Handbook on Optical Character Recognition and Document Image Analysis, pp. 000–000
Eds. P. S. P. Wang and H. Bunke
© World Scientific Publishing Company, 1996

CHAPTER 8

OVERVIEW AND SYNTHESIS OF ON-LINE CURSIVE HANDWRITING RECOGNITION TECHNIQUES

ISABELLE GUYON
AT&T Bell Laboratories
955 Creston Road, Berkeley, CA 94708, USA
isabelle@research.att.com

MARKUS SCHENKEL
Swiss Federal Institute of Technology (ETH)
CH-8092 Zürich, Switzerland
schenkel@isi.ee.ethz.ch

and

JOHN DENKER
AT&T Bell Laboratories
Holmdel, NJ 07733, USA
jsd@research.att.com

In this paper, we analyze several on-line cursive handwriting recognition systems. We find that virtually all such systems involve (a) a preprocessor, (b) a trainable classifier, and (c) a language modeling post-processor. Such architectures are described within the framework of Weighted Finite State Transductions, previously used in speech recognition by Pereira et al. We describe in some detail a recognition system built in our laboratory. It is a writer independent system which can handle a variety of writing styles including cursive script and handprint. The input to the system encodes the pen trajectory as a time-ordered sequence of feature vectors. A Time Delay Neural Network is used to estimate *a posteriori* probabilities for characters in a word. A Hidden Markov Model segments the word in a way which optimizes the global word score, taking a lexicon into account. The last part of the paper is devoted to bibliographical notes.

Keywords: Cursive; Handwriting; Recognition; On-line; Character recognition; Hidden Markov Models; Finite state automata; Finite state transductions; Neural networks; WFSA; WFST; HMM; Graph; Viterbi; TDNN; grammar; UNIPEN; segmentation; Product graph; INSEG; OUTSEG; script; handprint.

1. Introduction

"On-line" handwriting recognition refers to the case where we know the pen coordinates as a function of time and not only a static image of words written on paper. This task has become increasingly important over the past few years. Until the beginning of the nineties, on-line handwriting recognition research was mainly academic. The situation has changed with the rapid growth of the pen computing

industry, and in particular with the appearance on the market of the first personal communicators.

The mediocre quality of handwriting recognition has been a major obstacle to the success of pen computers. Users report that it is "too inaccurate, too slow, and too demanding of user attention". The entire pen computing industry will turn its back on handwriting and revert to *popup keyboards* unless major advances in handwriting recognition are made.

The most challenging version of this problem is writer-independent mixed-style handwriting recognition for large vocabularies:

- *Writer independent* means that the system can be used directly by any writer, without requiring an adaptation or training session. Potential users when they purchase a system expect high "walk up" recognition accuracy and it is therefore very important to guarantee a high degree of writer independence. The adaptation of the machine to a particular writer style can then be used to increase the accuracy of recognition.
- *Mixed style* means that the system can handle both handprint and cursive or even a mixture of both within the same word. This covers all natural writing styles and does not place on the writer any constraints (such as writing in boxes, combs, etc.).
- *Large vocabulary* means a vocabulary containing all the common words (25,000 words at least). Large vocabularies are required to be able to use handwriting for text entry applications.

In this paper, we describe and analyze several techniques for on-line cursive and mixed style recognition for the Latin alphabet which permit reaching high levels of writer independence and using large vocabularies.

2. Common Features of Most Systems

In this section, we describe a fairly general framework for the problem of on-line cursive and mixed style handwriting recognition; Sec. 3 will consider the pros and cons of various designs for some of the building blocks. Traditionally, on-line recognizers consist of a preprocessor, a classifier (which estimates probabilities for the different categories of characters or other subword units) and a language-modeling postprocessor (often a Hidden Markov Model). The system usually has adjustable parameters whose values are determined during a training session. Postprocessors incorporating language models are of special importance; without them the error rate would be unacceptable (typically around 30% on a per-character basis).

2.1. *System Architecture*

The overall architecture of a *generic* cursive recognition system can be found in Fig. 1.

The **input** to on-line cursive systems is generally a sequence of regularly time spaced pen coordinates: $X = [[x_1, y_1, z_1][x_2, y_2, z_2]...[x_t, y_t, z_t]...]$ where t is the time:

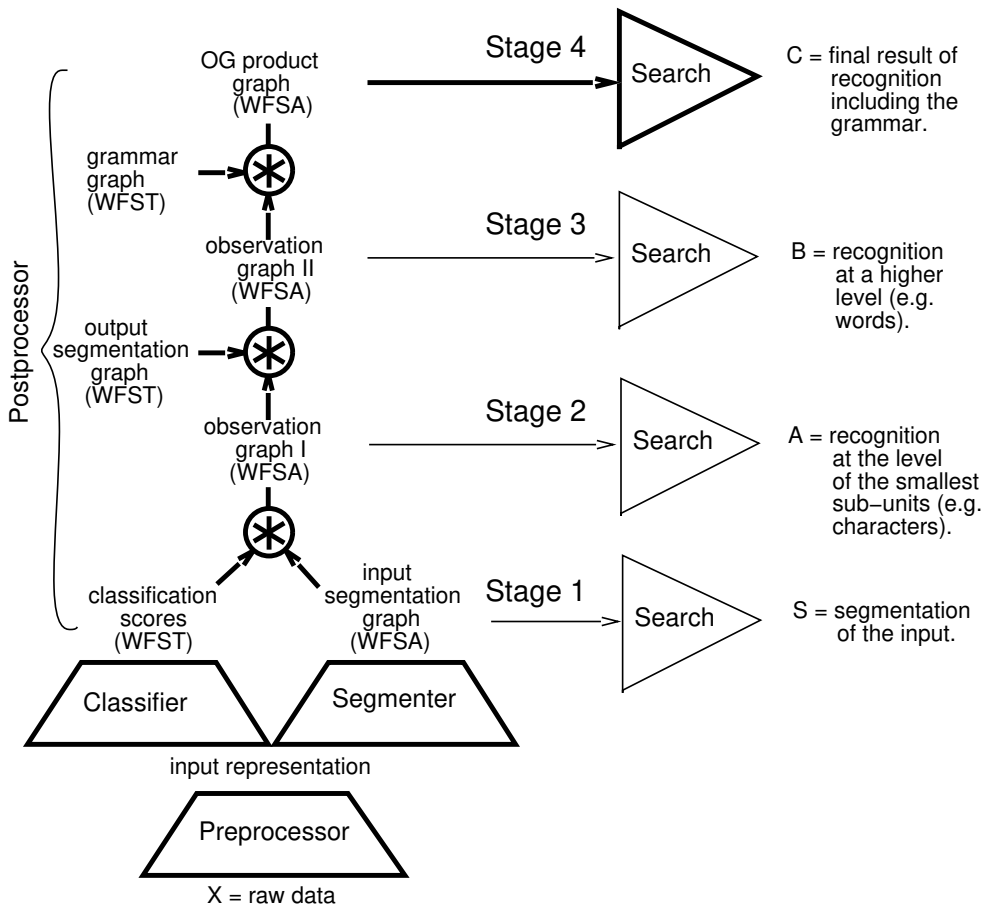

Fig. 1. **Overview of a generic cursive recognition system.** Most cursive recognition systems use graph algorithms to determine simultaneously the best recognition and segmentation. The overall process can be decomposed into intermediate steps. The lower level graphs are usually not searched. If they would be searched, the results would be an intermediate recognition result, less refined than the last level. For instance, the input segmentation graph would provide a low level segmentation (S) which does not take the classification scores or grammar into account. There are two main approaches to segmentation: *in*put *seg*mentation (*inseg*) and *out*put *seg*mentation (*outseg*). An *inseg* system (Fig. 7) uses an elaborate input segmentation graph build from segmentation points heuristically determined but has no output segmentation graph. An *outseg* system (Fig. 8) has a trivial input segmentation graph and an elaborate output segmentation graph. Graphs are composed with the transduction product method (see text). WFSA stands for Weighted Finite State Automaton and WFST for Weighted Finite State Transducer (see text).

$t = 1, ..., T$. A typical sampling rate is 100 points per second. The z coordinate tells whether the pen is "up" or "down" (i.e. touching the writing surface or not); more detailed pressure information is usually too unreliable to be useful.

The **preprocessor** accepts the raw input data X and improves its signal quality, enforces invariances, and detects relevant features. The output of the preprocessor is the "recognizer input representation". Many representations have been proposed, putting more or less emphasis on the temporal information (see Secs. 3.2 and 3.4). No consensus on representation has emerged so far.

The **segmenter** splits the recognizer input representation into "segments" that may or may not overlap. For a character-based recognizer, for instance, each segment corresponds to a "tentative character".[a] A simple example of segmentation is shown in Fig. 2.[b] There are usually many more segments than characters in the input. This is why the method is sometimes called "over-segmentation". Whether a segment coincides with a handwritten character is determined later with the help of the scores of the classifier.

For each segment, the **classifier** produces scores for each class associated with the characters of a chosen alphabet.[c] Scores typically represent either probabilities or negative-log-probabilities. A variety of classifiers have been proposed in the literature [16].[d] No one technique has been outstandingly successful, although many of the best classifiers are based on neural network techniques.

As explained in more detail in the next section, the **postprocessor** uses a graph algorithm to determine the most probable segmentation and recognition, taking into account a language model. Most language models used for cursive handwriting recognition are implementable by Weighted Finite State Automata or WFSA. In Fig. 3, we show two examples of language models implementable by WFSAs that are commonly used: A lexicon trie (or trie) and simple Markov model or n-gram. Lexicon trees are used to store lists of words with their frequency. Character n-grams are used to predict the next character given a window $(n-1)$ of past characters. Word n-grams are used to predict the next word given a window of $(n-1)$ past words. Grammars implementable by WFSAs are called "regular". They offer speed and accuracy advantages over other more complex grammars [35]. In the long run, as computing power and algorithms improve, many hopes are pinned on improving recognition by modeling the syntax and semantics of the language with more complex grammars.

2.2. *Graph-based Segmentation and Recognition*

Most cursive recognition systems formulate segmentation and recognition as a

[a] For some recognizers, segments may correspond to sub-characters, sub-words or words. For simplicity, we discuss the "character" case; the same line of reasoning applies to other cases.
[b] More examples of cursive word segmentation are shown in Fig. 13.
[c] In some approaches that classify sub-characters shapes (e.g. strokes) instead of characters, the classification stage may be trivial (e.g. a simple clustering). The classification at the character level takes place at a later stage, but the basic architecture is similar.
[d] See also the chapter by U. Kressel and J. Shürmann in this book.

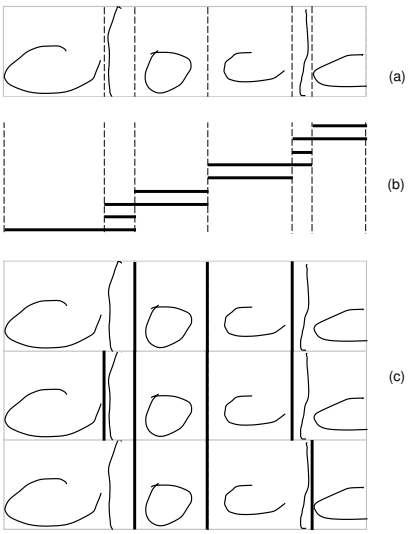

Fig. 2. **Handwriting segmentation:** (a) Tentative cuts are proposed by the segmenter to split the input representation into segments (dashed vertical lines).(b) Examples of segments delineated by two tentative cuts (bold horizontal lines). (c) Examples of segmentations (bold vertical lines).

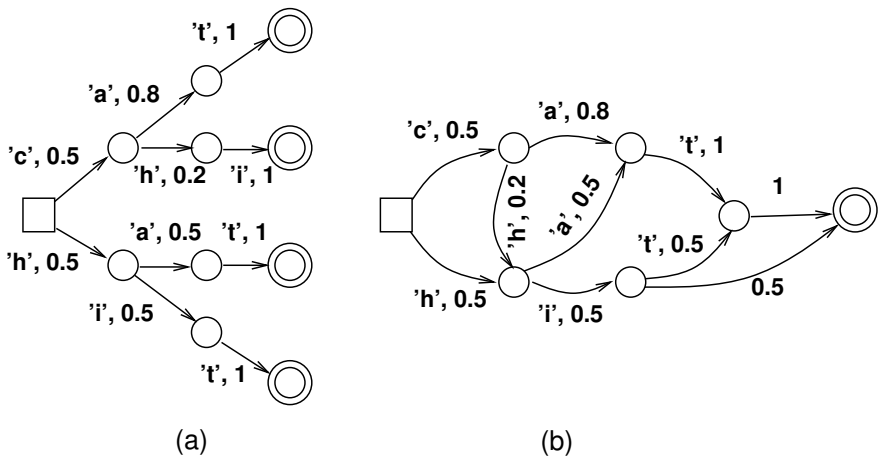

Fig. 3. **Regular grammars:** Regular grammars can be implemented by Weighted Finite State Automata. Nodes (circles) represent states. Arcs (arrows) represent transitions. The weights on the arcs represent conditional probabilities P(next-char|state). The overall score for a word is given by the product of the weights of the arcs traversed. (a) Lexicon tree for the words "cat", "chi", "hat" and "hit". Each node (state) can be reached from a single path starting at the root node (square). Therefore, each state "remembers" all the past characters since the beginning of the word. In this example, the score for "cat" is given by P("cat") = P("c") P("a"|"c") P("t"|"ca") = $0.5 \times 0.8 \times 1 = 0.4$. (b) Markov model of order $(n-1)$ or n-gram. Each state remembers at most $(n-1)$ past characters. In this example, the score for "cat" is given by P("cat") = P("c") P("a"|"c") P("t"|"c") = $0.5 \times 0.8 \times 1 = 0.4$.

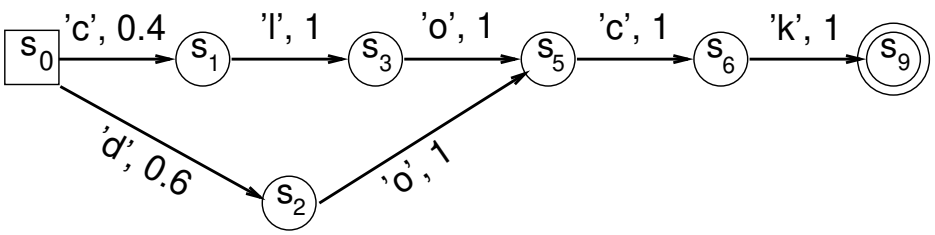

Fig. 4. **Graph-based segmentation and recognition:** The graph of a WFSA used for the recognition of the word of Fig. 2, incorporating classification scores, segmentation information, and a language model. The best path through the graph gives the final segmentation and recognition.

"best path through graph" problem. It is useful to unite the various approaches using the formalism of Weighted Finite State Automata (WFSAs) and Weighted Finite State Transducers (WFSTs) proposed in Ref. [55].

2.2.1. *Graph search techniques*

The basic idea of graph-based segmentation and recognition is to build a Weighted Finite State Automaton (WFSA) which incorporates classification scores, segmentation information, and a language model (Fig. 4). A WFSA is conveniently represented by a graph. It has finitely many states (the nodes of the graph), and a finite alphabet. Transitions between states (the arcs of the graph) are associated with the emission (or acceptance) of a character[e] and with a weight.[f]

If the weights are normalized like probabilities, the weight for a sequence of transitions is the product of the weights of the individual transitions; if, alternatively, the weights are normalized like costs (negative-log-probabilities), the weight for a sequence of transitions is the sum of the individual weights. Even though for computational reasons the negative-log-probability representation is often preferred, our discussion will use plain probabilities. A typical weight is

$$P(s', c|s) \tag{2.1}$$

which is the probability[g] of the transition along the arc $s \rightarrow s'$, with emission (or acceptance) of a character c.[h]

[e] We use the word character whether or not the recognizer is character-based; replace character by sub-character, sub-word or word if other atoms of handwriting are used.

[f] This representation of WFSAs with weights on the arcs only is completely general and equivalents can be found with representations with weights on nodes or on both nodes and arcs. For a proof, see Ref. [10]. For people familiar with HMMs, the WFSA framework corresponds to "discrete" HMMs. The equivalent of WFST for "continuous" HMMs has been proposed in Ref. [5].

[g] Note that any set of scores satisfying the axioms of probability theory can rightfully be called a probability.

[h] For notation simplicity, we omit a number of conditioning variables on our conditional probabilities. When we write, for instance, $P(s', c|s)$, we really mean $P(s', c|s, X, \mathcal{R}, \mathcal{S}, \mathcal{L}, \mathcal{D}, \mathcal{A}, \mathcal{U})$, where X is the input, $\mathcal{R}$ is the family of classifiers, $\mathcal{S}$ is the family of segmenters, $\mathcal{L}$ is the family of language models, $\mathcal{D}$ is the training data, $\mathcal{A}$ is the training algorithm, and $\mathcal{U}$ is the rest of the Universe.

The overall score for a path in the graph is given by the product of the weights of the arcs traversed:

$$P(s_1, c_1, s_2, c_2, ..., s_n, c_n) = \prod_{k=1}^{n} P(s_k, c_k | s_{k-1}) \quad , \tag{2.2}$$

where by convention $P(s_1, c_1|s_0) = P(s_1, c_1)$. Equation (2.2) embodies the assumption that the model is first-order Markovian.[i]

The probability of a given output string is given by summing over all possible ways of producing that string, i.e. by summing over all paths in the graph that produce the given string:

$$P(c_1, c_2, ..., c_n) = \sum_{\{s_1, s_2, ..., s_n\}} P(s_1, c_1, s_2, c_2, ..., s_n, c_n) \quad . \tag{2.3}$$

The most probable string is the recognition result. The forward algorithm [56] is an efficient dynamic programming technique to compute the above sum (2.3). An approximation often made is to pick the best single path in the graph as the recognition result. This amounts to replacing the sum in equation 2.3 by its largest term; the Viterbi algorithm [56] makes this approximation. Approximate search procedures to find the most probable string are often preferable for computational reasons. These include beam-search procedures and the $A*$ algorithm [21, 11]. Various "fastmatch" techniques can be used to narrow down the search space [21, 61, 41].

2.2.2. *Transduction cascade decomposition*

The Weighted Finite State Automaton (WFSA) used for recognition can be decomposed as a cascade of Weighted Finite State Transductions (WFST) corresponding to several intermediate stages of recognition.

Formally, a WFST is a mapping from pairs of strings over two alphabets to weights. WFSTs are represented by automata which accept a string of characters and emit another one. The arcs of the automata are labeled by pairs of characters $\lambda : \mu$, where λ is the character accepted and μ the character emitted.

Given two graphs, one can form their product, as defined below. The product of two WFSTs is another WFST. The product of a WFSA with a WFST is a WFSA. Let us denote by $s \xrightarrow{\lambda} s'$ the transition between state s and state s' with emission (or acceptance) of a character λ and $P(s', \lambda|s)$ the associated weight. Consider the two graphs:

$$\text{WSFA1: } s \xrightarrow{\lambda} s', \quad P(s', \lambda|s) \tag{2.4}$$

$$\text{WSFT: } t \xrightarrow{\lambda:\mu} t', \quad P(t', \lambda : \mu|t) \tag{2.5}$$

[i] This means that s_k depends on $s_{k-t} (1 < t \leq k)$ only through s_{k-1}. Markov models of higher order can be brought back to first order Markov models, by increasing the number of states (nodes) of the WFSA.

The product of WSFA1 and WSFT has states and transitions defined as:

$$\text{WSFA2: } st \xrightarrow{\mu} s't', \quad P(s't', \mu|st) = \sum_\lambda P(s', \lambda|s)P(t', \lambda : \mu|t) \tag{2.6}$$

where the labels are composed with the rule $(\lambda, \lambda : \mu) \rightarrow \mu$. Throughout the paper, we use the notation:

$$\text{WFSA2} = \text{WFSA1} \circledast \text{WFST} \tag{2.7}$$

to denote the transduction product. An example is shown in Fig. 5.

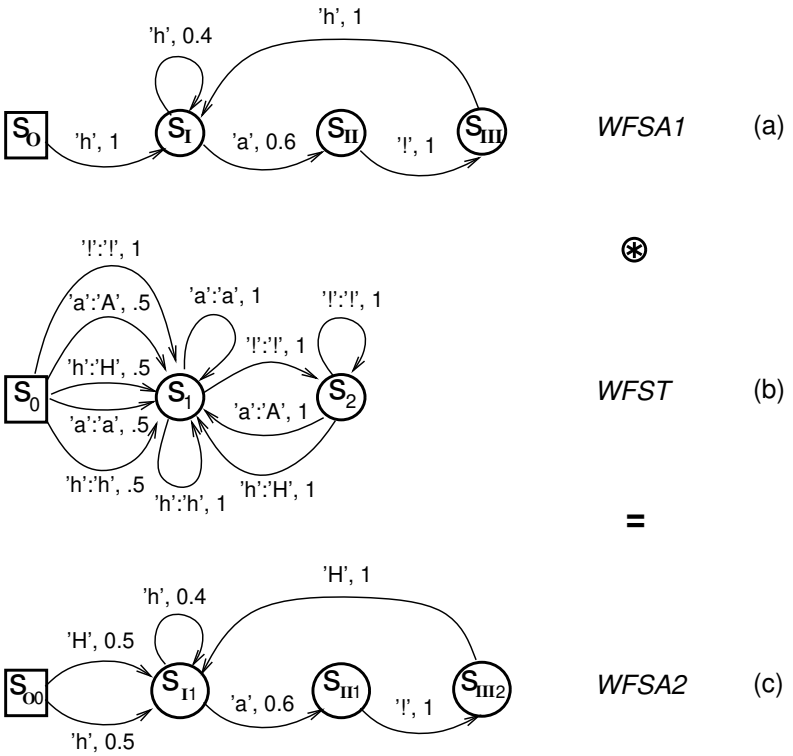

Fig. 5. **Transduction product:** (a) WFSA1. (b) WFST. (c) WFSA2 = WFSA1 $\circledast$ WFST.

In the general case, the product of two graphs has states composed of pairs of states of the original graphs. The product graph transitions emit and/or accept labels belonging to the Cartesian product of the labels emitted and/or accepted by the states of the original graphs. Similarly, the corresponding weights are a function of the pair of the corresponding original weights (e.g. sum or product). Examples of products of graphs are shown in Figs.17, 18, 19 and 20. For a more general description of operations on WFSAs and WFSTs, see Ref. [55].

2.2.3. *Search in a product graph*

Interestingly, in order to search for the best path in a product graph it is not necessary to explicitly build it. Those nodes in the product that are visited can be constructed "on-the-fly" during the search procedure (see Fig. 6).

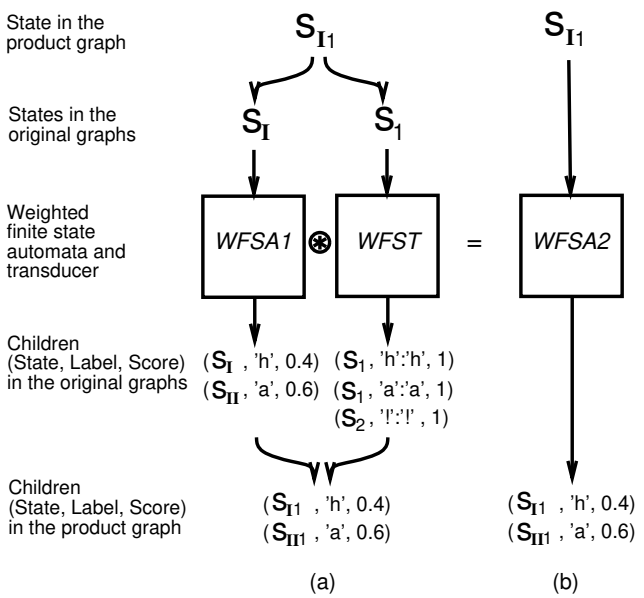

Fig. 6. **Search in a product graph:** This example is related to that of Fig. 5. Searching for the best path in a product graph does not require actually building the graph. Calculations can be done "on the fly". A "child state" is defined to be one that can be reached in one transition from a given state. The automata receive the following query: what are the children states of a given state, the label emitted during the transition and the corresponding probabilities? (a) WFSA1 and WFST are queried separately. The answers are combined to form the product states. Building WFSA2 explicitly is avoided, (b) WFSA2 is queried directly.

2.2.4. *Decomposition of a generic WFSA used for cursive recognition*

In our block diagram of Fig. 1, each stage corresponds to a transduction. There are two complementary strategies of segmentation that we refer to as *inseg* and *outseg*. Readers familiar with Hidden Markov Models (HMMs) will recognize their approach as an *outseg* strategy, whereas reader familiar with Neural Networks and "over-segmentation" techniques will recognize their approach as an *inseg* strategy. Both strategies are complementaries and can indeed be combined, although only a few attempts have been made. In the description that follows, we precise which stages are present or absent in *inseg* and *outseg* systems:

1. The first stage yielding the "input segmentation graph", considers only the low level segmentation information (such as pen-lifts, points of high curvature, etc.). This is where the segmentation really takes place for *inseg* systems (see

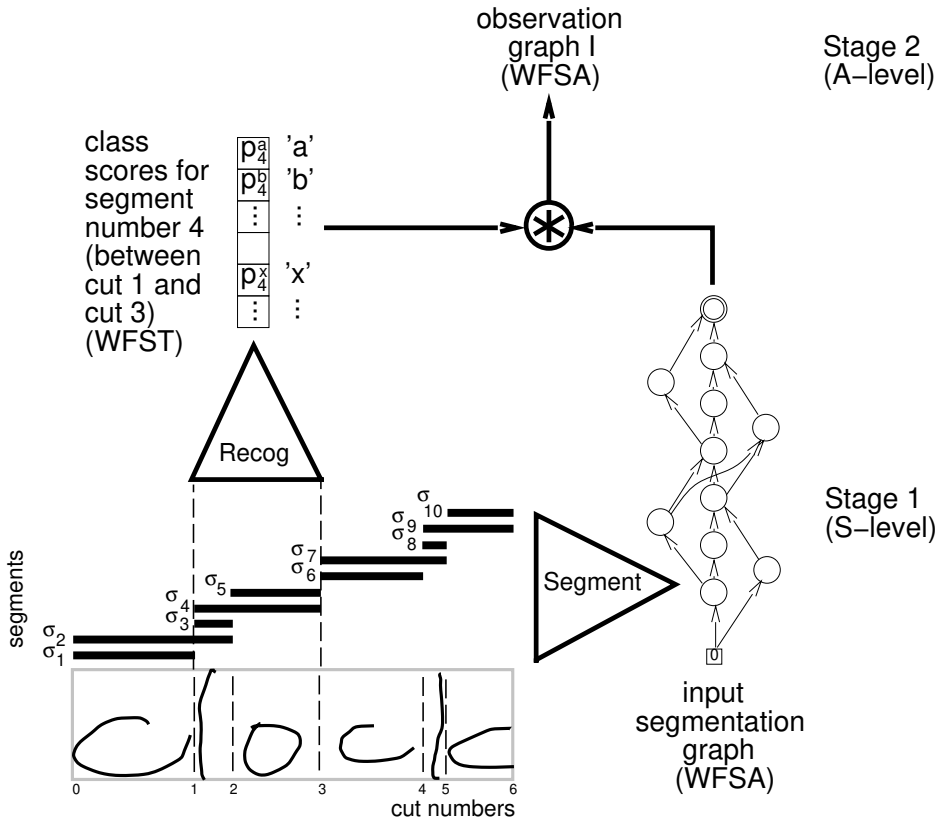

Fig. 7. **Central stages of *inseg* type systems:** The tentative cuts in the input representation are based on the dynamics of the pen (e.g. pen-lifts, pen velocity) and/or geometrical clues (e.g. spaces, corners). The input segmentation graph is non trivial, which gives the name *inseg* (for input segmentation) to the method. To simplify the diagram, we show examples of handprinted characters but the principles apply as well to cursive handwriting. The details of the various stages are shown in Figs. 19 and 20.

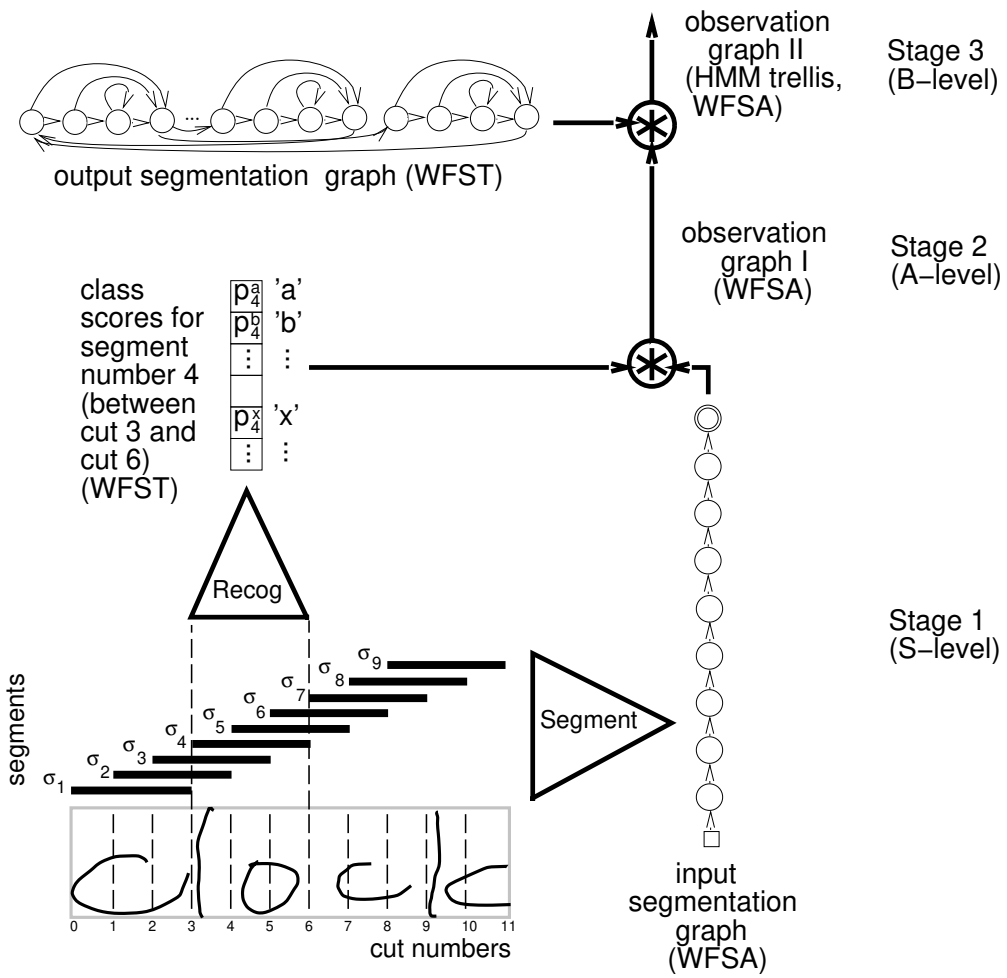

Fig. 8. **Central stages of** *outseg* **type systems:** The input segmentation graph is trivial. The segmentation is postponed until after the recognition, which gives the name *outseg* (for output segmentation) to the method. To simplify the diagram, we show examples of handprinted characters but the principles apply as well to cursive handwriting. The details of the various stages are shown in Figs. 17 and 18.

Fig. 7). By contrast, the first stage is trivial in the case of *outseg* systems (see Fig. 7 and Sec. 3.9). For people familiar with classical Hidden Markov Model approaches (HMM), HMMs are particular cases of *outseg* systems: the input segmentation graph consists of a chain of states representing the succession of time frames.

At the first stage level, the input alphabet is composed of triplets of numbers (x_t, y_t, z_t) and the output alphabet of segment labels $\sigma_1, \sigma_2, ...,$. The first stage provides weights $P(S|X)$ for all possible segmentations $S = \sigma_1\sigma_2...$, of the input X. These weights embody the quality or plausibility of segmentation, but no shape recognition.

2. The second stage combines the results of classification (shape recognition) to those of segmentation, yielding to the "observation-graph I". The input alphabet is $\sigma_1, \sigma_2, ...$, and the output alphabet may be alphanumeric characters or other type of sub-character, sub-word or word labels (depending on the granularity of segmentation) that we denote by $\alpha_1, \alpha_2, ...,$. Spaces or transitions may be included in that alphabet. The second stage provides weights $P(A|S)$ for all possible interpretations $A = \alpha_1\alpha_2...$, corresponding to the segmentations $S = \sigma_1\sigma_2...,$. These weights embody the scores of the classifier or shape recognizer.
3. The third stage adds a high-level model of segmentation (such as a model of character durations or a model of combination of strokes into characters), yielding the "observation-graph II". This third stage is absent from *inseg* systems (see Fig. 7 and Sec. 3.9) because they rely only on the first stage for segmentation. It is where the segmentation really takes place for *outseg* systems and classical Hidden Markov Model approaches (see Fig. 8 and Sec. 3.9). The input alphabet $\alpha_1, \alpha_2, ...$, and the output alphabet $\beta_1, \beta_2, ...$, are usually composed of either alphanumeric characters or word labels. The third stage provides weights $P(B|A)$ for all possible interpretations $B = \beta_1\beta_2...$, corresponding to the lower level strings $A = \alpha_1\alpha_2...,$. These weights embody the knowledge of the grouping of sub-units into units: if the α's correspond to stroke sub-units, the β's may correspond to character or word units; if the α's are characters, the β's may be words. The third stage may include several sub-stages, e.g. strokes-to-characters followed by characters-to-words.
4. The last stage adds a grammar (a linguistic model or a simple lexicon), yielding the observation-grammar product graph ("OG product-graph"). The input alphabet $\beta_1, \beta_2, ...$, and the output alphabet $c_1, c_2, ...$, are both either alphanumeric characters or word labels. The last stage provides weights $P(C|B)$ for all possible interpretations $C = c_1c_2...$, corresponding to the lower level strings $B = \beta_1\beta_2...,$. These weights embody the knowledge about the language.

Table 1. **Factors of a transduction cascade for a generic recognition system.**

Factor	Type of machine	Graph name	Characters accepted/emitted
$P(S\|X)$	WFSA	input segmentation	σ
$P(A\|S)$	WFST	classification	$\sigma : \alpha$
$P(A\|X)$	WFSA	observation I	α
$P(B\|A)$	WFST	output segmentation	$\alpha : \beta$
$P(B\|X)$	WFSA	observation II	β
$P(C\|B)$	WFST	grammar	$\beta : c$
$P(C\|X)$	WFSA	og product	c

The final WFSA in Fig. 1 is the observation-grammar product graph ("OG product graph"). It calculates $P(C|X)$ which can be factored as:

$$P(C|X) = \sum_{\{B,A,S\}} P(C|B)P(B|A)P(A|S)P(S|X) \tag{2.8}$$

where the factors represent the successive intermediate steps of recognition, as summarized in Table 1. Each step is conditionally independent of the previous step but may be differently conditioned on various external sources of information, such as the families of classifiers, segmenters and language models, the training data, and the training algorithms.

2.3. *Training*

Training consists of adjusting the parameters of the system to get the best recognition performance by using the examples of a training set.

Often some parameters of the system are adjusted manually, while others are trained automatically. Usually, the classifier contains most of the trainable parameters. As computers become more and more powerful, there is a tendency to rely more and more on the training data and to adopt a statistical classifier with many (hundreds of thousands) of adjustable parameters. Neural networks and decision trees are popular methods. Researchers originating from the speech community often prefer Gaussian mixtures and Radial Basis Functions (RBF).

The most popular training method is gradient descent which involves making incremental changes to the weights according to the negative gradient of the training error:

$$\Delta w = -\eta \frac{\partial E}{\partial w} \quad , \tag{2.9}$$

where η is a proportionality constant, the "learning rate".

Maximum likelihood techniques can be included in this formalism, by choosing E to be $-\log L$, where L is the likelihood function:

$$L = \prod_k P(X^k|C^k, \mathcal{U}) \quad , \tag{2.10}$$

where (X^k, C^k) are the training examples (data, class) and $\mathcal{U}$ represents other conditioning variables, including the family of classifiers, the family of segmenters, the family of language models, and the training data, the training algorithms.

Provided that all the stages are differentiable, it is possible to optimize globally the recognition system by back-propagating gradients, similarly to the back-propagation algorithm used to train neural networks [57] and the forward-backward procedure (or Baum-Welsh algorithm) used to train Hidden Markov Models [56]. This means that the weights of the segmentation graphs and of the classifier, plus other preprocessing parameters eventually, are all simultaneously optimized to minimize the training error.[j] Examples of global training applied to handwriting recognition are found in [13, 4].

When the sum over all paths in equation 2.3 is replaced by a maximum, a two-step algorithm, similar to an Expectation-Maximization or EM procedure [12], is used. In the Expectation step, the weights of the system are frozen and best segmentation for each training example (X^k, C^k) is determined. In the Maximization step, the weights are modified using gradient descent (equation 2.9) to maximize the likelihood (minimize the training error), using the segmentation thus defined. Steps E and M are iterated until convergence.

With either method above mentioned, no prior segmentation of the data is needed. For speed of convergence however, it is often recommended to pre-train the classifier on a set of isolated characters.

Techniques that improve recognition independently of the classifier family include: capacity control via cross-validation or Structural Risk Minimization [69], data cleaning [48], voting among several classifiers [72],[k] using an objective function which emphasizes learning of atypical patterns (works only on clean data) [25], and using a distortion model to enlarge the training set artificially [15].

3. Which Solution to What Problem?

In this section we discuss particular difficulties of on-line cursive handwriting and describe alternative designs of the main building blocks of the recognition systems.

3.1. *Intrinsic Ambiguity of Handwriting*

By definition, cursive handwriting contains connected characters which render recognition and segmentation problems inseparable. However, some recognition difficulties can be attributed to either shape recognition problems or segmentation problems.

Shape recognition is concerned with assigning class probabilities to the input given by the geometrical shape of the written character — including its form, relative position, and relative size.

Ideally, differences between several drawings of the same character would be less significant than differences between drawings of different characters. Unfortunately,

[j] This "global" optimization of the system does not imply that a global minimum of the training error is reached. Actually, gradient descent methods are prone to yielding local minima. This problem is alleviated with "stochastic gradient" techniques for which parameters are updated at each pattern presentation with the local gradient, as opposed to being updated at each "epoch" with the global gradient calculated with all the training patterns.

[k] See also the chapter by L. Lam et al. in this book.

there are a number of intrinsic shape ambiguities in the Latin alphabet: 0 (zero) versus uppercase O (as in Oscar), 1 (one) versus uppercase I (as in India) versus lowercase l (as in Lima), 2 (two) versus Z (as in Zulu), ; (semicolon) versus j (as in Juliet) which can be resolved only with the help of syntactic or semantic context. Less severe are size (e.g. C–c, O–o, S–s, U–u, V–v, W–w, X–x and Z–z) and position ambiguities (e.g. P–p, Y–y and 9–g) which can sometimes be resolved by comparing relative sizes and positions of characters. Aside from these intrinsic ambiguities, some writers write similarly pairs of characters such as a–u, u–n, M–m, m–n, F–f, r–v, 2–L, 3–z, 4–Y, 6–G, 6–b, i–I, j–J, l–L, g–S, k–K etc. Shape ambiguities are illustrated in Fig. 9.

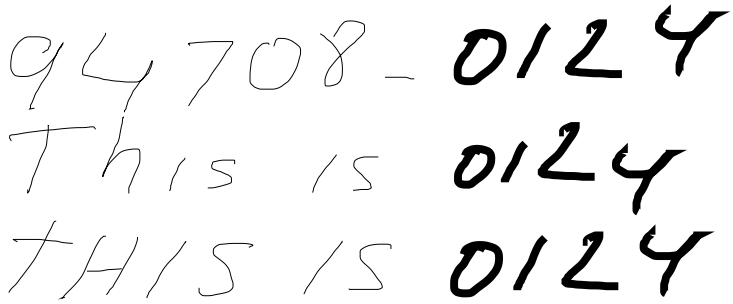

Fig. 9. **Ambiguity of character shapes:** The same four handwritten characters have different interpretations depending on their relative sizes and positions and on the syntactic and semantic context [54].

Character segmentation is the task of identifying character boundaries. In the case of consistently spaced characters, segmentation is fairly easy and can be done independently from the recognition process by setting a threshold on the space between strokes. For cursive and mixed-styles writing, segmentation points are harder to determine and can be very ambiguous. Segmentation cannot be done independently of recognition (Fig. 10).

One way to circumvent shape and segmentation ambiguities is to design a new alphabet with unambiguous shapes, each of which consists of a single stroke or "unistroke" [23]. Although this approach seems to defeat the purpose of using the pen as a more natural computer user interface than the keyboard, the company Palm Computing has introduced recently to the market a product called Graffiti which uses a unistroke alphabet that has been reasonably well accepted.

3.2. *On-line* versus *Off-line*

Strictly speaking, *on-line* recognition means that recognition takes place as the user is writing. In the handwriting recognition jargon, *on-line* also refers to a special type of input data containing information about the pen trajectory as given, for instance, by a digitizing tablet (Fig. 11 (a)). A typical application of on-line handwriting recognition is pen computer interfaces. Pen computers impose constraints on the speed of the processor (usually a low power PC) and the memory

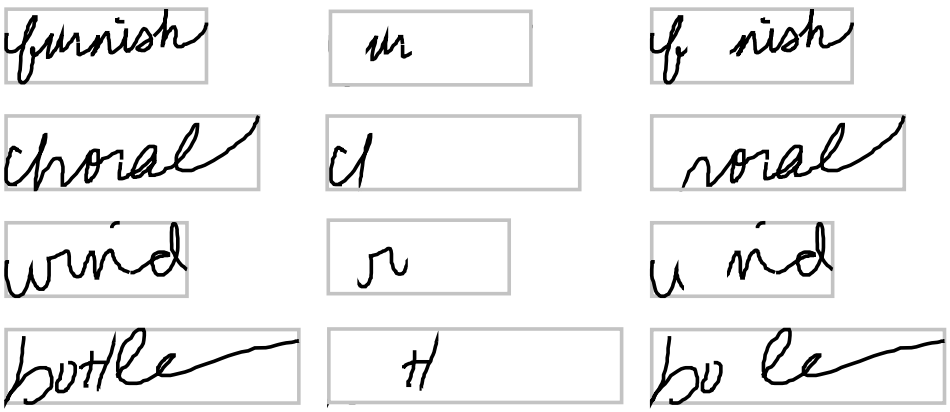

Fig. 10. **Ambiguity of segmentation and benefit of using context:** (a) Examples of cursive writing. (b) Segmentation ambiguities that cannot be resolved without context. c - Contexts that narrow down considerably the number of candidate characters that could fit in the gap.

availability (usually less than 500 Kbytes). A speed of at least 10 characters per seconds is required for the user to feel comfortable.

In contrast, *off-line* recognition takes place sometimes long after handwriting. The raw input data is most often a digitized image or "pixel map", obtained, for instance, with a scanner (Fig. 11 (b)). This is why "off-line" character recognition and Optical Character Recognition (OCR) are usually synonyms. Typical OCR applications include post office address readers and bank check readers. The recognizers must often process hundreds of characters per second but the customers can usually afford fast processors and large amounts of memory.

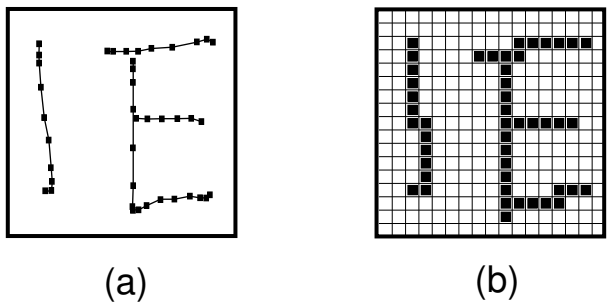

Fig. 11. **On-line and Off-line data.** (a) On-line data consist of a sequence of time ordered coordinate points; the writing trace is known. (b) Off-line data consist of image pixel data. They are space ordered and the writing trace is unknown.

There is a natural temptation to convert pen trajectory data to pixel images and process them with an "off-line" recognizer. But, the on-line handwriting recognition problem has a number of distinguishing features which must be exploited to get best

results:

- **preprocessing** operations such as normalization, smoothing, deslanting and deskewing and dehooking and **feature extraction** operations such as the detection of line orientations, corners, loops and cusps are easier and faster with the pen trajectory data than on pixel images.
- **discrimination** between optically ambiguous characters (for example "j" an ";") may be facilitated with the pen trajectory information.
- **segmentation** operations are facilitated by using the dynamic information and the pen-lifts.
- **immediate feedback** can be given by the writer whose corrections can be used to further train the recognizer.

For those reasons, on-line recognition is usually found easier than off-line recognition. In an experiment [45], on-line handwritten uppercase characters were converted to off-line data. The error rate increased from originally 4% to 10% resulting in a significantly superior recognition performance for on-line data.

In unfavorable cases, however, temporal information can actually complicate the recognition. For example the letter "E" can be written in one to four strokes and various stroke orders and still produce the same static image. Late-written "i" dots and "t" crosses can cause a problem because they can be far apart in time from the rest of the corresponding character. A reasonable compromise is to perform the preprocessing steps on the pen trajectory data but then generate a time independent representation [4].

Conversely, on-line techniques can be used for off-line recognition. This requires reconstructing temporal information from static off-line data (e.g. [14]). However, this approach is still more a curiosity than a competitive way of handling off-line recognition.

3.3. *Cursive* versus *Handprint*

There is a debatable distinction between "cursive handwriting" and "natural unconstrained handwriting". Indeed, one is necessarily a subset of the other, but is arguable whether it is desirable to build recognizers for pure cursive styles, knowing that many people write in mixed styles which incorporate both cursive and handprinted styles (see Fig. 12). For that reason, we have concentrated our attention on the "mixed style" problem rather that pure cursive.

The problem of segmentation is harder for cursive and mixed style handwritings than for handprint. Since the writer is not constrained to lift the pen between characters, there is a quasi continuum of possible segmentation points, as opposed to a limited number of pen lifts. Two strategies are commonly used (Fig. 13):

- **Making heuristic provisional segmentation points**, usually based on pen lifts, extrema of the speed of the pen or on points of high curvature. Hypothetical characters are then obtained by grouping 1 to n elementary segments defined by these provisional segmentation points, where n does not exceed 4 or 5.

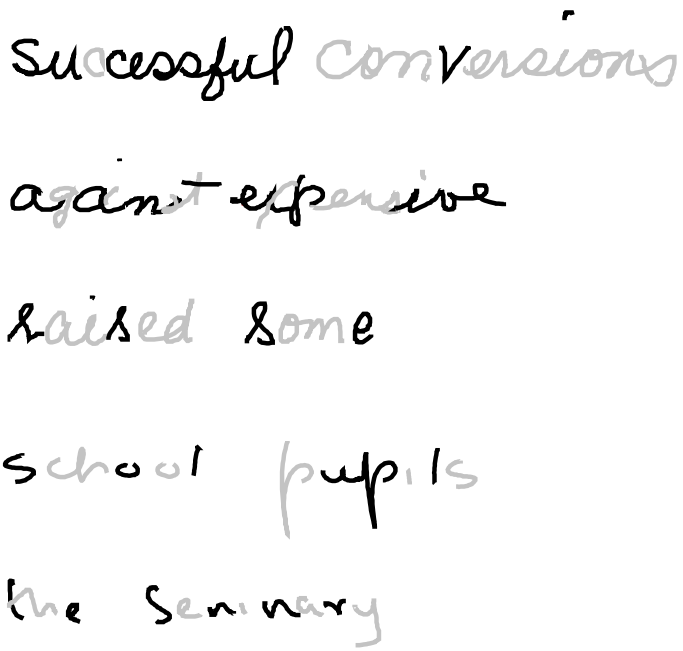

Fig. 12. **Mixed styles:** Many writers do not have a pure cursive or handprinted style. The figure shows handwriting samples with various amounts of pen lifts which are indicated by alternating black and grey ink.

- **Making regularly spaced provisional segmentation points** along the pen trajectory. Hypothetical characters are then obtained by grouping a fixed number of elementary segments defined by these provisional segmentation points. This is equivalent to sweeping a window of fixed size over the input.

3.4. *Local* versus *Global Representations*

After all low level preprocessing operations have been performed (i.e. normalization, smoothing, deslanting and deskewing, dehooking etc.) the system must generate a data representation which is suitable for the classifier. Representations can be either very faithful to the original data, in which case most of the work is left to the classifier or, on the contrary, composed of sophisticated high level features.

3.4.1. *Local Representations*

The representations that are most faithful to the data preserve the finely sampled pen trajectory information. An advantage of such representations is that they do not make any early and potentially "information destructive" decisions. This is particularly important if the segmentation decisions are postponed until the last stages of the system, as in the *outseg* architecture (see Sec. 3.9).

Preprocessing for pen trajectory representations commonly includes resampling to increase, reduce, or standardize the number of points. Resampling might preserve

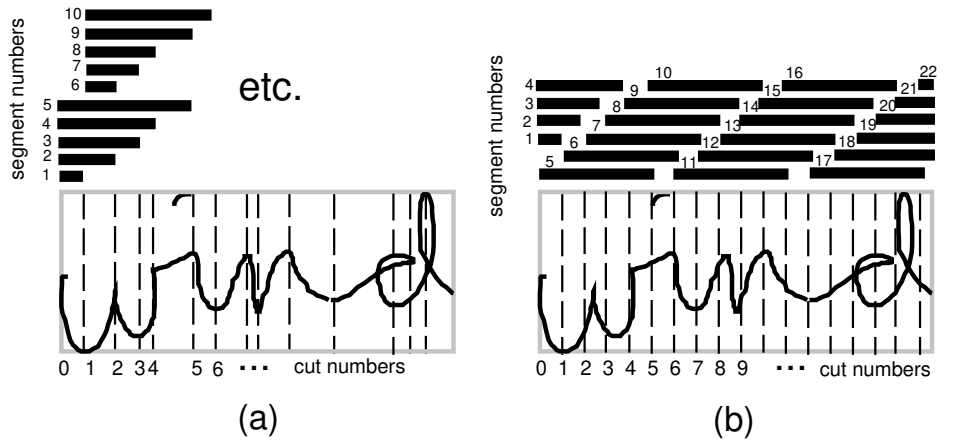

Fig. 13. **Cursive handwriting segmentation:** The input representations are split into segments to be presented to the classifier. Each segment (indicated by a bold horizontal bar) is delineated by two tentative cuts (indicated by vertical dashed lines). Input representations are generally not images and tentative cuts are detected using the pen trajectory information. These two-dimensional representations are only shown to help the reader visually. a - Tentative cuts generated at the zeros of the vertical velocity. (b) Tentative cuts regularly spaced.

the initial time spacing or produce points regularly spaced in arc length. Because of inconsistencies in writing speeds, the latter solution is preferable. One difficulty is to determine the appropriate sampling step. For instance, the sampling step can be a fixed fraction of the character core height (height of a character such as "a", "e", "o", or"u" which have no descenders nor ascenders) if that one is accurately determined.

Many features of the pen-trajectory have been suggested as being particularly relevant for recognition [34, 46, 61]; standard choices include local direction and local curvature. Pen trajectory representations suffer from the difficulty of detecting and reordering delayed strokes and dealing with multiple variants of the same character which differ in pen trajectory but not shape. A solution to that problem is to do all the preprocessing steps using the pen trajectory information but then arrange the features in a two-dimensional map [4].

3.4.2. *Global Representations*

Another approach which preserves the pen trajectory ordering, but not the fine sampling, is to encode sub-characters with global features.

Several authors base their method on a pre-segmentation of the data into "ballistic" strokes, that are pen traces delimited by zeros of the speed of the pen (zeros of the vertical, tangential or angular velocity) and pen lifts. Each ballistic stroke is then encoded with a small number of parameters (typically between 3 and 10 parameters). For instance, strokes can be modeled as arcs of circles with only 3

parameters: the tangent at the first point, the radius and the total arc length. In Ref. [68], using signal processing arguments, the author concludes that 4 parameters per stroke are sufficient to obtain a faithful encoding. The original data, usually sampled at 100 points per seconds, is very redundant. Substantial compression rates of 5 to 10 have been achieved by using such handwriting models [33, 1].

Compressed representations have the advantage of alleviating overfitting problems from which the classifier may suffer: classifiers with a small number of adjustable parameters require less training data. However this argument is weak, especially since large databases are becoming publicly available [26][l]. The danger of making early decisions about segmentation is that character segmentation points are not necessarily a subset of the ballistic stroke segmentation points (Fig. 14).

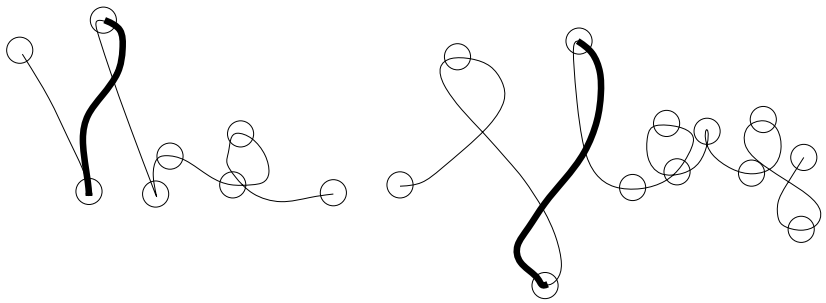

Fig. 14. **Danger of using a segmentation method independent of recognition:** Ballistic strokes are often used as atoms of handwriting. Unfortunately, some ballistic strokes (e.g. the bold pen traces on the figure) may belong to several characters.

3.5. *Word-based, Character-based or Stroke-based Recognition*

Most of our discussion so far was centered on the idea of recognizing characters. Characters seem to be a natural atom of handwriting because we are taught in school how to handwrite characters. However, other sub-divisions of handwriting may be better for recognition purpose, depending on the nature of the task:

- **Word recognition**: for small vocabularies (no more than a few hundred words), it is justified to recognize entire words. Problems of normalization such as core height detection are avoided. Global features on the word shape can then be used. Segmentation is simple and based on word spacing. Simple template matching can eventually be used for writer dependent word recognition.
- **Stroke-based recognition**: many characters share common sub-characters. The idea is to create an alphabet of sub-characters and base the recognition on those sub-characters.
- **Glyph recognition**: atoms of handwriting can also be subwords consisting of more than one character. This notion is particularly relevant to cursive

[l] See also the chapter by Guyon, Haralick, Hull and Phillips in this book.

recognition where many frequently written groups of characters are severely distorted and cannot be segmented into individual character shapes (Fig. 15).

Fig. 15. **Subword units as handwriting atoms:** The bold pen traces show word endings whose characters are difficult to recognize individually but that are not ambiguous as a whole. This suggests that some subword units (or glyphs) could be used as handwriting atoms.

3.6. *Invariant Representations* versus *Invariant Metrics*

Most systems rely on the preprocessing to enforce invariances with respect to translation, scale, skew, and stretching. Another strategy is not to normalize the data but use a classifier which is insensitive to such geometrical distortions [64, 71]. Assume, for instance, that we are interested in building a word recognizer based on template matching. The "tangent distance" [64] method allows to find the closest template, up to given geometrical transformations, including translation, scaling and hyperbolic transformations (Fig. 16).

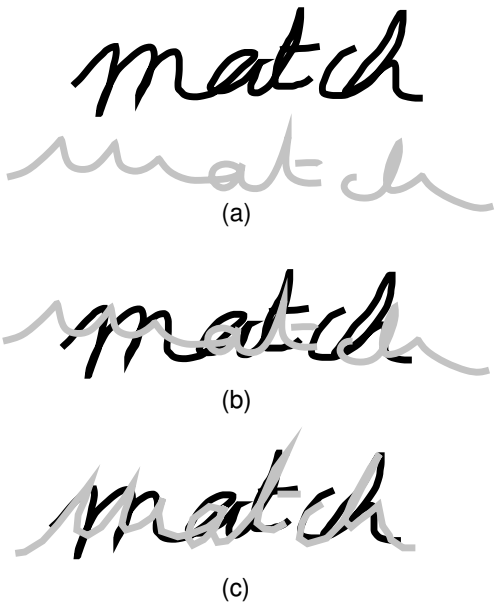

Fig. 16. **Invariant metric:** (a) Original words. (b) Words superimposed. (c) Words matched with the "tangent distance".

3.7. *Direct Estimation of* a posteriori *Probabilities* versus *Analysis by Synthesis*

Another debatable question is whether the recognizer should be a model of handwriting generation.

Handwriting generation can be described in the classical "noisy channel" framework which has been successfully applied to speech recognition for a number of years. The writer's brain produces strings of characters C with a probability $P(C)$. These strings are then "distorted" by the motor control system which generates handwriting characterized by a probability density function $p(X|C)$. $p(X|C)$ is the probability density function over the geometrical shapes X that can occur given the intended character sequence C.

Analysis by synthesis is a recognition method which consists of approximating $p(X|C)$ usually with a Hidden Markov Models and $P(C)$ by a statistical language model. Using Bayes rule we obtain the desired posterior probabilities:

$$P(C|X) = \frac{p(X|C)P(C)}{p(X)} \quad . \tag{3.11}$$

The trouble is that this framework does not apply well to a large number of recognition systems which do not make any attempt at modeling handwriting, i.e. do not compute an approximation of $p(X|C)$.

Fortunately, the transduction cascade framework presented in Sec. 2.2 is consistent with the direct estimation of $P(C|X)$. Furthermore, the same framework applies as well to the analysis by synthesis method and $p(X|C)$ can be decomposed into several intermediate synthesis steps.

3.8. *Weight Arithmetic*

When using graph algorithms, there are a number of subtle issues with weight normalizations and weight arithmetic. We list some of them here in an arbitrary order and leave some questions open:

- **Normalization of the classifier scores:** Not all classifiers provide estimates of probabilities or at least well behaved probability-like scores that add up to one. Graph algorithms do not require using probabilities. It is often better **not** to normalize the scores provided by the classifier since there may be good reasons why they do not add up to one (e.g. there is an implicit extra class for "meaningless" patterns which accounts for the lost probability mass). It is only important to determine whether the scores are multiplicative or additive [13].
- **Correction of "wrong" priors:** Sometimes the training data has very biased frequencies of words, characters, or groups of characters. As a consequence, some classes may have systematically too low or too high scores. Normalizing the score of each class by its average value over positive training examples of that class is one possible way to alleviate that problem. A better solution is to design the training set in a way that avoids this problem.

- **Normalization of the weights of WFSTs:** WFSTs implement conditional probabilities. Consequently, one should not expect that the weights of the outgoing arcs of a given node add up to one.
- **Normalization of the weights of the OG product graph:** In the transduction cascade framework, either the observation graph is a WFSA and the grammar graph a WFST (in the case of the direct estimation of *a posteriori* probabilities) or *vice versa* (in the case of the analysis by synthesis). In practice, both graphs are often represented as WFSA graphs which emit (or accept) characters of the same alphabet. The weight normalizations of the product graph are usually not performed.
- **Normalization with respect to the amount of ink consumed:** A classical problem is that of segmentation graphs which favor short strings of characters. This arises because string probabilities are obtained by multiplying the probabilities of its constituents. Therefore, the fewest factors, the most probable! This problem is automatically corrected if the weights of the segmentation graph are globally optimized during training with a discriminant criterion. If the weights are set "by hand", the amount of ink consumed should be accounted for.
- **Multiplicative or additive weights:** Whether the scores represent probabilities or log-probabilities is not the only criterion to decide on their arithmetic. For instance, the observation graph and the grammar graph can be thought of as two "experts" which estimate independently $P(C)$ under different conditions. The overall recognition system could combine the expertise of these two experts by weighing their opinions $P_o(C)$ and $P_g(C)$ according to how much they can be trusted: $\gamma P_o(C) + (1-\gamma)P_g(C)$. This contrasts with the approach presented in Sec. 2.2 which suggests to combine the scores as: $P_o(C).P_g(C)$.

3.9. *Input* versus *Output Segmentation*

The question is: should we take advantage of the geometrical and dynamical properties of the pen trace to pre-segment handwriting before any recognition takes place (*inseg* strategy, Fig. 7) or should we make as few assumptions about segmentation as possible before recognition and perform recognition and segmentation simultaneously (*outseg* strategy, Fig. 8). Few studies have been made to compare the two approaches using the same classifier and the same data [60, 70]. In principle, *outseg* should be better for cursive since it is almost impossible to decide where the cuts are before recognizing the characters. However, we do not know of any systematic study on cursive recognition which has demonstrated that this is actually true.

The input segmentation graph of the *outseg* method is trivial (Fig. 17). All the knowledge of the designer is put into the architecture of the output segmentation graph (Fig. 18).

The input segmentation graph of the *inseg* method is nontrivial (Fig. 19). It

connects segments using geometric rules. The simplest rules are to connect segments delineated by cuts (i, j) to segments delineated by cuts (j, k), thus forbidding reusing the same ink in more than one character.

Computationally, *inseg* is more economical because only a few segments are considered for classification. This results not only in fewer calls to the classifier, but also in a smaller observation graph and observation-grammar product graph (Fig. 20). Therefore, searching for the final segmentation and recognition solution is faster for *inseg* than for *outseg*, unless some clever search pruning techniques are implemented.

For best recognition accuracy, a combination of *inseg* and *outseg* approaches may be the solution [20, 27], but combined approaches are still in their infancy.

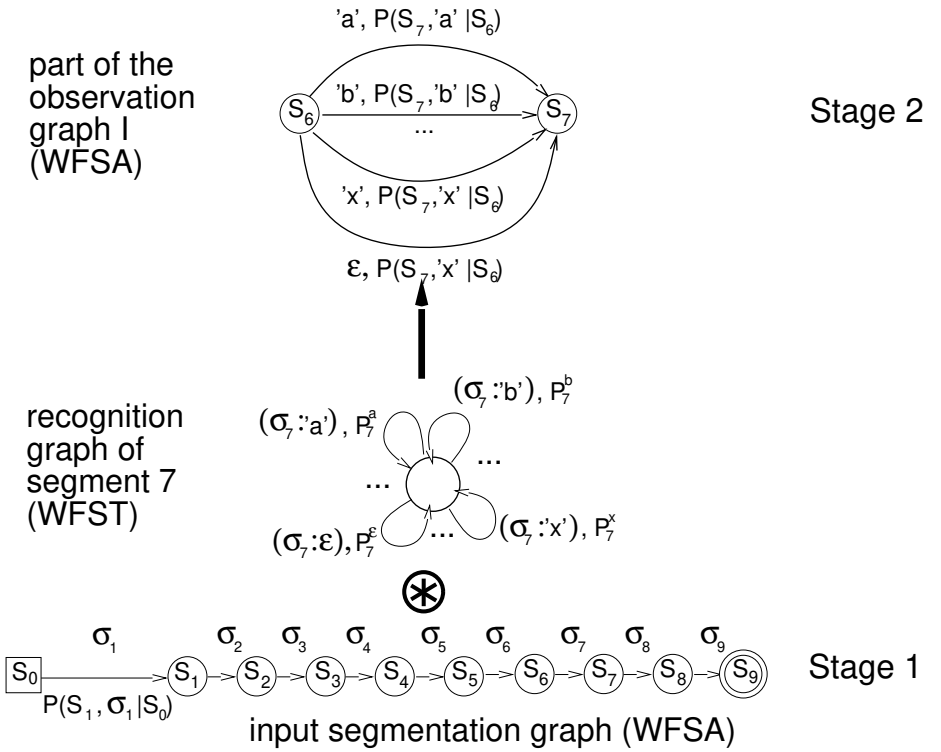

Fig. 17. **Calculation of an *outseg* observation graph I:** This figure is related to Fig. 8. The notation ε represents an empty character (no character observed). ε characters are emitted by segments that are not well aligned with and actual handwritten character (e.g. ligatures).

3.10. *Writer Adaptation*

User expectations with respect to handwriting recognition accuracy are around 1% character error rate. Error rates larger than 2% character error are intolerable: they require too much user attention to make corrections and slow down text

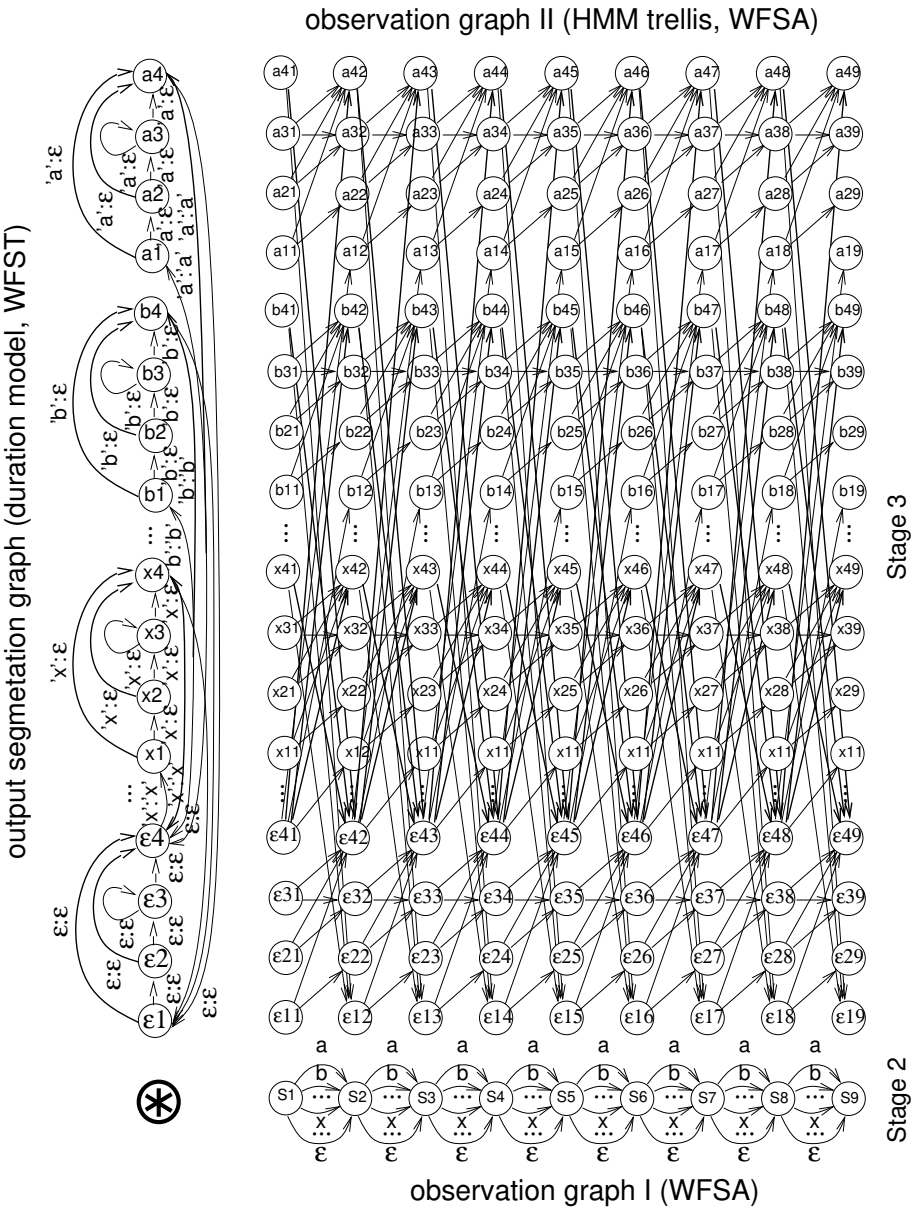

Fig. 18. **Calculation of an** *outseg* **observation graph II (HMM trellis):** This figure is related to Fig. 8. In this particular example, the output segmentation graph is a duration model such as the ones used in Sec. 2.4. It forces alternating between a character model and a model of transitions between characters or ligatures.

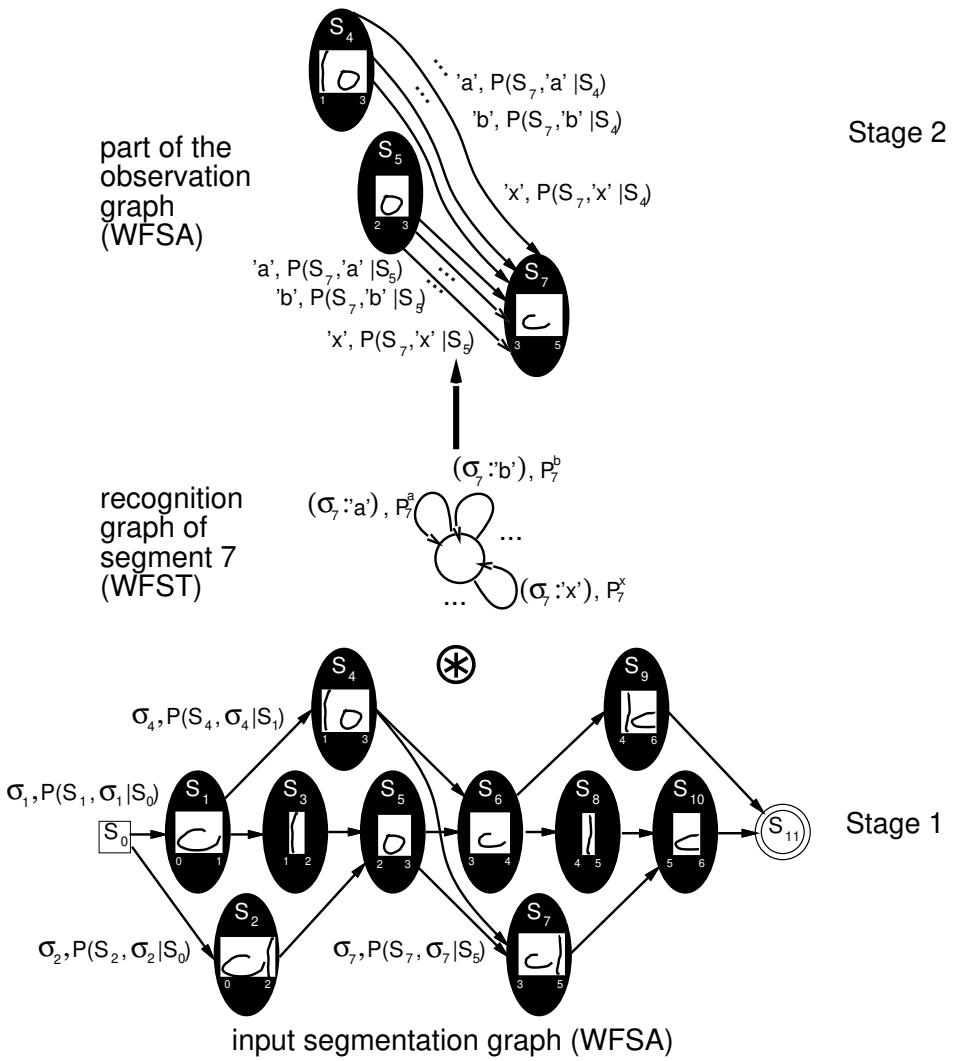

Fig. 19. **Calculation of an *inseg* observation graph:** This figure is related to Fig. 7. The notation $(\sigma_7 : \text{'}a\text{'})$ means that the classification automaton accepts segment σ_7 and emits character 'a'. The weights of the observation graph are products of weights of its two parent graphs. For instance: $P(s_7, \text{'}a\text{'}|s_5) = P(s_7, \sigma_7|s_5)P_7^a$.

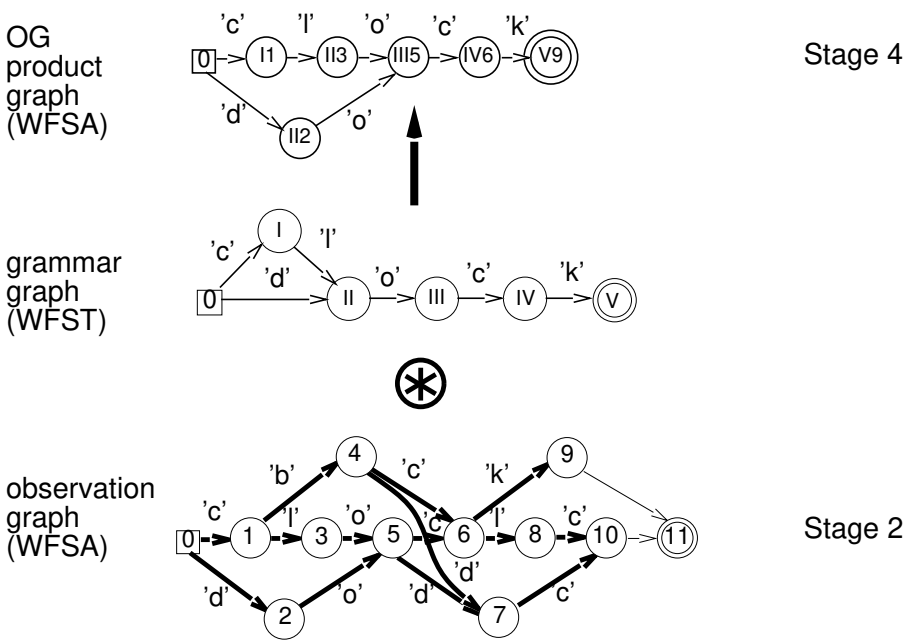

Fig. 20. **Calculation of an *inseg* observation-grammar product graph:** This figure is related to Fig. 7. In the observation graph, a bold arrow represents multiple connections, one per class. Only the character corresponding to the best score is represented. The grammar graph is a transducer which accepts an emits the same characters. The notation 'c' : 'c' is therefore simplified to 'c'. The weights are not represented. Note that stage 3 (observation graph II) is absent from *inseg* systems.

entry considerably [28]. Even for limited size vocabulary, such low error rates are very difficult to achieve (see Sec. 2.5). They are probably out of reach of writer independent recognizers, for the time being.

Although "walk-up" recognition accuracies may be quite poor, users do adapt to a particular recognizer and often get the illusion that the recognizer improves over time, even if in reality it is the user who improves. Commercial products have been reluctant to introduce any other form of adaptation because of the danger of degrading the recognition performance rather than improving it. It is indeed very difficult to design a writer adaptation training algorithm which is robust against mis-uses such as intentional or non-intentional introduction of meaningless patterns.

Technically, writer adaptation is not difficult to achieve under laboratory conditions [49, 17]. The basic idea is to retrain superficially the system by modifying only the weights of the upper levels.

In real world applications, writer adaptation is harder to deploy. The adaptation engine must be designed jointly with an appropriate user interface. Ideally, adaptation should be transparent to the user. The system would continuously learn from its mistakes, the corrections of the user and the redundancy of the language. This remains to be seen.

4. Example of an Entire System Design

In this section, we describe an entire system design as an illustration of the main concepts presented in the previous sections. The overall system is a word recognizer, consisting of a trainable character classifier and a language model. This gives us the flexibility of changing the language model, e.g. changing the vocabulary, without retraining the character classifier.

The system was designed to provide acceptable "walk-up" recognition accuracy — without forcing the writer to go through a training session and without imposing any restriction in writing style. The preprocessing and the classifier are designed to avoid making potentially wrong segmentation decisions at an early stage in the recognition process.

4.1. *Description of the System*

The main building blocks of our system, shown in Fig. 21, are a Time Delay Neural Network (TDNN) and Hidden Markov Model (HMM). The architecture is of the *outseg* type (see Sec. 3.9 for details). In Fig. 22, we show a schematic graph interpretation of the system in terms of transduction cascade.

The **data collection device** (Fig. 23) provides pen trajectory information as a sequence of (x, y) coordinates at regular time intervals (10-15 ms). The pen trajectory of one word is collected. For each word, the **preprocessing** begins by estimating the height of the *middle area* (or *core height*) with two linear regressions through the local minima and maxima of the y-coordinate, excluding the outliers. This results also in an estimate of the orientation. After the word has been normalized,

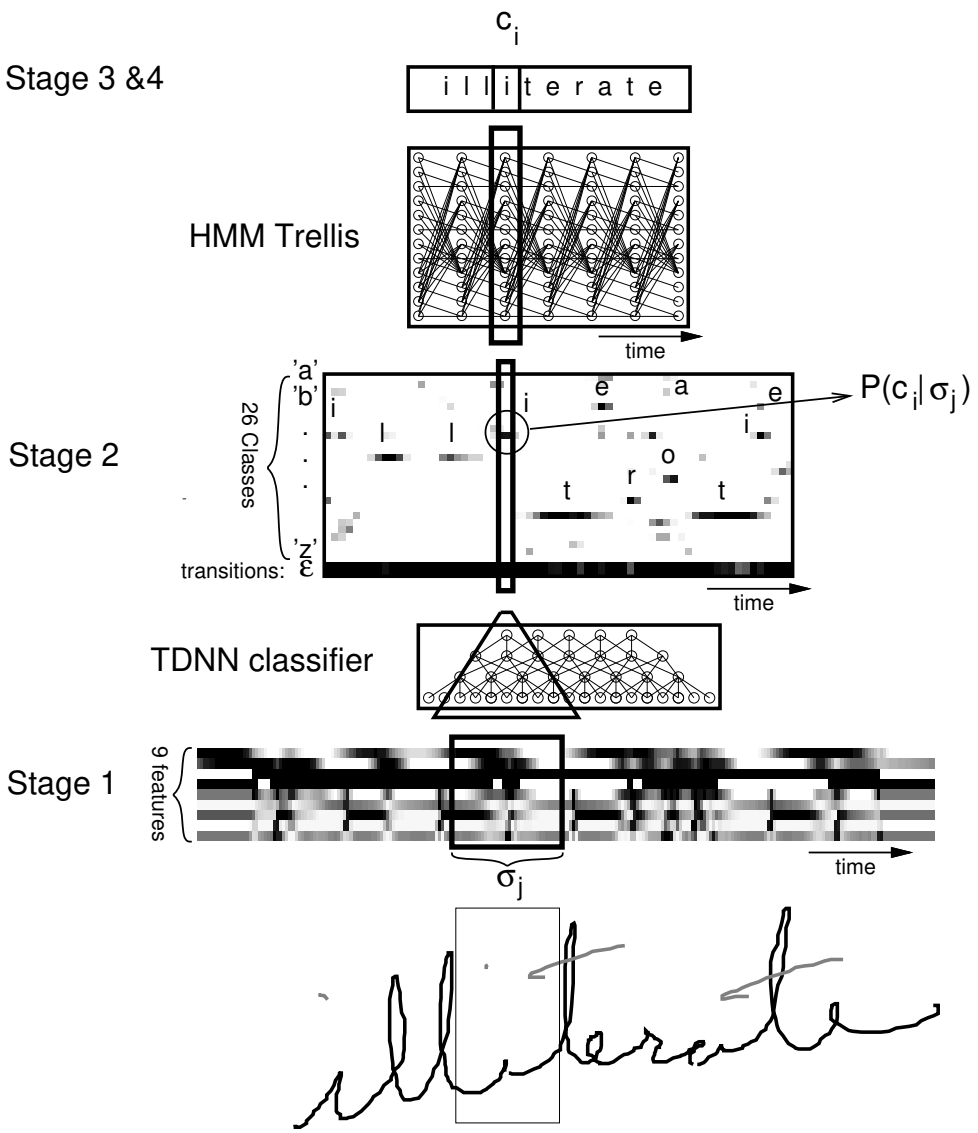

Fig. 21. **A TDNN-HMM hybrid system.** After the preprocessing, a time ordered sequence of feature vectors is presented to the Time Delay Neural Network (TDNN). The darkness indicates the amplitude of the coefficients and the color (black or white) their sign. The TDNN estimates *a posteriori* probabilities for characters in a word. A Hidden Markov Model (HMM) segments the word in a way which optimizes the global word score. If the HMM does not use any language model or grammar (no stage 4), the word "lliteroti" is the best interpretation provided by the system. If the HMM is forced to chose words from an English lexicon (stage 3 and 4 combined), the correct word "illiterate" is found.

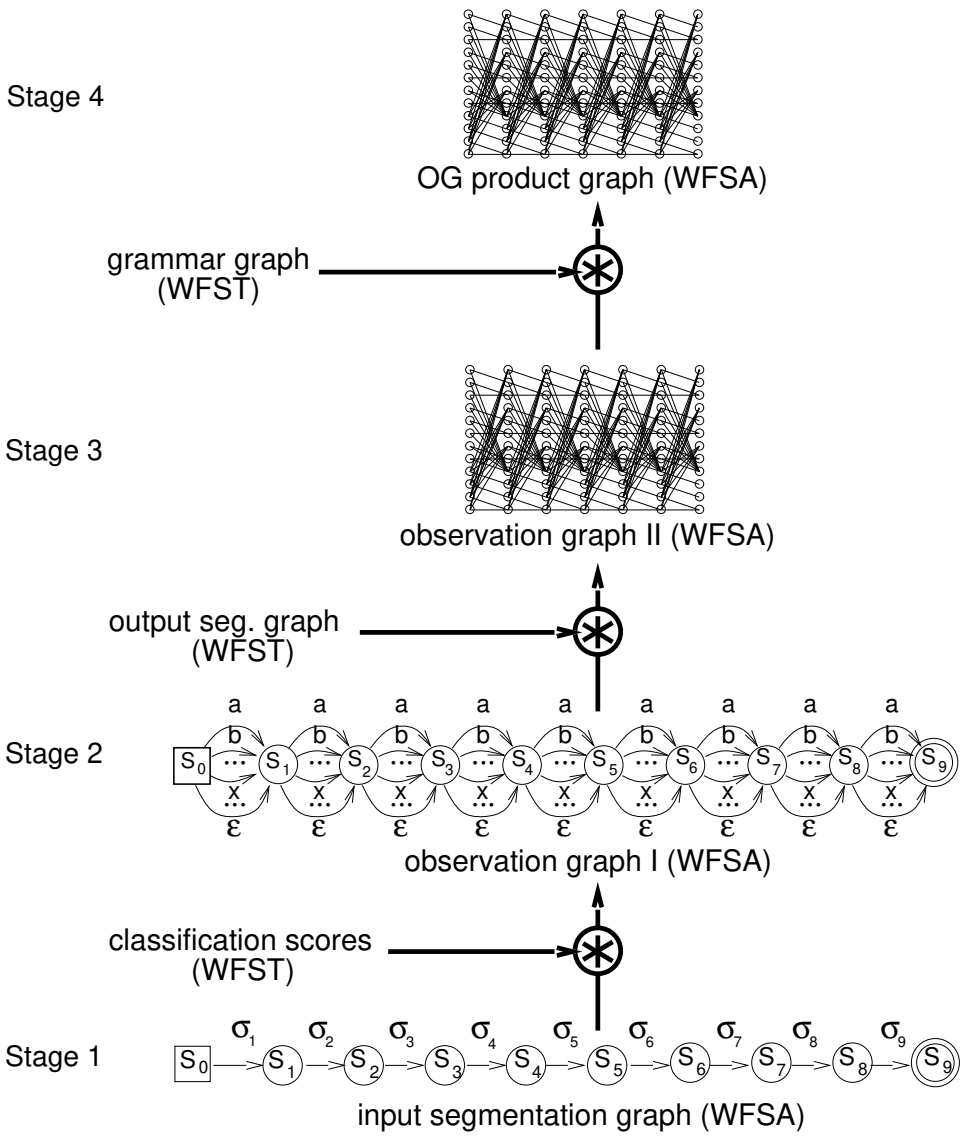

Fig. 22. **Graph interpretation of Fig. 21.** Each stage is obtained from the preceding one by a transduction product.

oriented and smoothed, we perform a resampling of the data to points regularly spaced in arc length. The spacing between points is a fixed fraction of the core height. We then compute local features of the pen trajectory at each point: slope, curvature and speed [24, 60]. Disjointed components such as diacritical marks, the dot on i, and the cross on t are removed and treated as additional features. The result of this preprocessing is a finely sampled sequence of time-ordered feature vectors at regular intervals in space. This representation is very faithful to the original data and is suitable both for cursive and printed handwriting.

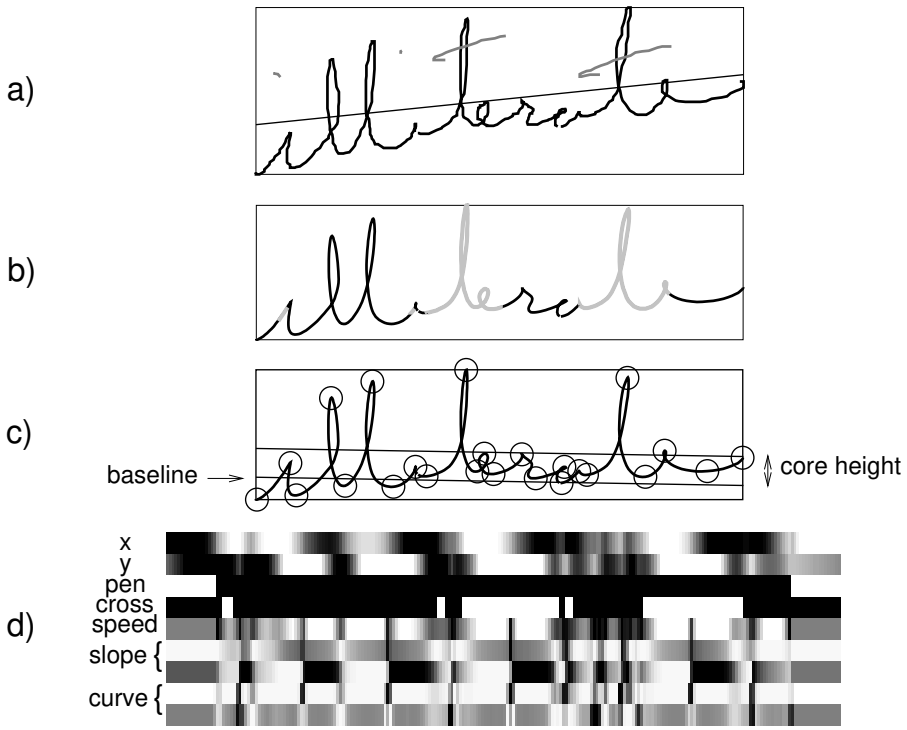

Fig. 23. **A preprocessing suitable for TDNN's.** (a) Word orientation obtained by linear regression. (b) Smoothing and removal of i dots and t bars. Grey areas indicate the projection of the shade of i dots and t bars. (c) Detection of the baseline and the core height by fitting two lines with the local extrema of the y-coordinate, excluding the outliers. (d) The final representation encoding local features of the pen trajectory regularly spaced in arc length.

After the preprocessing is performed, a **Time Delay Neural Network** is used to estimate posterior probabilities of occurrence for each character given a part of the input data whose length corresponds approximately to one character. This operation is repeated with segments of the input which overlap considerably and are shifted in time, thus creating probability estimates $P(c_i|\sigma_j)$ for each character c_i given the input segment σ_j at time step j. An additional category, the ε category, is introduced to model transitions between characters. The observation probability

for ε is $P(\varepsilon|\sigma_j) = 1 - \sum_i P(c_i|\sigma_j)$ which represents the probability that there is no character in the segment σ_j.

The TDNN is a multi-layer feed-forward network with shared weights and with connections constrained to be "local" in time. Its successive layers perform successively higher-level feature extraction. There is one output unit for each character-class, and its activation level represents the score for that class. TDNNs, which were previously applied to speech recognition [39], are well suited for sequential signal processing. This allows us to use a representation which preserves the sequential nature of the data, in contrast to approaches based on pixel-map representation.

It may seem computationally expensive to perform so many classifications, most of which correspond to segments of data not aligned with a character. Thanks to the convolutional structure of the Time Delay Neural Network (TDNN) used here, most of the calculations between two consecutive overlapping segments are identical and are thus done only once.

We train the network with a "position invariant learning" method [36]. 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 c_i is computed as:

$$P(c_i) = 1 - \prod_j (1 - P(c_i|\sigma_j)) \tag{4.12}$$

which approximates the probability that character c_i appears *at least once* in the input field. This allows us to train the network with one target vector containing only informations about presence or absence of a character anywhere in the input field.

Although this target vector contains no character order information anymore, position guessing is enforced by the architecture of the TDNN and the redundancy of the data. It self-organizes the activities of the units in a competitive fashion to infer the positional information and approximates the *posterior* probabilities $P(c_i|\sigma_j)$ (see [36] for details). Therefore no manual segmentation is needed, word labels can be used directly.

The outputs of the TDNN are processed by a **Hidden Markov Model** which uses a duration model as shown in Fig. 24. The probability of remaining exactly d time steps in the same character model after entering it is given by $\rho(d) = \prod_{i=1}^{d-1} p_i(1-p_d)$. With a simple recursion formula, one can determine p_i from values of $\rho(d)$. We truncate the distribution by limiting the model to a finite number of states and add a self loop to the last state. This is equivalent to replacing the tail of the distribution by an exponential. For $\rho(d)$ we chose a Poisson distribution. The parameter of the distribution (both mean and standard deviation) is determined by estimating character durations from the training data statistically.

The observation probabilities for the HMM at each time step are given by the Neural Network and the transition probabilities between states are the parameters of the duration model. In terms of WFST, the duration model is the "output

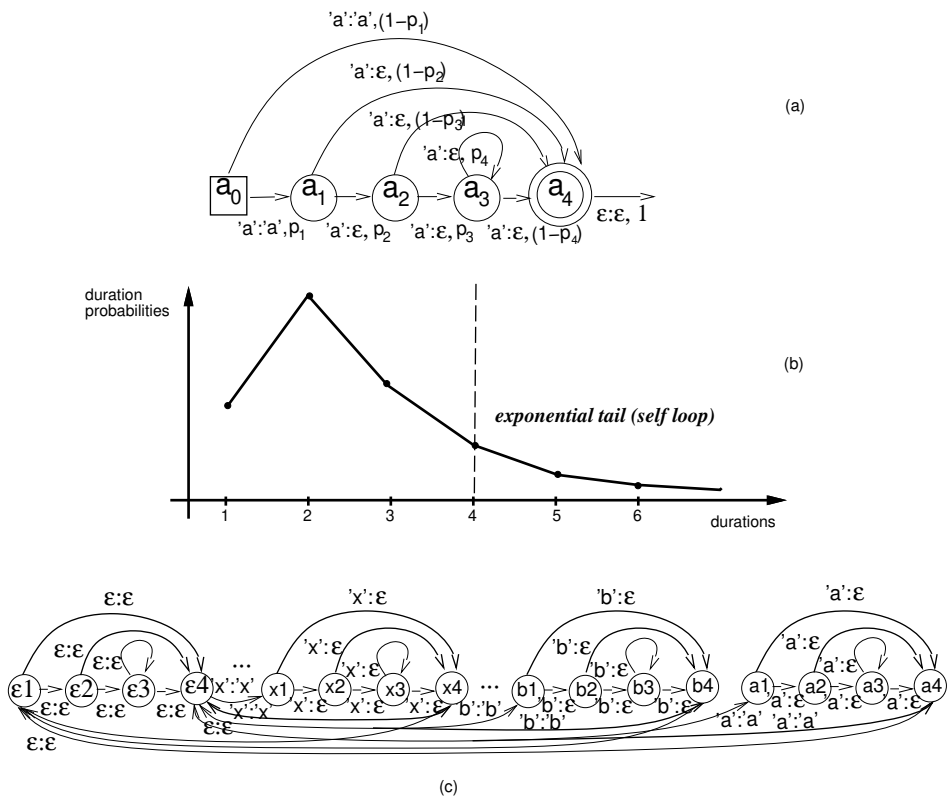

Fig. 24. **Duration model** (a) Duration model for character 'a'. The ε character is the empty character. Character 'a' is accepted and emitted at the first transition; then, characters 'a' are "eaten", that is accepted with the emission of an ε character. (b) The corresponding durations. (c) The entire duration model used as "output segmentation graph".

segmentation graph". The details of the graph products are found in Figs. 17 and 18.

The Viterbi algorithm [56] is used to find the best path in the HMM trellis and determine the best recognition and segmentation.

A natural way of introducing a **lexicon** in this process is to restrict the solution space of the HMM to words included in a given lexicon. Unfortunately this means building the product graph between the observation graph and the grammar graph, or OG product graph (see example in Fig. 20). If the grammar graph is a large lexicon tree, the OG product graph gets prohibitively large and the Viterbi algorithm gets prohibitively slow. One way of keeping calculations tractable is to narrow the lexicon search to those words which are closest in edit-distance (or "Levenhstein distance" [42]) to the most probable word obtained at stage 3, without the lexicon. With this method, the word list is typically reduced to 1 to 5 words. Since the calculation time of the edit-distance is a negligible fraction of the total recognition time, the recognition speed is increased substantially. Even more importantly, the speedup is at the expense of very little performance degradation.

In our example of Fig. 21, the words in the 25,000 word lexicon of the $UNIX^{(TM)}$ *spell* dictionary that are closest to "lliteroti" are: "alliterate", "illiterate" and "literate". They are all at an edit distance of three. For instance, "illiterate" is obtained from "lliteroti" by inserting an 'i', substituting 'o' for 'a' and 'i' for 'e'. Edit distances can be computed with the Viterbi algorithm [56]. We show in Fig. 25 the reduced grammar graph thus obtained.

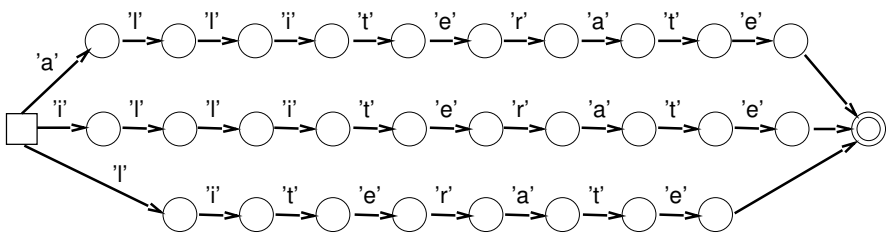

Fig. 25. **A reduced grammar graph obtained with the fastmatch technique.** This figure represents the grammar graph used for the example of Fig. 21. Out of a 25,000 word lexicon, only 3 words are closest in edit distance to the best guess without lexicon "lliteroti". This allows considerable pruning of the grammar graph, and consequently of the OG product graph.

4.2. *Recognition Performance*

Several networks were trained for uppercase, cursive, and mixed-case recognition tasks. The networks used have three to four convolutional layers and a total of 8,000 to 10,000 weights. Only the best results for the various tasks are reported here. Explanations on the parameters chosen are found in Ref. [61].

Training was done on up to 30,000 uppercase, 27,000 lowercase and 27,000 cursive words. Training took up to 60 back-propagation cycles through the training set which corresponds to two weeks on a SUN Sparc10. The system was tested on 607

words from a disjoint set of writers. The $UNIX^{(TM)}$ *spell* dictionary containing 25,000 English words was used to generate lexicons.

The best performance using a separate network for each writing style category is summarized in Table 2. Character and word error rates are reported with and without lexical constraints. Table 3 gives the results obtained for a single mixed style network.

Table 2. **Performance on the test set.** Best performance for the uppercase and cursive task. For each task a separate network was trained.

	No Lex.		*25,000 Word Lex.*	
Task	% Char.	% Word	% Char.	% Word
Upper	7.5	37.1	1.1	3.1
Cursive	26.7	75.7	11.1	20.8

Table 3. **Performance on the test set.** Best performance for the uppercase, lowercase and cursive task using a single network.

	No Lex.		*25,000 Word Lex.*	
Task	% Char.	% Word	% Char.	% Word
Upper	13.0	55.3	2.0	4.9
Lower	15.2	56.1	3.3	6.8
Cursive	32.7	82.8	13.8	24.7

Note that there is no severe loss in cursive recognition performance when using a single network for all writing styles. This shows that the approach presented is able to handle mixed style writing.

On average, only 4 words are checked when the word list is restricted to words that are closest in edit-distance; if the second closest words are also included, about 57 words are checked. Compared to the full search where all 25,000 words in the lexicon would have to be checked, a speedup factor of 400 to 6000 is gained by using the proposed fast-matcher. The recognition of an average seven character word takes about 2 seconds on a Sparc10/40.

As expected, the size of the lexicon has a big influence on the performance, as is shown in Table 4. Changing the lexicon from 1000 words to 25,000 words changes the error rate by about a factor of three.

Table 4. Performance for various lexicon sizes.

	cursive 50-3111	
Lexicon	% Char.	% Word
Unrestricted	32.7	82.8
25000 Words	11.1	20.8
1000 Words	5.7	7.3

5. Bibliographical Notes

The history of cursive on-line handwriting recognition goes back to the early 1960s. Many attempts to build recognition systems have been made; good reviews on the different approaches are given in [43, 6, 67, 53, 40].

In precursor papers around 1962 and 1972, Harmon [31, 32] uses a set of five geometrical features to classify characters with a decision tree. The segmentation is based on a rough localization of "landmarks" and an estimate of character width. The system is not trained and needs carefully written words. He reports a 40% character error rate on the specimen sentences of five persons. Mermelstein and Eden [50] propose a model for the generation of handwriting based on the concept of ballistic strokes, segmented at points of zero Y-velocity. They derive one of the first model-based recognition methods, but do not yet combine recognition and segmentation. For a vocabulary of 59 similar words they report results on 100 words written by four subjects. Their word error rate varies from 9%, when training and testing on the same writer to 31%, when testing on novel writers.

In the early 70's, a big change is introduced by realizing the importance of the segmentation being part of the recognition process. In his off-line recognition system, Sayre [59] considers each "dip" in the writing as a possible provisional segmentation point. Segments encoded with geometrical features are categorized. All possible combinations which may form an acceptable character are listed in a lexicon. The postprocessor selects the most likely word given the recognition results and letter trigram statistics. An on-line system based on similar ideas is presented by Ehrich and Koehler [18]. They report 29% word rejection rate with an error rate close to zero, using a 300 word lexicon of seven-letter words.

As larger data sets become available, more emphasis is put on statistical techniques and Neural Networks. More recently Schomaker et al. [63] and Morasso et al. [51] use stroke-based or character-based syntactic recognition methods. Their new contribution is that their feature representation is subject to training. They use an unsupervised learning technique, Kohonen's self-organizing maps [37]. They both report only writer dependent results ranging between 17 and 40% word error rate, using training data sets of 400-600 words and test sets of 100-200 words from a lexicon of 4000 words.

Another important step is the introduction of Hidden Markov Models (HMM). Traditionally used in speech recognition [56], HMMs are applied to character recognition since 1980 [52, 38, 47, 9, 8]. With HMMs, recognition-based segmentation is more rationalized: segmentation and recognition are optimized simultaneously during training, therefore it is no longer necessary to segment the data manually. Recent work of Fujisaki et al. [22] report writer dependent results by training on 1600 distinctly written characters and testing on 161 connected handwritten words. They obtain 6.8% word error rate with a 2,171 word lexicon. Ha et al. report writer independent results [30] with 4,783 training characters manually segmented from 6 writers and 728 test words from 3 writers. They obtain 26.4% word error rate with a 25,000 word lexicon.

Classical HMMs use Gaussian mixtures to estimate emission probabilities. As demonstrated in speech recognition applications, Neural Networks advantageously replace Gaussian mixtures [3, 7] because of their inherent discriminant power. Several groups [60, 70, 46, 4, 61] have demonstrated the applicability of the combination Neural Network/HMM to on-line handwriting recognition of run-on hand-printed words and cursive. The results reported in this paper are typical of the performance of such systems: ranging from 3% word error rate for uppercase printed words to 20% word error rate for cursive words, with a 25,000 word lexicon.

In an effort to reduce the dimension of input space and thus limit the number of parameters to be estimated during training, most authors perform preprocessing which makes early and irreversible decisions about segmentation and recognition [50, 18, 6, 62, 66, 58, 68, 1, 65]. The typical method consists of pre-segmenting the data at local extrema of the velocity and to encode each stroke with the coefficients of a geometric or a dynamic model of the trajectory, such as the one proposed in [33]. When large data sets are available, it is possible to rely more on the training data and to avoid such early decisions which may limit *a priori* the performance of the system. Finely sampled local and redundant feature representations [24, 2, 20, 46, 4, 61, 44] overcome this problem.

A completely different approach is taken by whole-word recognition systems which do not attempt to segment at all and take words as basic units such as [19]. As whole-word systems need a class label for each *word* to be recognized, they are limited to small lexicon sizes due to the necessary training data and computational load.

Over time, the systems have moved from being writer dependent to writer independent by increasing the number of writers in the training data. Writer independent systems are praised because they do not impose a training session to the user prior utilization. However, to boost recognition accuracy, several authors have proposed to fine tune writer independent systems to the particular style of a given writer with a small number of examples (writer adaptation) [49, 17].

Research on language models is evolving slowly. Most systems use simple lexicons, n-gram models, or a combination of those [29].[m]

The recognition rates reported in the literature are generally optimistic and difficult to interpret or compare. The tasks reported vary strongly in difficulty according to writing styles, quality of writing, data sets, lexicon sizes etc. The task chosen by the authors usually demonstrates the strength of their system. Objective benchmarks are needed to compare systems on various, well defined tasks [26].[n]

6. Conclusion and Perspectives

Only a few years ago, cursive handwriting recognition seemed out of reach. Today the dream has become reality. Yet, recognizers currently available are still disappointing to users. There is a wide margin for improvement which should

[m] For more details on this topic, see the Chapter by A. Dengel et al. in this book.
[n] See also the chapter by Guyon, Haralick, Hull and Phillips in this book.

challenge researchers and developers.

To be able to read cursive writing, humans make extensive use of contextual information. The keys to progress in that field will be to understand how to design and use elaborate language models and how to introduce adaptation to particular writer styles. The success of incorporating such contextual information in speech recognition systems is an encouragement for handwriting recognition researchers to pursue in that direction.

The classical distinction between *statistical pattern recognition* and *syntactic and structural pattern recognition* has become practically irrelevant for cursive recognition since most systems borrow techniques from both fields: they have a lot of adjustable parameters that are optimized during a training session and use *graph algorithms* to integrate all the stages of the recognition process.

The Weighted Finite State Transduction framework provides a unification to various graph-based segmentation and recognition frameworks that have been proposed in the literature. In the future, it may become a powerful tool since it allows merging various design concepts such as "input" segmentation and "output" segmentation. Having a single consistent framework facilitates the system *global optimization* during training. The modular decomposition into *recognition stages* permits to manipulate only small graphs. One can avoid building the large *product graph* of the overall system since training and search can be performed by building "on-the-fly" only the nodes that are needed.

Finally, there is often a large discrepancy between the error rate obtained in laboratory experiments and those obtained on the field. Recognizers should be tested, as far as possible, in realistic conditions of utilizations or at least on realistic test data. With projects such as UNIPEN [26], it will be possible to exchange a wide variety of data and organize public competitions.[o]

Acknowledgements

We gratefully acknowledge discussions with our colleagues at AT&T, Y. Bengio, C. Burges, Y. Le Cun, C. Nohl, and F. Pereira. We are particularly thankful to Y. Bengio, C. Burges, J. Hu and C. Nohl for their comments on this paper.

[1] A. Alimi and R. Plamondon. Performance analysis of handwriting stroke generation models. *Third International Workshop on Frontiers in Handwriting Recognition*, pages 272–283, Buffalo, 1993.

[2] E. J. Bellagarda, J. R. Bellagarda, D. Nahamoo, and K. S. Nathan. A probabilistic framework for on-line handwriting recognition. In *Third International Workshop on Frontiers in Handwriting Recognition*, pages 225–234, Buffalo, May 1993. IAPR.

[3] Y. Bengio, R. deMori, G. Flammia, and R. Kompe. Global optimization of a neural network-hidden Markov model hybrid. *IEEE Transactions on Neural Networks*, 3(2):252–259, 1992.

[4] Y. Bengio, Y. LeCun, C. Nohl, and C. Burges. LeRec: A NN/HMM Hybrid for On-Line Handwriting Recognition. *Neural Computation*, 7(6):1289–1303, 1995.

[o] See also the chapter by Guyon, Haralick, Hull and Phillips in this book.

[5] Y. Bengio and P. Frasconi. An Input/Output HMM Architecture. Im G. Tesauro, D.S. Touretzky, and T.K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 427–434. MIT Press, Cambridge, MA, 1995.

[6] M. Berthod. On-line analysis of cursive writing. In C. Y. Suen and R. De Mori, editors, *Computer Analysis and Perception: Vol. 1 – Visual Signals*, pages 55–81. CRC Press, 1982.

[7] H. Bourlard and N. Morgan. *Connectionist Speech Recognition.* Kluwer Academic, Boston, MA, 1993.

[8] 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.

[9] J. Camillerapp, G. Lorette, G. Menier, H. Ouladj, and J-C. Pettier. On-line and off-line methods for cursive script recognition. In S. Impedovo and J-C. Simon, editors, *From Pixels to Features III*, pages 273–287. North-Holland, 1992.

[10] E. Charniak. Statistical Language Learning. MIT Press, Cambridge, MA, 1993.

[11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. Introduction to algorithms. *MIT Press.* Cambridge MA, 1992.

[12] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal statistical Society Series B*, 39:1–38.

[13] J. Denker and C. J. C. Burges. Image Segmentation and Recognition. In *The Mathematics of Generalization: Proceedings of the SFI/CNLS Workshop on Formal Approaches to Supervised Learning.* Addison Wesley, 1994.

[14] D. S. Doermann and A. Rosenfeld. Recovery of temporal information from static images of handwriting. In *Proceedings of Conference on Computer Vision and Pattern Recogntition CVPR*, pages 162–168, Champaign, IL, 1992.

[15] H. Drucker, R. Shapire, and P. Simard. Boosting performance in neural networks. In I. Guyon and P.S.P. Wang, *Advances in pattern recognition systems using neural network technologies.* pages 61–75. World Scientific, Singapore, 1993.

[16] R.O. Duda and P.E. Hart. *Pattern Classification And Scene Analysis.* Wiley and Son, 1973.

[17] L. Duneau and B. Dorizzi. On-line cursive script recognition: a system that adapts to an unknown user. In *Proceedings of $12^{t}h$ International Conference on Pattern Recognition*, pages 24–28, Jerusalem, October 1994. IAPR. IEEE.

[18] R. W. Ehrich and K. J. Koehler. Experiments in the contextual recognition of cursive script. *IEEE Transactions on Computers*, 24(2):182–194, 1975.

[19] R. F. H. Farag. Word-level recognition of cursive script. *IEEE Transactions on Computers*, 28(2):172–175, 1979.

[20] N. S. Flann and S. Shekhar. Recognizing on-line cursive handwriting using a mixture of cooperating pyramid-style neural networks. In *World Congres on Neural Networks*, 1993.

[21] Advances in speech signal processing. S. Furui and M. M. Sondhi eds. *Marcel Dekker.* New York, 1992.

[22] T. Fujisaki, K. Nathan, W. Cho, and H. Beigi. On-line unconstrained handwriting recognition by a probabilistic method. In *Third International Workshop on Frontiers in Handwriting Recognition*, pages 235–241, Buffalo, May 1993. IAPR.

[23] David Goldberg and Cate Richardson. Touch-typing with a stylus. In *Proceedings of INTERCHI'93*, pages 80–87, Amsterdam, 1993.

[24] I. Guyon, P. Albrecht, Y. Le Cun, J. Denker, and W. Hubbard. Design of a neural network character recognizer for a touch terminal. *Pattern Recognition*, 24(2):105–119, 1991.

[25] I. Guyon, N. Matic, and V. Vapnik. Discovering informative patterns and data clean-

ing. In *AAAI workshop on Knowledge Discovery in Databases, KDD'94* pages 145-156, Seattle, WA, July 1994. To appear also as a book chapter in KDD-II, P. Smyth ed., *MIT Press*, 1995.

[26] I. Guyon, L. Schomaker, R. Plamondon, M. Liberman, and S. Janet. UNIPEN project of on-line data exchange and recognizer benchmarks. In *12th International Conference on Pattern Recognition*, Jerusalem, Israel, 1994.

[27] I. Guyon, J. Bromley, N. Matić, M. Schenkel, and H. Weissman. Penacée: A Neural Network System for Recognizing On-line Handwriting. In *Models of Neural Networks*, volume 3, E. Domany, et al. eds. *Springer Verlag*, New York", 1994.

[28] I. Guyon and C. Warwick. Handwriting as computer interface. In *Joint EC-US Survey of the State of the Art in Human Language Technology*. NSF, 1995 (to appear).

[29] I. Guyon and F. Pereira. Design of a Linguistic Postprocessor using Variable Memory Length Markov Models. In *Proceedings ICDAR'95*, pages 454–457, Montreal, Canada, August 1995. IEEE Computer Society Press.

[30] J. Y. Ha, S. C. Oh, J. H. Kim, and Y. B. Kwon. Unconstrained handwritten word recognition with interconnected hidden Markov models. In *Third International Workshop on Frontiers in Handwriting Recognition*, pages 455–460, Buffalo, May 1993. IAPR.

[31] L. D. Harmon. Automatic reading of cursive script. In G. L. Fisher, D. K. Pollock, B. Raddack, and M. E. Stevens, editors, *Optical Character Recognition*, pages 151–151(A), Washington, DC, 1962. Spartan.

[32] L. D. Harmon. Automatic recognition of print and script. *Proceedings of the IEEE*, 60(10):1165–1176, 1972.

[33] J.M. Hollerbach. An oscillation theory of handwriting. *Biological cybernetics*, pages 139–156, 1981.

[34] J. Hu, M. K. Brown, and W. Turin. Handwriting recognition with hidden Markov models and grammatical constraints. *Fourth International Workshop on Frontiers in Handwriting Recognition*, Taipei, December 1994.

[35] F. Jelinek. Up from trigrams! In *Proceedings of the Eurospeech Conference*, pages 1037–1040, 1991.

[36] J. Keeler, D. E. Rumelhart, and W-K. Leow. Integrated segmentation and recognition of hand-printed numerals. In R. P. Lippmann, Moody J. E., and Touretzky D. S., editors, *Advances in Neural Information Processing Systems 3*, pages 557–563, San Mateo CA, 1991. Morgan Kaufmann.

[37] T. Kohonen. *Self-Organization and Associative Memory 2nd edition*. Springer-Verlag, New York, 1984.

[38] A. Kundu and P. Bahl. Recognition of handwritten script: A hidden Markov model based approach. In *International Conference on Acoustics, Speech & Signal Processing*, pages 928–931, 1988.

[39] 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.

[40] E. Lecolinet and O. Baret. Cursive word recognition: Methods and strategies. In *Fundamentals in Handwriting Recognition*, S. Impedovo Ed., volume 124, pages 235–263. *Springer Verlag*. NATO-ASI series. 1994.

[41] A. Leroy. Lexicon reduction based on global features for on-line handwriting. Preprint, 1995.

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

[43] N. Lindgren. Machine recognition of human language, part III – Cursive script recognition. *IEEE Spectrum*, pages 104–116, May 1965.

[44] R.F. Lyon and L.S. Yaeger. On-line hand-printing recognition with neural networks. In *Fifth International Conference on Microelectronics for Neural Systems and Fuzzy Systems.* Lausanne, Switzerland, Feb. 12-14, 1996. IEEE Computer Society Press. Preprint.

[45] E. Mandler, R. Oed, and W. Doster. Experiments in on-line script recognition. In *Proc. 4th Scandinavian Conference on Image Analysis*, volume 1, pages 75–86, Champaign, IL, 1985.

[46] S. Manke and U. Bodenhausen. A Connectionist Recognizer for On-Line Cursive Handwriting Recognition. In *Proceedings ICASSP-94.* Adelaide, Australia, 1994. IEEE.

[47] 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, editor, *Advances in Neural Information Processing Systems 4*, pages 488–495, San Mateo CA, 1992. Morgan Kaufmann.

[48] N. Matić, I. Guyon, L. Bottou, J. Denker, and V. Vapnik. Computer Aided Cleaning of Large Databases for Character Recognition. In *Proceedings of 11^{t}h International Conference on Pattern Recognition*, volume II, pages 330–333. Amsterdam, The Netherlands, August 1992. IAPR. IEEE.

[49] N. Matić, I. Guyon, J. Denker, and V. Vapnik. Writer adaptation for on-line handwritten character recognition. Second International Conference on Pattern Recognition and Document Analysis, pages 187–191, Tsukuba, Japan, October 1993. IAPR. IEEE.

[50] P. Mermelstein and M. Eden. Experiments on computer recognition of connected handwritten words. *Information and Control*, 7:255, 1964.

[51] 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.

[52] R. Nag, K. H. Wong, and F. Fallside. Script recognition using hidden markov models. In *ICASSP*, pages 2071–2074, Tokyo, May 1986. IEEE.

[53] F. Nouboud and R. Plamondon, On-line recognition of handprinted characters: survey and beta tests. *Pattern Recognition*, 23(9):1031–1044, 1990.

[54] E. P. Pednault. A hidden Markov model for resolving segmentation and interpretation ambiguities in unconstrained handwriting recognition. Technical Memorandum BL0113520-930115-01TM, AT&T Bell Labs, Holmdel NJ, 1993.

[55] F. Pereira, M. Riley, and R. Sproat. Weighted rational transductions and their application to human language processing. In *Proceedings ARPA Natural Language Processing workshop.* 1994.

[56] L. R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77-2, pages 257–285, February 1989.

[57] 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*, D. E. Rumelhart et al eds., volume I, pages 318–362. *Bradford Books.* Cambridge MA, 1986.

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

[59] K. M. Sayre. Machine recognition of handwritten words: A project report. *Pattern Recognition*, 5:213–228, 1973.

[60] M. Schenkel, H. Weissman, I. Guyon, C. Nohl, and D. Henderson. Recognition-based segmentation of on-line hand-printed words. In S. J. Hanson et al., editor, *Advances in Neural Information Processing Systems 5*, pages 723–730, San Mateo CA, 1993.

Morgan Kaufmann.

[61] M. Schenkel, I. Guyon, and D. Henderson. On-line cursive script recognition using Time Delay Neural Networks and hidden Markov models. In Special issue of Machine Vision and Applications on cursive script recognition, R. Plamondon ed. *Springer Verlag*. 1995 (to appear).

[62] L. Schomaker and H.L. Teuling. A handwriting recognition system based on properties of the human motor control system In *International Workshop on Frontiers in Handwriting Recognition*, pages 195–211, Montreal, 1990.

[63] 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.

[64] P. Simard, Y. Le Cun, and J. Denker. Efficient pattern recognition using a new transformation distance. In S. Hanson et al. editors, *Advances in Neural Information Processing Systems 5*, pages 50–58, San Mateo CA, 1993. Morgan Kaufmann.

[65] Y. Singer and N. Tishby. Dynamical encoding of cursive handwriting. In *IEEE Conference on Computer Vision and Pattern Recognition*, 1993.

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

[67] C. C. Tappert, C. Y. Suen, and T. Wakahara. The state of the art in on-line handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(8):787–808, 1990.

[68] H.-L. Teuling. Invariant handwriting features useful in cursive-script recognition. Technical Report Nijmegen Institute for Cognition and Information, Nijmegen, The Netherlands, 1993.

[69] V. Vapnik Estimation of dependences based on empirical data. *Springer Verlag*. New York, 1982.

[70] H. Weissman, M. Schenkel, I. Guyon, C. Nohl, and D. Henderson. Recognition-based segmentation of on-line handprinted words: Input vs. output segmentation. *Pattern Recognition*, 27(3):405–420, 1994.

[71] G. Wilfong, F. Sinden, and L. Ruedisueli. On-line recognition of handwritten symbols. Technical Memorandum 11228-940801-06TM, AT&T Bell Laboratories, Murray Hill NJ, 1994.

[72] L. Xu, K. Adam, and C. Y. Suen. Methods of combining multiple classifiers and their applications to handwriting recognition. IEEE trans. Syst. Man and Cybernetics, 22(3):418-435, 1992.