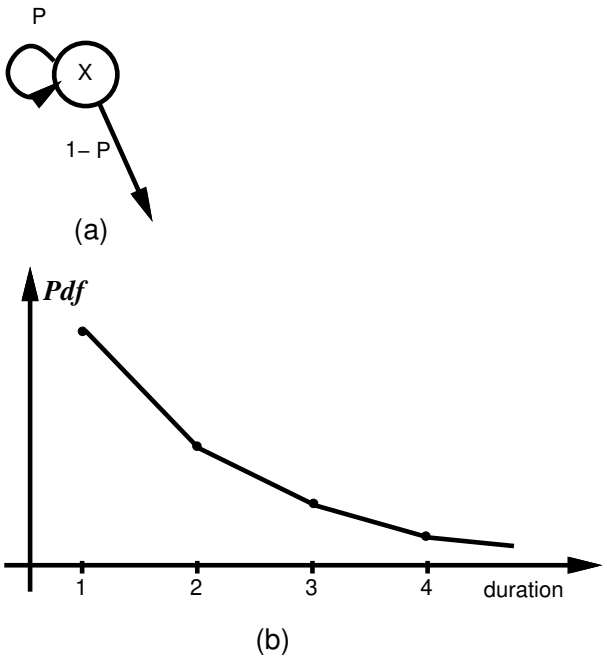

Figure 12: **Duration model with exponential decay.**

(a) Markov model. (b) Duration probability density function.

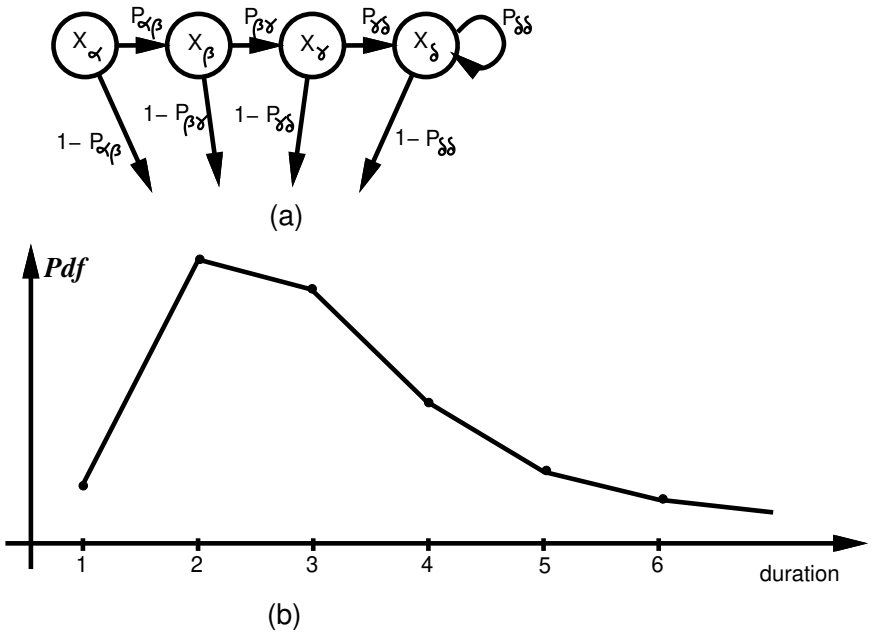

Figure 13: **Duration model.**

(a) Markov model. (b) Duration probability density function. The exponential tail results from the self loop of the last unit.

input window length	total subsampling	klen / #kfeat 1^{st} layer	klen / #kfeat 2^{nd} layer	klen / #kfeat 3^{nd} layer	#indep. weights	$DB2$ test error
62	$2.2.1 = 4$	12 / 8	12 / 15	8 / 24	10,817	10

Table 4: **Best $TDNN$ specifications for the $OUTSEG$ system.** Kernels have dimensions klen by #kfeat (see figure 5). The error rate on $DB2$ is per character and without dictionary.

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 11 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. The transitions between models (with probabilities (1 - $P_{\alpha\beta}$), (1 - $P_{\beta\gamma}$), etc.) enforce alternating valid characters and nil.

4.2 Network architecture and training

In this section, we justify the specifications of our best $TDNN$, designed for output space segmentation. We report the results obtained with position invariant learning (see section 2.4).

We explored semi-randomly the very large space of possible network architectures, following our intuition and the guidelines provided in reference [7]. We then performed a more systematic exploration around the architecture of our best network (table 4). The comparison is made on the error rate per character, which includes insertions substitutions and deletions, as measured by the Levenshtein distance between target word and recognized word [21]. The following results are obtained by training and testing with the $DB2$ data:

- **Input window length.** Enlarging it or reducing it by 30% results in a 2 to 3% error rate increase.
- **Total subsampling.** Doubling it or dividing by two results in a 2% error rate increase.
- **Number of layers.** Going from three to four layers results in a 1.5 to 3.5% error increase.
- **Number of independent weights.** Increasing the number of features in each layer so as to get 30% more independent weights results in a 1% error increase, while reducing it by the same amount results in a 3% error increase.

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 third of the network connections are reevaluated for each new tentative character. As a

	without lexicon		*with lexicon*	
Method	% char. error	% word error	% char. error	% word error
Training with $DB1$	55	82	59	84
Training with $DB2$	10	21	8	17

Table 5: **Performance of the $OUTSEG$ system on short handprinted words ($DB2$ test set).** $DB1$ contains only isolated characters and $DB2$ short words. The lexical checking involved only the shortest path (as opposed to the k-shortest path in the previous section).

	without lexicon		*with lexicon*	
Method	% char. error	% word error	% char. error	% word error
$INSEG$	9	18	8.5	15
$OUTSEG$	10	21	8	17

Table 6: **Performance on short handprinted words ($DB2$ test set), with our best $TDNN$ recognizers.**

consequence, although the $OUTSEG$ system processes about three times as many tentative characters as 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 2.4. During training, we add a layer which transforms the outputs of the $TDNN$ into a single score vector indicating presence or absence of a given character anywhere in the input field. The presence indicator of character 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 [3]. The target is 1 for the correct interpretation and 0 otherwise.

To prove the efficiency of the position invariant training, we tested first a network trained only with isolated characters from $DB1$ (table 5). The same network was retrained using position invariant training with $DB2$ words. We also trained a network directly with $DB2$ examples. We found that the network trained directly with $DB2$ performs best. We report also this last result in table 5. The table shows that the position invariant training method, which allows to train the network on entire words to recognize both characters and transitions between characters, improves very significantly the results over just training on isolated characters.

	without lexicon		*with lexicon*	
Method	% char. error	% word error	% char. error	% word error
$INSEG$	8	33	5	13
$OUTSEG$	11	48	7	21

Table 7: **Performance on handprinted English words ($DB3$ test set), with our best $TDNN$ recognizers.**

	without lexicon		*with lexicon*	
Method	% char. error	% word error	% char. error	% word error
$DB2$	7	13	7	10
$DB3$	7	26	4	11

Table 8: **Performance of or best system: $INSEG$ with a combination $TDNN + orientation - net$.**

5 Comparison results and conclusions

In this paper we proposed two solutions for the problem of on-line character recognition of run-on handprinted characters. In both systems, the recognizer scores are used to determine the string segmentation and the core recognition engine is a Time Delay Neural Network ($TDNN$). Our first system (segmentation in input space or $INSEG$) relies on writers to separate their letters with spaces or pen-lifts. This permits using heuristics to select only a few likely tentative characters. Our second system (segmentation in output space or $OUTSEG$) does not make any segmentation decision prior to recognition. Many finely sampled tentative characters are processed by the $TDNN$, which must empirically learn both to recognize characters and to identify character transitions.

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.

To complement the results obtained with database $DB2$ (table 6) we tested (without retraining) with database $DB3$ containing words of any length from the English dictionary (table 7). 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 8) were obtained with the $INSEG$ method, by combining with a voting scheme two recognizers: the $TDNN$ and the $orientation - net$ (see section 3.2).

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 [4] the authors had success with a version of $OUTSEG$. Work is in progress to extend the capabilities of our systems to cursive.

Acknowledgments

We wish to thank all the Neural Network group at Bell Labs Holmdel for supportive discussions and in particular Bernhard Boser, Chris Burges, Yann Le Cun, John Denker and Larry Jackel for many useful suggestions. We are grateful to Anne Weisbuch, Yann Le Cun and Jan Ben for giving us their Neural Networks to try on our input space segmentation system for comparison. This work would not have been possible without the help of Ed Pednault and Doug Riecken with the data collection device and software.

References

[1] C. J. C. Burges, O. Matan, Y. Le Cun, D. Denker, L. D. Jackel, C. E. Stenard, C. R. Nohl, and J. I. Ben. Shortest path segmentation: A method for training neural networks to recognize character strings. In *IJCNN'92*, volume 3, Baltimore, 1992. IEEE.

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

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

[4] D. Rumelhart and et al. Integrated segmentation and recognition of cursive handwriting. In *Third NEC symposium Computational Learning and Cognition*, Princeton, New jersey, 1992 (to appear).

[5] H. L. Teuling and L. R. B. Schomaker. Unsupervised learning of prototype allographs in cursive-script recognition using invariant handwriting features. In S. Impedovo, editor, *From pixels to features III*, Amsterdam, 1992. Elsevier.

[6] K. A. Sizov. Recognition of symbols and words written by hand. In *Proceedings of the International Conference on Document Analysis and Recogntion*, volume 2, Saint-Malo, France, 1991. IAPR, IEEE, IEE.

[7] I. Guyon, P. Albrecht, Y. Le Cun, J. Denker, and W. Hubbard. Design of a neural network character recognizer for a touch terminal. *Pattern Recognition*, 24(2), 1991.

[8] K. J. Lang and G. E. Hinton. A time delay neural network architecture for speech recognition. Technical Report CMU-cs-88-152, Carnegie-Mellon University, Pittsburgh PA, 1988.

[9] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37:328–339, March 1989.

[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel distributed processing: Explorations in the microstructure of cognition*, volume I, pages 318–362. Bradford Books, Cambridge, MA, 1986.

[11] I. Guyon, D. Henderson, P. Albrecht, Y. Le Cun, and J. Denker. Writer independent and writer adaptive neural network for on-line character recognition. In S. Impedovo, editor, *From pixels to features III*, Amsterdam, 1992. Elsevier.

[12] M. Gondran and M. Minoux. *Graphs and algorithms*. Wiley, New York, 1984.

[13] Jr. Forney, G. D. The Viterbi algorithm. In *Proceedings of the IEEE*, volume 61-3, March 1973.

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

[15] H. Sakore and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. In A. Waibel and K. F Lee, editors, *Readings in Speech Recognition*, page 159. Morgan-Kaufmann, 1990.

[16] R. E. Gorin et al. $UNIX^{TM}$ man-page for ispell, version 3.0.06 (beta). 09/17/91.

[17] L. Y. Bottou, F. Fogelman Soulié, P. Blanchet, and J. S. Liénard. Speaker independent isolated digit recognition: Multilayer perceptrons vs. dynamic time warping. *Neural Networks*, 3-4, 1990.

[18] N. Morgan and H. Bourlard. Continuous speech recognition using multilayer perceptrons with hidden markov models. In *Proceedings ICASSP-90*, Albuquerque, 1990. IEEE.

[19] Y. Bengio, R. De Mori, G. Flammia, and R. Kompe. Global optimization of a Neural Network-Hidden Markov Model hibrid. *IEEE Transactions on Neural Networks*, 3-2:252–259, 1992.

[20] P. Haffner. Connectionist word-level classification in speech recognition. In *Proceedings ICASSP-92*, San Francisco, 1992. IEEE.

[21] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Cybernetics and control theory*, 10(8):845–848, 1965.

[22] Y. Le Cun, L.D. Jackel, B. Boser, J.S. Denker, H.P. Graf, I. Guyon, D. Henderson, R.E. Howard, and W. Hubbard. Handwritten digit recognition: Application of neural network chips and automatic learning. *IEEE Communications Magazine*, pages 41–46, November 1989.

[23] A. Weissbuch and Y. Le Cun. Private communication. 1992.