FAST TWO–LEVEL VITERBI SEARCH ALGORITHM FOR UNCONSTRAINED HANDWRITING RECOGNITION

Alessandro L. Koerich[*]*, Robert Sabourin*

Lab. d'Imagerie, de Vision et d'Intelligence Artificielle
École de Technologie Supérieure
Montréal, H3C 1K3, Canada

Ching Y.Suen

Centre for Pattern Recognition and Machine Intelligence
Concordia University
Montréal, H3G 1M8, Canada

ABSTRACT

This paper describes a fast two–level Viterbi search algorithm for recognizing handwritten words as a sequence of characters concatenated according to a lexicon. The algorithm is based on hidden Markov model (HMM) representations of characters and it breaks up the computation of word likelihood scores into two levels: state level and character level. This enables the reuse of likelihood scores of characters to decode all words in the lexicon, avoiding repeated computation of state sequences. Experimental results with an 85,000–word vocabulary indicate that the computational cost of an off–line handwritten word recognition system may be reduced by more than a factor of 20 while not introducing search errors.

1. INTRODUCTION

Handwriting recognition technology is steadily growing toward its maturity. Significant results have been achieved in the past few years [1–4]. While generic content text recognition seems to be a long–term goal [3, 5], some less ambitious tasks are currently investigated that address relevant problems such as recognition of postal addresses [1, 4], and bank checks [2]. Current off–line systems are capable of transcribing unconstrained handwriting with average recognition rates of 90–99%, depending on the lexicon size and experimental conditions [1, 3, 4].

One of the most common constraints of current recognition systems is that they are only capable of recognizing words that are present in a restricted vocabulary, typically comprised of 10–1,000 words [1, 3, 4]. The lexicon is a key point to the success of such recognition systems, because it is a source of linguistic knowledge that helps to disambiguate single characters by looking at the entire context. Some open vocabulary systems have also been proposed but their accuracy is still far below from those relying on limited lexicons. However, most of the research efforts in the field have been devoted to improve the accuracy of small vocabulary systems without looking at the complexity or speed.

As the technology of handwriting recognition is emerging for some small–scale problems, a fast and accurate search algorithm with low complexity is needed to support large vocabulary applications. As in many other fields, in handwriting recognition the complexity of the search algorithms is one of the main bottlenecks to build large vocabulary systems. Current approaches have to make a trade–off between the speed and the optimality of the decoding process by using heuristics methods to prune the lexicon prior to the decoding [6, 7] as well as during the search [8].

A search algorithm is an essential component of any handwriting recognition system. In HMM–based systems, given an observation sequence and a set of HMMs, a decoding algorithm is needed to search for the best state sequence, such that the overall likelihood score of the word is maximum. Current handwriting recognition systems have inherited the search techniques used in speech recognition, such as Viterbi, beam search, stack decoder, multi–pass search, etc. However, due to the nature of the signal, high–level features can be extracted from handwritten words, what yields to observation sequences that are considerably short (40–60 observations for a 13–character word). Furthermore, the models can be based on structural assumptions, and include a high number of states (≥ 10) [4]. Another important remark is that in off–line handwriting recognition the entire observation sequence that represents the word to be recognized is available before the decoding.

Based on these particular characteristics we developed a fast search strategy for HMM frameworks that deals efficiently with large vocabularies. The search algorithm breaks up the computation of word likelihood scores into two levels. At the first level, the entire sequence of observations is matched with each individual character model and the likelihood scores of all possible state sequences through the model are computed by a Viterbi algorithm. The best likelihood scores of each individual model are stored in separated arrays for further use. At the second level, the arrays representing individual characters are concatenated according

[*]Thanks to the CNPq–Brazil and the MEQ–Canada for the financial support and the SRTP–France for providing the baseline system.

to the composition of the words described in a lexicon and the overall likelihood score for the word is computed using a second Viterbi algorithm. Doing that, we avoid repeated computation of the state sequences of the character models each time they show up within a word. The details of this search strategy are presented in the following sections as well as some experimental results that confirm its optimality and speedup achieved.

2. TWO–LEVEL VITERBI ALGORITHM

The basic problem in large vocabulary handwritten word recognition is given a handwritten word to recognize represented by a sequence of observations $O = (o_1 o_2 \dots o_T)$ where T is the number of observations in the sequence, and a recognition vocabulary represented by R_V corresponding to V unique words, find the word $w \in R_V$ that best matches to the input pattern. The standard approach is to assume a simple probabilistic model of handwriting production whereby a specified word, w, produces an observation sequence O with probability $P(w, O)$ The goal is then to decode the word, based on the observation sequence, so that the decoded word has the maximum *a posteriori* (MAP) probability, i.e.:

$$\hat{w} \ni P(\hat{w}|O) = \max_{w \in R_v} P(w|O) \tag{1}$$

Using Bayes' Rule and assuming that $P(O)$ does not depend on w, and equal *a priori* probabilities of words $P(w)$, the MAP decoding rule can be approximate by:

$$\hat{w} = \arg \max_{w \in R_v} P(O|w) \tag{2}$$

The way we compute $P(O|w)$ for large vocabularies is to build statistical models for sub–word units (characters) in an HMM framework, build up word models from these sub–word models using a lexicon to describe the composition of words, and then evaluate the model probabilities via standard concatenation methods.

Considering a discrete symbol observation wit hsize M, we can define a sub–HMM by its compact notation as $\lambda = (A, B, \pi)$, where A is the state–transition probability distribution, B is the observation symbol probability distribution, and π is the initial state distribution. The number of states in the models is denoted as N. In our framework of handwriting recognition, there are several sub–word HMMs that model characters, digits and symbols and a recognition vocabulary represented by a lexicon R_V with V words where the length is L characters. So, a word model λ^w, regarded as a "super–HMM" is build by the concatenation of L sub–word HMMs, i.e.:

$$\lambda^w = \lambda_1 \oplus \lambda_2 \oplus \dots \oplus \lambda_L \tag{3}$$

Here, we have made use of the so–called maximum approximation which is also referred to as Viterbi approximation. In this maximum approximation, the search space can be described as a huge network through which the best time alignment path has to be found. The search has to be performed at two levels: at the state level and at the word level w. So, the decoding rule of Equation 2 can be rewritten as:

$$\hat{w} = \arg \max_{w \in R_V} \{\max_{Q} P(o_1 \dots o_T, q_1 \dots q_T|\lambda_1 \dots \lambda_L)\} \tag{4}$$

where q_t denotes the state at time t and $Q = (q_1 q_2 \dots q_T)$ is the best state sequence.

Conventional procedure employed in handwriting recognition is to concatenate character HMMs λ at state level to build up word HMMs λ^w that are further matched against the sequence of observations using a strict left–right Viterbi algorithm. So, in such an approach, if an individual character appears repeated within a word or within several words, its likelihood scores have to be computed again because it appears in a diferent context (different neighbor characters). So, its initialization depends now on the Viterbi score of the immediate preceding character, that is, the likelihood score and the frame where the preceding character terminates. The repeated computation may be minimized if the lexicon is represented as a tree structure. The computation of the prefixes can be done once and shared by words with similar spellings. However, the remainder of the word still have to be computed in a conventional manner.

The proposed search strategy deals with these limitations of the conventional decoding techniques and attempts to avoid repeated computation of character likelihood scores by decoupling the computation of the likelihood scores of individual character models and the computation of word likelihood scores. This approach is somewhat similar to the two–level DP matching used in speech recognition for connected word recognition [9].

2.1. First Level: Decoding of Sub–Words Models

The idea underneath the proposed search strategy lies in avoiding repeated computation of the best state sequences of sub–word HMMs. Given a sequence of observations O and K HMMs denoted by λ that model characters and digits,at the first level, we evaluate the matching between O and all λ's. In other words, we pre–compute the best state sequences between entry and exit states for all possible beginning s ending frames e separately from the context, that is, the word where the model may show up. Furthermore, we store the best state sequences and likelihood scores of all λ's between each possible pair of beginning and ending frames (s, e). For each individual character model λ the Viterbi algorithm is used to decode the entire sequence of observations O for the range of beginning frames s, $1 \leq s \leq T$, for the range of ending frames e, $1 \leq e \leq T$. Considering the

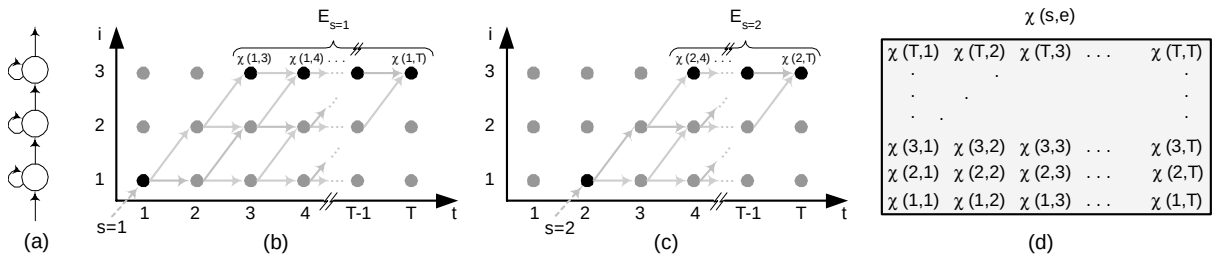


Fig. 1. (a) 3–state left–right HMM, (b) Computation of the forward probabilities for a single 3–state HMM for $s = 1$ and $3 \leq E_s \leq T$, (c) Computation of the forward probabilities for a single 3–state HMM for $s = 2$ and $4 \leq E_s \leq T$, (d) Resulting array of best likelihood scores $\chi(s, e)$ for a given character model λ_k.

3–state left–right HMM shown in Figure 1a we compute the Viterbi score between each possible pair of beginning and ending frames (s, e). The complete decoding procedure at the first level is as follows:

Initialization: For $1 \leq i \leq N$, $t = s$

$$\delta_i(t) = \pi_i b_i(o_t). \tag{5}$$

Recursion: For $s < t \leq e$, $1 \leq j \leq N$

$$\delta_j(t) = \max_{1 \leq i \leq N} [\delta_i(t-1)a_{ij}]b_j(o_t). \tag{6}$$

Termination: For $t = e$

$$\chi(s, e) = \delta_N(t). \tag{7}$$

where $\delta_j(t)$ is the Viterbi score for $s \leq t \leq e$ and $1 \leq j \leq N$, b_j defines the symbol distribution in state j, and a_{ij} is the state transition probability for going from state i to state j. The same procedure is repeated over all K HMMs. So, for a given O we end up with K arrays of best likelihood scores $\chi(s, e)$ (Figure 1d). Doing that, the likelihood scores of the character models are totally independent of the context (the word within the character may appear) and may be reused unrestrictedly to compute the likelihood scores of word models. Note that we have focused our attention to obtain the likelihood score, however, the procedure to obtain the best state sequence is equally simple.

So, we have changed our space of representation, and now characters are no longer modeled by HMMs, but by simple arrays that maps all possible entry points to output points (or beginning to ending frames). So, to recognize words, we no longer need to concatenate HMMs, but only pieces together the likelihood scores of $\chi(s, e)$ according to the description of the word in the lexicon and determine the overall accumulate likelihood score over the entire word by dynamic programming.

2.2. Second Level: Decoding of Words

To compute the likelihood score of words, we need to have a language model and the pre–computed likelihood scores of the sub–word (character) models $\chi(s, e)$. Since we use a fixed lexicon as language model, the likelihood scores of the lexicon words are given by the product of the individual character models that are concatenated to form the word. To compute the word likelihoods scores we use another Viterbi algorithm that maximizes the accumulated likelihood score over the entire word models. Since words start at well defined points, that is, at the first character $(l = 1)$ and at the first frame $(s = 1)$, we have:

Initialization: For the first character $l = 1$, $s = 1$

$$\omega_l(e) = \chi(s, e), \qquad 2 \leq e \leq L. \tag{8}$$

where $\omega_l(e)$ is the likelihood of the best path ending at frame e.

Recursion: For the remaining characters $l = 2, 3, \ldots, L$ we have:

$$\omega_l(e) = \max_{1 \leq s \leq e} [\chi(s, e)\omega_{l-1}(s-1)], \qquad s \leq e \leq L. \tag{9}$$

At the end, when characters of the word have been processed $l = L$, the likelihood score of the word will be given at $e = T$ by:

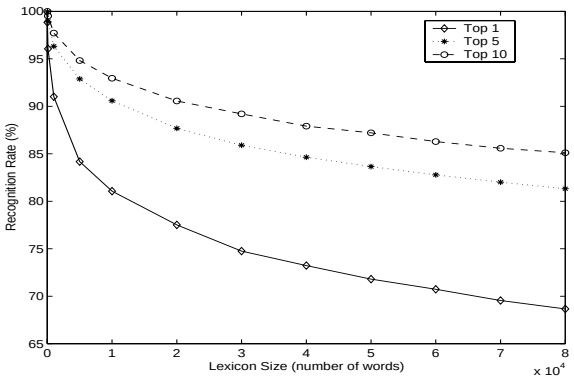
$$P(w|O) = \omega_L(T) \tag{10}$$

3. EXPERIMENTS

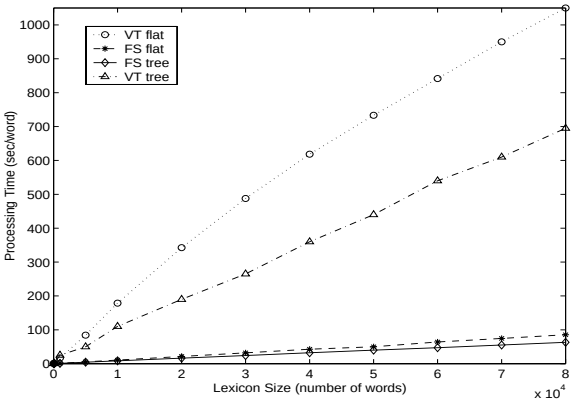
The fast two–level Viterbi decoding was applied to the recognition of unconstrained handwritten words. Sixty–two 10–state transition–based HMMs with forward and null transitions were used to model alphabetic and digits (a–z, A–Z, 0–9). The HMM models were trained using a dataset of 12,100 words and a validation set of 3,700 words. A testing dataset containing 4,674 samples was used for the final performance evaluation. The lexicon has 85,092 word representing city names. Compound words such as *Chire en Montreuil* are also present in lexicon. The average length of the words in the lexicon $(\overline{L})$ is 12 characters.

To evaluate the performance of the proposed search strategy, another version of the system was implemented, but

using a standard decoding based on Viterbi search. The optimality of the search algorithm was evaluated by comparing the overall likelihood scores obtained for each word. The likelihood scores obtained are exactly the same as those computed by the classical Viterbi. So, the two–level search strategy does not introduce search errors. Figure 2a shows the recognition rate of the system for different sizes of lexicon. However, the advantage of the fast search strategy over the classical Viterbi algorithm can be seen in Figure 2b. There is a significant reduction in the processing time that leads to a 25–speedup factor for an 80k–entry lexicon.



(a)



(b)

Fig. 2. (a) Recognition accuracy for different number of top choices and lexicon sizes, (b) Comparison of the recognition time for the conventional Viterbi algorithm with a flat (VT flat) and a lexical tree (VT tree), and the two–level Viterbi search with a flat (FS flat) and a lexical tree (FS tree), for a typical set of values: K=62 models, N=10 states, $\overline{L}$=12 characters, $\overline{T}$=40 frames.

4. CONCLUSIONS

We have presented a search algorithm that is regarded as a two–stage Viterbi search. It breaks up the decoding of words into two levels to avoid repeated computation of best state sequences for sub–HMM units. We have demonstrated that this two–level search strategy keeps the optimality of the conventional Viterbi algorithm in terms of maximum likelihood scores. The complexity of the fast two–level algorithm is approximately $\overline{T}^2 \cdot \overline{L} \cdot \overline{V}$ and that of the Viterbi is $\overline{T} \cdot N^2 \cdot \overline{L} \cdot \overline{V}$ which means that the fast two–level search strategy starts to be attractive when the length of the observation sequence is lower than the square of the number of states of the sub–word HMMs ($\overline{T} < N^2$). Different from speech recognition in off–line handwriting recognition, this situation usually comes up due to the high–level features used.

We ended up with a faster search strategy that is suitable to be used together with large vocabularies. However, there are several ways of even further reduce the computational load of the algorithm, such as beginning and global range reduction (duration).

5. REFERENCES

[1] M. Y. Chen, A. Kundu, and J. Zhou, "Off–line handwritten word recognition using a hidden markov model type stochastic network," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 5, pp. 481–496, 1994.

[2] H. Bunke, M. Roth, and E. G. Schukat-Talamazzini, "Off–line cursive handwriting recognition using hidden markov models," *Pattern Recognition*, vol. 28, no. 9, pp. 1399–1413, 1995.

[3] A. W. Senior and A. J. Robinson, "An off–line cursive handwriting recognition system," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 3, pp. 309–321, 1998.

[4] A. El-Yacoubi, M. Gilloux, R. Sabourin, and C. Y. Suen, "Unconstrained handwritten word recognition using hidden markov models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 8, pp. 752–760, 1999.

[5] G. Kim, V. Govindaraju, and S. N. Srihari, "An architecture for handwriting text recognition systems," *International Journal on Document Analysis and Recognition*, vol. 2, pp. 37–44, 1999.

[6] M. Zimmermann and J. Mao, "Lexicon reduction using key characters in cursive handwritten words," *Pattern Recognition Letters*, vol. 20, pp. 1297–1304, 1999.

[7] S. Madhvanath, V. Krpasundar, and V. Govindaraju, "Syntatic methodology of pruning large lexicons in cursive script recognition," *Pattern Recognition*, vol. 34, pp. 37–46, 2001.

[8] S. Jaeger, S. Manke, J. Reichert, and A. Waibel, "Online handwriting recognition: The npen++ recognizer," *International Journal on Document Analysis and Recognition*, vol. 3, pp. 169–180, 2001.

[9] H. Sakoe, "Two–level dp–matching — a dynamic programming–based pattern matching algorithm for connected word recognition," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 27, no. 6, pp. 588–595, 1979.