Recurrent HMMs and Cursive Handwriting Recognition Graphs

Marc-Peter Schambach
Siemens AG, Germany
marc-peter.schambach@siemens.com

Abstract

Standard cursive handwriting recognition is based on a language model, mostly a lexicon of possible word hypotheses or character n-grams. The result is a list of word alternatives ranked by confidence. Present-day applications use very large language models, leading to high computational costs and reduced accuracy. For a standard HMM-based word recognition system, a new recurrent HMM approach for very fast lexicon-free recognition will be presented.

The evaluation of this model creates a "recognition graph", a compact representation of result alternatives of lexicon-free recognition. This structure is formally identical to results of single character segmentation and recognition. Thus it can be directly evaluated by interpretation algorithms following this process, and can even be merged with these results. In addition, the recognition graph is a basis for further evaluation in terms of word recognition. It allows fast evaluation of word hypotheses, easy integration of various language models like n-grams, and the efficient extraction of lexicon-free n-best result alternatives.

1. Introduction

Cursive script recognition is often associated with word list recognition, where a set of word hypotheses is ranked by similarity to the input word image. In this paper, general lexical input is regarded, and useful output is discussed.

1.1. Standard HMM word recognizer

The idea of HMM-based word recognition systems is to build word models for all hypotheses. This is done most often by concatenating character models which are built during a training phase. For each word, a rating value is calculated, which can be interpreted as similarity or probability. As recognition result, word hypotheses are ranked by this value. Bayes's formula gives the mathematical foundation for this procedure. For speed-up, word *prefix tries* are used to share calculations for all words. Further speed-up can be achieved by pruning this trie during calculation, which gives approximate results.

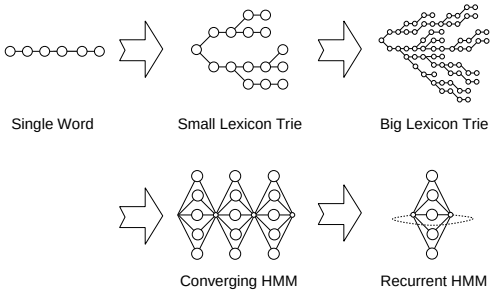

Figure 1. Various word automatons (HMMs), sorted by lexicon size.

1.2. Recurrent HMM word recognizer

This approach is extended to lexicon-free recognition accepting *all* words composed of the alphabet of trained character models. It is based on the idea that the automaton for lexicon-free recognition is much smaller compared to the prefix trie of a big lexicon (Fig. 1). The script word image is now evaluated in various steps (Fig. 2):

1. Build a *recurrent* HMM that accepts all words composed by characters of the alphabet. (Fig. 3). It is evaluated by the Viterbi algorithm which gives the *probability* of the best matching character sequence.
2. The sequence itself, the word, can be determined if the intermediate results of the Viterbi, the trellis, are kept and are traced back. This way also the character segmentation of the word can be given. These values may be used for confidence calculations.
3. If n-best results are requested, additional measures have to be taken. While backtracking, more than the best predecessor has to be taken into account. A convenient structure for further evaluation is the "recognition graph", containing all alternatives of segmentation

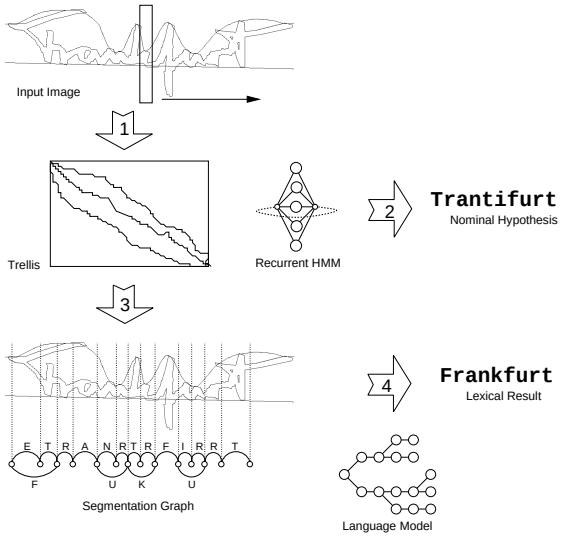

Figure 2. System overview: Calculate Viterbi on recurrent HMM (1), get nominal result by backtracking (2), create recognition graph (3), get best results with language model (4).

points and character hypotheses. Its size is limited by given maximum "probability costs".

4. For n-best recognition, search strategies like Viterbi and Dijkstra have to be applied. Lexical context can be used to limit the search space: a list of valid words (lexicon) or language models, e.g. character n-grams.

Section 2 shortly describes the baseline recognition system. Section 3 presents recurrent HMMs for lexicon-free recognition, followed by a discussion of the recognition graph in section 4. Its evaluation with the language models defined in section 5 is shown in sections 6 and 7. Experimental results in section 8 finally lead to the conclusions drawn in section 9.

2. Baseline system

Much of the system is based on techniques developed in 1993 presented in [2, 4]. It is based on Hidden Markov Models, and can process both cursive and printed script. Over time, a series of improvements has been applied to the system. The system has been successfully applied to the recognition of Latin, Cyrillic, Greek and Arabic [5] script. It is used widely in the postal domain.

Features are extracted by a sliding-window approach, extracting 20 structural features from each frame. Semi-continuous HMMs are applied, using a Gaussian classifier for vector quantization. Fig. 3 top right shows the structure

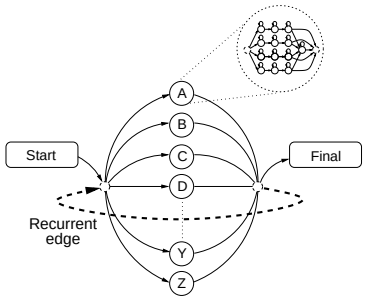

Figure 3. Recurrent HMM: Each character node is a multi-state character HMM, with each path representing an allograph.

of the character HMMs: Allographs are modeled by parallel left-to-right HMMs, followed by an optional pause state.

3. Lexicon-free recognition

If no lexicon is given, the result is called "nominal", because it delivers the pure recognition result, unbiased by lexical input.

3.1. Recurrent HMM

Usually, HMMs represent single words or lexica, constructed as prefix tries. In the proposed approach, the HMM is a *recurrent* automaton, containing all modeled characters, as shown in Fig. 3. If the alphabet is 'A' to 'Z', it is equivalent to the regular expression `[A-Z]+`. It is an extension of the converging HMM presented in [6] which models arbitrary words up to a maximum length L, with regular expression `[A-Z]{1,`L`}`. Fig. 1 shows how HMM size is further reduced by recursion while the language is extended.

3.2. Nominal recognition

Standard Viterbi on the recurrent HMM calculates the likelihood for the best matching character sequence. This result alone is useful already: In combination with "normal" word recognition, the nominal likelihood can be used for the calculation of a confidence value for rejection.

If the best nominal *word* itself is needed, the intermediate results of the Viterbi calculation, the *trellis*, has to be kept. This can be done without additional computational costs, and with memory requirements only proportional to the length of the feature sequence. When backtracking the trellis along frames — always following the best predecessor state and remembering the HMM character nodes — the best character sequence can be reconstructed, including

segmentation information. The calculation is very efficient, because no search is involved.

4. Recognition graph

In order to get a better foundation for the subsequent steps, it is useful to define a structure which contains the relevant information from the Viterbi trellis: the *recognition graph* (Fig. 2 bottom). It contains the costs of characters between arbitrary segmentation cuts, abstracting away the information about within-character states [3, 7]. It is a directed, acyclic, edge based character graph. Nodes represent segmentation cuts, while edges represent characters. The edge direction gives the left-to-right relation.

With feature extraction based on a sliding-window technique, segmentation cuts are just the boundaries between neighboring frames. With the HMM approach, costs signify "likelihood loss" in relation to the optimal state sequence.

4.1. Character costs

The first step in construction of the recognition graph is to calculate the individual character costs $C(c, f_{\text{start}}, f_{\text{end}})$ of each character c between all frame pairs f_{start} and f_{end}. The costs of characters lying on the 1-best nominal character sequence are defined to be zero: All character costs are *additional* costs; they are always positive, but zero costs are also possible for characters outside the optimal path.

Given character c and end frame f_{end}, costs for all start frames f_{start} are calculated by dynamic programming. Frames f are iterated backward from f_{end}, and an array of costs $C_f(s)$ for all character model states s is kept. The algorithm can be limited by maximum additional costs $C_{\max}$.

At $f = f_{\text{end}}$ the array is initialized with $C_f(s_{\text{final}}) = 0$ at the final state, and $C_f(s \neq s_{\text{final}}) = C_{\max}$ otherwise. For each frame, it is updated with the minimal costs for reaching the predecessor states. If entry state costs are $C_f(s_0) < C_{\max}$, they are stored as character costs $C(c, f, f_{\text{end}})$. The iteration terminates early if $\min_s C_f(s) \geq C_{\max}$.

Character costs are calculated efficiently if results are cached appropriately. The next section will show that the joint calculation of all character costs ending at a common frame f_{end} fits quite well into the construction of the recognition graph.

4.2. Graph construction

The character costs provide the set of edges for the recognition graph. The task is now to construct a *valid* recognition graph with *limited path costs*. It is valid if each character edge lies on a path from start to end node; it has limited costs if at least one path through this character has costs below a given maximum $C_{\max}$.

The graph is constructed using *all* segmentation cuts f as nodes. Iterating f from right-to-left, nodes are (de)activated and valid edges are added. This is done by updating an array of optimal costs $C(f)$ for each segmentation node f, holding the costs of the cheapest path to f_{end}. It is initialized to $C_{\max}$ for all nodes, except the final node which is 0.

At each segmentation node f, character edges for all characters c and start frames f_{new} are added if $C(c, f_{\text{new}}, f) + C(f) < C_{\max}$. The optimal node costs $C(f_{\text{new}})$ are adjusted accordingly. The iteration continues at the next node f with $C(f) < C_{\max}$. It reaches the start node f_0, because costs of the nominal best path are zero. It is easy to see that the graph has the desired properties.

The recognition graph could already be seen as the final result of script recognition, with the subsequent interpretation stages evaluating the graph. This is what is done in single character recognition, where the results of segmentation and character recognition are evaluated by interpretation stages. However, in the following sections, the typical evaluation steps are described, such providing results that are "normally" given by script word recognition.

5. Language models

The methods presented below for hypothesis evaluation and n-best nominal recognition can be adapted to the usage of generic language models. In this paper, a language model determines whether a character sequence P is a valid word or not, optionally marked with an a-priori probability. Additionally, for efficient search, also valid *partial* words are indicated, again accompanied by optional a-priori probabilities. (Example: While "wor" is no valid English word, it is a valid partial character sequence.)

The two types of language models that are used most frequently are character n-grams [1] and lexica. Experiments have been conducted with these types of language models. Other types, e.g. grammatical or morphological rules, could also be used but have not been considered here.

6. Hypothesis evaluation: Viterbi

To evaluate given lexical hypotheses, the shortest path through the recognition graph which matches the hypotheses is searched. Again, this is done efficiently by dynamic programming. In comparison to standard "time"-synchronous Viterbi algorithm (section 1.1), characters are iterated, not frames.

6.1. Single hypothesis evaluation

To get the probability and best sequence for a *single* word hypothesis, its characters are processed from left to

right. For each partial word path P, the array of optimally summed costs $C_P(f)$ for all segmentation cuts f is updated. For the complete word, costs are given finally by $C_P(f_{\max})$. For segmentation, additional backtracking along best predecessors is necessary.

It is important to note, that the same results as in "normal" direct Viterbi calculation are obtained. The difference lies in the fact that characters are processed, not frames. Thus additional search strategies can be applied easily.

6.2. Breadth-first search with lexical trie

For recognition with a language model, the same dynamic programming applies, while a prefix-trie with partial result words is built dynamically [3].

Starting with an empty word hypothesis, the trie is built and evaluated breadth first. At a time, all partial words have the same length. In each step, all valid (single) character additions according to the language model (section 5) are added. For each new partial word P, the array of optimally summed costs $C_P(f)$ is calculated like in the last section. In case P is a valid word completion, it is stored with costs $C_P(f_{\max})$.

In each iteration, only additions P with minimal costs $\min_f C_P(f) < C_{\max}$ are kept. Calculation costs can be reduced by limiting the number of active word hypotheses. Calculation terminates if no more hypotheses are available.

7. N-best evaluation: Dijkstra

For extraction of n-best results, Dijkstra can be applied as alternative search strategy. Costs in the recognition graph are normalized, only "additional" costs are stored. In speech recognition literature, Dijkstra (or A*) is often referred to as "stack decoder", with the stack representing the priority queue.

Two search variants have been tested. They differ in search space, i.e. which elements are in the priority queue:

Hypothesis & frame: The best sequences of characters with their corresponding positions are searched: Pairs $\{P, f\}$ of (partial) word hypotheses P and segmentation nodes f are stored in the priority queue, sorted by summed costs $C(P, f)$.

Hypothesis only: Search space can be reduced significantly, if only best character sequences are searched: The (partial) word hypothesis P is stored in the priority queue, accompanied by costs $C(P, f_i)$ of *all* frames f_i. The search then is guided by "best" costs $\min_i C(P, f_i)$ of all frames f_i.

The second variant is quite similar to dynamic programming of section 6.1, but using an alternative search strategy.

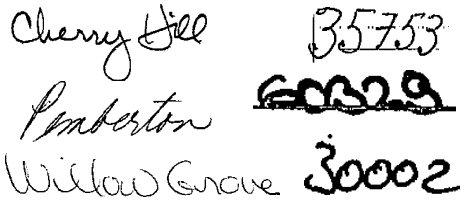

Figure 4. Examples of word images of US city names and German postal codes.

8. Experiments

For evaluation of the proposed algorithms no changes have be made to the script model and the recognition architecture itself, i.e. the classifiers and the HMM structure. Only the way of evaluating the script model has been changed. Therefore no changes in recognition *accuracy* are expected.

Experiments have been conducted to prove the correctness of the proposed algorithms, and — most important — show the computational costs of the methods. Additionally, the experiments show the new viewpoint of recognition when results from the recognition graph are evaluated.

8.1. Data

Tests have been performed on 1144 isolated word images of US city names and 580 German postal codes, taken from original mail documents. Fig. 4 shows some example images. All words are written cursively, with many connected characters — also in the case of postal code digits. Language models are based on valid word lists: 39251 US city names, including writing variants, and 27540 valid German postal codes.

HMM configurations have been trained with sets of approximately 20000 word images. Parameters that influence the absolute recognition performance (e.g. $C_{\max}$) are left unchanged. In the experiments, emphasis is placed on the relative figures.

8.2. Discussion of results

Two types of experiments have been conducted: The influence of the lexical pattern on the recognition rate, and the computational costs of the proposed algorithms. Table 1 shows the results. Test sets and configurations reflect two different recognition conditions: Postal codes are "easy", while city names are "difficult".

Recognition rates for various lexical contexts are shown in the upper part of Table 1, sorted by decreasing a-priori information. Only the single best result is counted, and no

result is rejected. It is not surprising, that a-priori knowledge and recognition rates are highly correlated (a–f). If no lexical context is given, recognition even goes down to zero in the city test case (f). This shows the importance of the recognition graph output in case no lexical information is available. While it reduces the set of possibilities considerably, it still keeps the correct result in most cases (g). This is a good starting point for following interpretation stages.

The basis of computational costs is given in the first two lines of the second section in Table 1. While a single hypothesis is valuated quickly (1), the evaluation of the complete lexicon takes a multiple of time (2). If a-priori word probabilities are given, ordered hypothesis valuation is preferable. It can be supported by nominal recognition. The nominal probability is evaluated by forward processing of the recurrent HMM (3); it is useful for confidence valuations and can be calculated merely at no time. Backtracking the trellis and getting the nominal results itself still is computational cheap (4,5). (It has to be noted, that for n-best nominal calculations not the complete recognition graph has to be calculated.)

N-gram recognition shows high computational costs (6) while not resulting in acceptable recognition rates. While competitive results have been reported in case of probabilistic high order n-grams [1], time limitations will forbid n-grams in most cases. A useful application could still be the improvement of results in case lexicon based recognition fails completely.

Calculation of a complete recognition graph (7) is still expensive, but it gives complete recognition information to interpretation stages without losing correct results. Drastic speed improvements are still possible by restricting acceptable costs $C_{\max}$, or if the graph can be divided safely into sub-words.

9. Conclusions

A simple recurrent HMM structure for lexicon-free handwriting evaluation, and a convenient graph structure for giving error tolerant results have been presented. There are various use cases: Nominal valuation for confidence calculation, and nominal or language model based n-best recognition and segmentation for hypothesis valuation. The important result is that the recognition graph is an abstraction from cursive script to general text recognition. Recurrent HMMs are an efficient method to create these graphs for cursive script.

The interesting part lies now in the text interpretation stages, where ambiguities inherent to cursive handwriting will be resolved. This can be facilitated by merging the graphs of standard character segmentation/recognition with cursive recognition results, making use of the strengths of both approaches. Future work aims at this direction.

		postal codes	city names
	alphabet	[0-9]	[a-Z0-9]
	lexicon size	27540	39251
a	lexicon = 100	93,6 %	93,6 %
b	lexicon = 1000	88,1 %	87,9 %
c	lexicon = all	56,7 %	69,5 %
d	5-gram	45,6 %	24,0 %
e	3-gram	29,3 %	2,7 %
f	lexicon-free	29,1 %	0,0 %
g	lexicon-free (graph)	100,0 %	91,5 %
1	hypothesis valuation	1,1 ms	12,2 ms
2	lexicon trie recognition	70,2 ms	252,7 ms
3	nominal probability	0,6 ms	11,2 ms
4	nominal recognition	1,5 ms	15,2 ms
5	10-best nominal	7,9 ms	45,0 ms
6	10-best 3-gram	83,2 ms	531,4 ms
7	complete graph	33,6 ms	232,8 ms

Table 1. Recognition rates 1-best (a-f), graph correctness (g), and computational costs for various algorithms. (Xeon 5150, 2.66 GHz)

Acknowledgements

The work presented here was partially funded by the German Federal Ministry of Economy and Technology (BMWi) under the THESEUS project.

References

[1] A. Brakensiek and G. Rigoll. A comparison of character N-grams and dictionaries used for script recognition. In *6th ICDAR*, pages 241–245, Seattle, WA, Sept. 2001.

[2] T. Caesar, J. Gloger, and E. Mandler. Preprocessing and feature extraction for a handwriting recognition system. In *2nd ICDAR*, pages 408–411, Tsukuba Science City, Japan, Oct. 1993.

[3] D. Y. Chen, J. Mao, and K. Mohiuddin. An efficient algorithm for matching a lexicon with a segmentation graph. In *5th ICDAR*, pages 543–546, Bangalore, India, Sept. 1999.

[4] A. Kaltenmeier, T. Caesar, J. Gloger, and E. Mandler. Sophisticated topology of hidden Markov models for cursive script recognition. In *2nd ICDAR*, pages 139–142, Tsukuba Science City, Japan, Oct. 1993.

[5] V. Märgner and H. E. Abed. ICDAR 2007 — Arabic handwriting recognition competition. In *9th ICDAR*, pages 1274–1278, Curitiba, Brazil, 2007.

[6] M.-P. Schambach. Fast script word recognition with very large vocabulary. In *8th ICDAR*, pages 9–13, Seoul, Korea, Aug. 2005.

[7] J. Sternby and C. Friberg. The recognition graph — language independent adaptable on-line cursive script recognition. In *8th ICDAR*, pages 14–18, Seoul, Korea, 2005.