Stroke Level HMMs for On-line Handwriting Recognition

T. Artières
LIP6, Université Paris 6, France
Thierry.Artieres@lip6.fr

P. Gallinari
LIP6, Université Paris 6, France
Patrick.Gallinari@lip6.fr

Abstract

The recent development of new terminals such as phones, mobile computers, e-books, etc, raises the needs for new interface modalities, in order to replace or complement the traditional mouse/keyboard interface. Ideally, these new interfaces should use limited computing resources and should be easy to adapt to a specific user and to a large variety of user needs. We propose here a new handwriting recognition system that is an attempt to handle these constraints. We evaluate its performances and ability to adapt to new users on a part of the Unipen database.

1. Introduction

The recent development of mobile terminals such as electronic notepads, palm pilots, electronic books, mobile phones and computers creates a need for new types of interfaces. Electronic Pen appears as a natural medium to interact with many of these devices. However, classical functionalities associated to pen interaction have to be reconsidered in the light of these new terminals and their associated usages. In this paper, we consider how Handwriting Recognition (HWR) system requirements are affected by this evolution and propose a HWR system designed for these new usages.

Let us consider the major requirements for a HWR system operating on a mobile terminal. This type device offers limited -although increasing- memory or processor resources, so that the computational demand of a HWR system should be limited. Also, these terminals being most often dedicated to a personal use, HWR systems should be conceived for an easy adaptation to the individual user demand. On one side, this could be an advantage since user adapted HWR systems may have better performance and may be built at a lower cost than writer-independent systems. On the other side, the systems must be very flexible: most of these devices are aimed at a large diffusion and should cope with very different user demands. In particular, they should be easily customizable in order to allow users to define their own specific gimmicks, abbreviations, signs or even alphabet.

Existing HWR systems hardly meet these requirements. Early commercial HWR systems were isolated character recognition systems that overcome the complexity of HWR by using a constrained input procedure that may be close or not to natural handwriting. In such systems, the user must adapt himself to the system, and it is usually not possible to define new characters. More recent systems, dealing with the “natural” handwriting signal have been developed for word recognition, they are generally based on statistical models such as Hidden Markov Models (HMMs). Most of the time they necessitate a large amount of parameters, and need large databases for training. They also require significant computing resources and, when possible, user adaptation is a tedious process. Furthermore in such systems, it is usually not possible to define new characters or signs.

The aim of this study is to explore the design of HWR systems that could satisfy the properties discussed above: computationally economic, easily customizable to different usages and user needs. In the proposed system, letters or drawings are encoded as a succession of elementary strokes from a generic finite stroke alphabet. A letter or drawing model is a stochastic automaton describing all allowed stroke sequences for the letter/drawing. Models are learned automatically from data.

Using elementary strokes is not new in handwriting. Elementary graphical primitives such as loops, crosses etc, have been used in many off-line HWR systems. Although on-line HMM-HWR systems designed for European alphabets usually operate directly on the dynamic input signal without considering specific graphical primitives, defining letters as combinations of elementary strokes has also been used for on line processing of complex characters in Chinese or Korean alphabets [2, 4, 5, 6]. Using intermediate stroke representations simplifies the design of recognizers when dealing with large graphical alphabets. Strokes are usually modeled using simple HMMs which are then combined into letter models.

In our system an input handwriting signal is processed in two steps. First, a low-level system transforms the signal...
(pen input) into an intermediate level representation which mainly consists in a sequence of elementary strokes. In a second step, character HMMs take as input this intermediate representation and output a score. The originality of the system is that after the signal has been segmented into a stroke sequence, character HMMs directly operate on strokes only [4, 6], this is probably well adapted to e.g. Kanji, but not complex enough to model signs with many curves. Strokes could also be learned directly from the data, but this did not bring any improvement in our experiments. We found the choices presented here as a good compromise.

The paper is organized as follows. We present successively the low level system (section 2) and the stroke level system (section 3). In section 4, we discuss complexity issues. We provide experimental results in section 5 and give some conclusions in section 6.

2. Low-level system

The role of this system is to decode pen signals into a sequence of elementary strokes. We used a fixed stroke alphabet of 36 elementary strokes as primitives; 12 straight lines, 12 corresponding to convex curves and 12 to concave curves (Fig. 1). This set of elementary strokes will be denoted \( ES = \{es_1, ..., es_{36}\} \). Stroke models are implemented using trajectory models [1]. These models implement time-varying functions describing the time evolution of the tangent angle of the drawing. They use a normalized time varying between 0 and 1, 0 corresponding to the first point in the stroke and 1 to the last point. More precisely, the model for \( es_i \) is the following:

\[
\begin{align*}
\begin{bmatrix}
\sin(\alpha(t)) \\
\cos(\alpha(t))
\end{bmatrix} &= f_i(t) \\
&= i = 1, 36
\end{align*}
\]

where \( f_i \) is implemented via a multi-layer perceptron (MLP).

![Figure 1: Set ES of 36 fixed elementary strokes used in the low-level system - from left to right 12 straight lines, 12 convex and 12 concave stroke models.](image)

The stroke alphabet is rich enough to represent a large collection of characters and graphical signs. The use of trajectory models allows to handle stroke length variation. The combination of stroke representation and trajectory models allows to model very closely any drawing.

Other choices could be made for this intermediate representation. Some authors make use of straight line originality of the system is that after the signal has been segmented into a stroke sequence, character HMMs directly operate on the discrete stroke level representation. This considerably reduces the model complexity and allows for an increased flexibility. This system meets many of the requirements of mobile applications and new usages. It is fast and computationally economic since much computation is done in the “stroke space”. As a consequence, it can learn from a very limited number of examples. It also allows to easily learn new symbols and to adapt to a specific user. For simplicity, we focus here on letter recognition, but the model is generic and can be adapted to any graphical input.

The stroke level system is an ergodic HMM with all transitions allowed.

The aim of the low-level is to produce a stroke level representation (SLR) of an input handwritten signal. This SLR is obtained via dynamic programming using the automaton depicted in Fig. 2. The SLR has three components. The first is a stroke sequence \( S \) with each stroke in \( S \) belonging to \( ES \). The second is the duration of each stroke in \( S \). The third, \( Sk \), characterizes the spatial inter-relations between successive strokes and consists in the spatial skips between these strokes -this point will be further detailed-. Using the low-level system, an input handwritten signal may then be represented by its SLR, i.e. as a triple \( R=(S, D, Sk) \). Fig. 3 illustrates the SLR of an input handwritten signal corresponding to letter ‘x’. We have also in this figure the “rebuilt signal” that is the signal built from the SLR information. Comparing the rebuilt signal and the original signal one can see that the SLR includes a rich information about the signal.

![Figure 2: The low-level system is an ergodic HMM with all transitions allowed.](image)

3. Stroke-level system

Using the low-level system, any handwriting may be represented by its SLR. In this section we focus on a stroke level HWR system working on this representation. As
discussed in the introduction, our main motivation is to build flexible HWR systems able to learn easily new symbols and to adapt easily to a specific user handwriting. Each letter is modeled as a stochastic automaton, defined over a set of reference SLRs. These reference SLRs are selected from the training set of SLRs.

To begin with, we define in section 3.1 stroke-level letter models. Then we show in section 3.2 how to associate to a given SLR a stroke-level HMM. Finally, we discuss the training strategy for letter models in section 3.3 and the user-adaptation algorithm in section 3.4.

3.1. Stroke level Letter Model

A character \( l \) may be written in various styles and will be represented here by a mixture model (eq. 2):

\[
P(R / l) = \sum_{i=1}^{N(l)} P(M_i(l))P(R / M_i(l))
\]

where \( N(l) \) is the number of stroke level models for \( l \), \( M_i(l) \) stands for the \( i \)-th stroke level model with prior \( P(M_i(l)) \). Models \( M_i(l) \) are HMMs operating on SLRs. \( P(R / M_i(l)) \) stands for the probability of an SLR \( R \), given model \( M_i(l) \).

3.2. SLR-based Stroke-level HMM model

In the following, we describe how to derive a stroke-level HMM from a SLR. This is first detailed for a simple SLR reduced to the stroke sequence \( S \), i.e. \( R=(S) \), we then show how to incorporate spatial and duration information into this model.

3.2.1. Basic stroke-level HMM model

Let \( R=(S) \) with \( S=(s_1, s_2, \ldots, s_N) \) be a SLR for a particular character \( (s_i \in ES, \forall i) \). \( R \) may be used to define an \( N \) states left-right HMM, \( M_b \), one state being associated to each stroke \( s_i \). Emission probabilities are defined for states in \( ES \). The probability law attached to a state corresponding to \( s_i \) models the probability that an observed stroke corresponds to \( s_i \). It is a discrete law, there are 36 possible observations. Authorized transitions are from a state to itself and to the next state, transition probabilities are assumed uniform for simplicity. The probability of a stroke sequence \( S'=(s'_1, s'_2, \ldots, s'_{N'}) \) being produced by \( M_b \), is then:

\[
P(S' / M_b) = \sum_{q_1, \ldots, q_{N'} \in Q} p(q_1) \prod_{i=1}^{N'} p(s'_i / q_i) \prod_{i=2}^{N'} p(q_i / q_{i-1})
\]

where \( q_i \) stands for a state of model \( M_b \), \( p(s' / q) \) is the probability of observing stroke \( s' \) (\( s' \in ES \)) in state \( q \).

Let \( s \in ES \), emission probabilities in a state \( q \) corresponding to \( s \) are computed as:

\[
P(s' / q) = \alpha \exp(-\frac{1}{\alpha^2} \left \| s' - s \right \|^2)
\]

where \( \left \| s' - s \right \| \) is a point to point distance between two stroke representations [1] and \( \alpha \) is a normalizing factor. Note that only \( \text{card}(ES) \), i.e. 36 different emission probability laws are used for all characters representations. The same emission probability density will then be shared by many character representations. Once learned on a representative set of letters or drawings, they could be considered stable enough to be used without retraining on any new drawing or character set.

Note that a Viterbi approximation can also be used instead of the summation in (3).

3.2.2. Spatial information

Spatial information (e.g. pen up movements) are not represented in the stroke sequence encoding. They may be important for the recognition of drawings and characters (e.g. diacrit marks). Different approaches have been proposed to handle this information, ranging from simple pen up movements models [4] to more complex modeling tuned by hand [5]. We chose to introduce the spatial information using a simple characterization of spatial skips between successive strokes. This allows this information to be integrated into the decoding process with negligible additional computational cost. The SLR we consider here are of the form \( R=(S, Sk) \) with \( S=(s_1^N) \) as before and \( Sk=(sk_2^N) \), where \( sk_i \) encodes a two-dimensional displacement (in \( x \) and \( y \) coordinates) between successive strokes of \( S \). More precisely, the displacement is between the last point of a stroke and the first point of the next stroke. As done in the previous section, for each reference representation \( R \) we define an associated HMM model \( M_b \). The core of the model (states, transitions, emission probability laws) is derived from \( R \) as in section 3.2.1. To handle spatial information, we introduce a variation of the above HMM. Since spatial skips correspond to between-strokes transitions, it seems natural that the skips should be related to transitions between states of the HMM \( M_b \). We then chose to associate spatial emissions to transitions. Considering simultaneously stroke and skip modeling, the probability of an input sequence \( R'=(S', Skip') \) with \( S'=(s_1^N) \) and \( Skips'=(skip_2^N) \) being generated by model \( M_b \) may be computed through:

\[
P(R' / M_b) = \sum_{q_1, \ldots, q_{N'} \in Q} p(q_1) \prod_{i=1}^{N'} p(s'_i / q_i) \prod_{i=2}^{N'} p(q_i / q_{i-1}) \prod_{i=2}^{N'} p(skip_i / q_i, q_{i-1})
\]
or alternatively via the Viterbi approximation by replacing the summation over state sequences in (4) by a maximization.

The underlying HMM architecture, we call a “spatial” HMM, is illustrated in Fig. 4. The dependence between the different variables, and the succession of computations is probably better illustrated by the dynamic Bayesian network representation of this model in Fig. 5. The generation of a SLR using such a model consists in iterating three steps: emit a stroke using the emission probability law of the current state; choose a transition according to transition probabilities (considered uniform here); emit a spatial skip according to the emission probability law associated to the chosen transition etc.

Figure 4: A Spatial HMM architecture.

Figure 5: Dynamic bayesian network corresponding to the spatial HMM.

Skip probability emissions are modeled as 2-dimensional gaussian laws with identity covariance matrices. Given $M_k$ associated to $R=(S, Sk)$, the mean is chosen as $sk_{i+1}$ for a transition between two successive states $s_i$ and $s_{i+1}$, for self transitions the mean value is the null vector. Then:

$$\mu(s_i,s_{i+1}) = sk_{i+1}; \mu(s_i,s_i) = 0$$

and

$$P(sk_{i+1}/q_i,q_{i-1}) = \frac{1}{C} \exp(-\beta(sk_{i+1} - \mu(q_i,q_{i-1})))$$

3.2.3 Duration information

Duration information is only indirectly taken into account into the above model since this model operates at the stroke level and not at the signal level. This information is essential for distinguishing between different drawings or letters. Absolute duration being not relevant here, we associate to each stroke its relative duration in the character model. These quantities are not used directly in the decoding process since it leads to significant computation increase, it is rather used for rescoring the SLR probability. The representation considered here is $R=(S,D,Sk)$ with $D=(d_i)$, where $S$ and $Sk$ are as before and $d_i$ is the relative duration associated to $s_i \sum d_i = 1$.

Let $R^*=(S^*,D^*,Sk^*)$ be the decoding of an unknown letter, and $M_k$ be the model associated to a representation $R$. For computing $p(R^*/M_k)$, $(S^*,Sk^*)$ is first aligned to $(S,Sk)$. Then relative duration $D^*$ are cumulated along the optimal path, resulting in a duration sequence $D^*$, each duration in $D^*$ corresponding to a state of $M_k$. Then the probability of duration sequence $D^*$ given model $M_k$ is computed as $P(D^*/D)$ after:

$$P(D^*/D) \equiv \exp(-\gamma\|D^* - D\|^2)$$

where $\gamma$ is a normalizing coefficient.

The probability of $R^*$ being emitted by $M_k$ is then:

$$P(R^*/M(R)) = \max_{q_1,\ldots,q_N} \left[ P(R^*,q_1^N/M(R))P(D^*/D) \right]$$

3.3 Learning strategy

Training a character model resumes to learning for each character a set of representative reference sequences $R$ and the associated $M_k$ statistics. For a letter $l$, we collect SLR (output by the low-level system) of each handwritten sample corresponding to $l$ in the training database. These SLR constitutes the training database for designing and learning the stroke-level HMM system. We then perform an incremental clustering over these SLR representations and select a subset of at most $N$ reference SLR for a fixed $N$. At last, reference SLR that are not enough representative are deleted, $N(l)$ may thus vary according to $l$.

Once the $N(l)$ models have been selected, priors are estimated by computing the average posterior probabilities of the $M_j(l)$ for all training letter $l$ samples. First, we compute posterior probabilities for each training sample for $l$:

$$P(M_j(l)/R) = \frac{P(R/M_j(l))}{\sum_{j=1}^{N(l)} P(R/M_j(l))}$$

Then, we compute average values for each model:

$$P(M_j(l)) = \frac{\sum_{R\in\text{Training set for } l} P(M_j(l)/R)}{\sum_{R\in\text{Training set for } l} P(M_j(l)/R)}$$

Note that parameters $\beta$ and $\gamma$ in eq. (6) and (7) are shared by every model and are determined heuristically on a validation data set.

After training, for recognition, an input signal is decoded using the low-level system, then its SLR is
processed by the stroke-level system which outputs a recognition score.

3.4. Writer adaptation

This model can easily take into account new drawings, letters or writing styles. In all cases, the signal to stroke decoding step remains unchanged, and only the stroke-level system parameters have to be modified. We used here a simple method that consists in re-estimating only new priors \( P(M_i(l)) \). Assume we want to adapt a writer-independent letter model for letter \( l \) characterized by \( N(l) \) stroke-level component models and priors \( P^{wi}(M_i(l)) \) using a set \( \{R_1,...,R_K\} \) of \( K \) training patterns of letter \( l \) from a target writer. We first compute writer-dependent priors \( P^{wd}(M_i(l)/R_k) \) using these \( K \) patterns according to the scheme described in eq. (9) and (10). Second, new priors (writer-adapted) are computed through:

\[
P^{wad}(M_i(l)) = \theta P^{wi}(M_i(l)) + (1-\theta)P^{wd}(M_i(l)) \quad (11)
\]

where parameter \( \theta \) is a mixing coefficient determined heuristically.

4. Complexity considerations

In the proposed system, there are 36 stroke models in the low-level each defined with 30 parameters. A stroke-level model is defined with approximately 10 parameters. With 20 stroke-level models per letter, each letter is then represented by 200 parameters. For isolated lowercase tasks, the overall system then requires 30x36+26x200, i.e. around 6k parameters. Furthermore, if the system has to be adapted to writers, the writer-specific parameters, which consist in only priors, represent only 20x26=520 parameters per writer. These numbers are to be compared to the number of parameters of a reference HWR system ([1]) which requires about 100k parameters.

There are two immediate conclusions to this discussion. First, the reduced size of the system comes together with a reduced computational cost. Second, since there are very few parameters in the system, it may be learned from a very limited training database, allowing to easily adapt an already existing system to a specific writer, or to learn completely new symbols from very few examples. We will see these points in the next section.

5. Experiments

5.1. Database

Experiments have been performed on the Unipen database [3] for isolated lowercase character recognition. We used training databases from directory \( lc \). The first database, denoted Train-MW (Multi-Writer Training Database) contains 30k characters from 265 writers. We also used 5 writer-databases (Train-W\(_i\), \( i=1..5 \)) containing 130 characters each (5 samples for each letter) each from a single writer. For testing, we used a set (Test-MW) of 1k characters from the same 265 writers as Train-MW, a set of 1k characters from 34 other writers (Test-WI for Writer-Independent Test Database) and 5 sets of 130 characters (Test-W\(_i\), \( i=1..5 \)) corresponding to each writer of Train-W\(_i\), \( i=1..5 \). In all cases, preprocessing consists in first normalizing the input signal, i.e. a sequence of coordinates (x,y) in a rectangular box, then in spatial re-sampling.

5.2. Experimental results

Preliminary experiments investigate the ability of our system to handle the variability of handwriting. Training is performed on Train-MW and tests on different test data. The first experiment investigates the accuracy of our system as a function of the number of stroke-level models per letter (Fig. 6), evaluation is on Test-MW. As may be seen, using 20 stroke-level models per letter allows to reach 80% accuracy and about 95% top-5 accuracy. To get a better idea of the significance of these results, Fig. 7 compares these performances to those of our reference system [1] trained on the same database, with evaluation on Test-MW and Test-WI. As may be seen again, for both test set (multi-writer and writer-independent tasks), our new system performs similarly to the reference system, this is a good result since it is much less complex (see §4).
The second series of experiments concerns the adaptation ability of the system and its ability to learn new symbols from a few samples.

This is evaluated with writer-dependent experiments, adaptation is realized with only five training samples per letter. Three systems are compared. The first system is a reference system, it is a multi-writer system trained on Train-MW that uses 20 prototypes per letter, i.e. the system is not adapted to the writer (it is noted “non adapted MW”). The second system is the same as above, but after an adaptation step to writer-specific handwriting (it is noted “adapted MW system”). This system allows to evaluate the efficiency of the adaptation algorithm. The third and last system is a writer dependent system (noted “WD”) with at most 5 prototypes per letter, it is directly trained using the specific writer database, with only 5 samples per letter. Five experiments have been performed using the five writer-specific databases. For the second and third systems, adaptation or learning are performed on Train-Wi databases. For the three systems, tests are performed on Test-Wi databases. Test results are then averaged over these five experiments. There are a few things to note about these results. First, the adaptation of a multi-writer system to a specific writer allows to significantly improve recognition results using only few writer specific data. Second, a whole system learned from scratch with very few learning samples allows to reach almost the same performance as the Multi-writer adapted system. Furthermore, it must be noted that these writer-dependent systems use only at most 5 prototypes per letter. These performances show that the system architecture allows both a very fast adaptation to a user and a very fast learning of new symbols.

Table 1: Adaptation performances. Three systems are compared, a system with 20 prototypes per letter trained on Train-MW (non adapted MW), the same system after an adaptation step to writer data (adapted MW), and a system with 5 prototypes per letter directly learned with the writer specific database (WD). Performances are evaluated and averaged over writer specific test databases.

<table>
<thead>
<tr>
<th></th>
<th>Non adapted MW</th>
<th>adapted MW</th>
<th>WD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-1</td>
<td>90.2</td>
<td>92.2</td>
<td>91.7</td>
</tr>
<tr>
<td>Top-2</td>
<td>97.9</td>
<td>98.9</td>
<td>98.3</td>
</tr>
<tr>
<td>Top-3</td>
<td>99.4</td>
<td>99.7</td>
<td>99.1</td>
</tr>
</tbody>
</table>

6. Conclusion

We have presented in this paper a new handwriting recognition system that exhibits essential properties required for new mobile terminals. The system is based on an intermediate stroke level representation of handwriting. Through experimental results, we have shown that this architecture allows to build systems that are fast and easily adaptable to a specific user. We have focused here on isolated character recognition, our current work concerns the extension of this system to word, sentence recognition and arbitrary symbols.

7. References