Design of a Linguistic Postprocessor using Variable Memory Length Markov Models

Isabelle Guyon* and Fernando Pereira**
* AT&T Bell Laboratories, USA
{isabelle,pereira}@research.att.com

Abstract

We describe a linguistic postprocessor for character recognizers. The central module of our system is a trainable variable memory length Markov model (VLMM) that predicts the next character given a variable length window of past characters. The overall system is composed of several finite state automata, including the main VLMM and a proper noun VLMM. The best model reported in the literature (Brown et al *1992) achieves 1.75 bits per character on the Brown corpus. On that same corpus, our model, trained on 10 times less data, reaches 2.19 bits per character and is 200 times smaller ($\simeq$ 160,000 parameters). The model was designed for handwriting recognition applications but could also be used for other OCR problems and speech recognition.*

Introduction

Writing is primarily a means of communication between human beings who use the same language and share much of the same background knowledge. Handwriting is often distorted, but that shared knowledge can be used by readers to undo the effects of distortion. Furthermore, some characters are intrinsically ambiguous (eg. zero vs. uppercase O, one vs. uppercase I vs. lowercase l). Therefore, high character recognition accuracy for general handwriting requires a language model. The literature offers a choice of language models [1, 11] of various levels of complexity. While a language model based on detailed knowledge of linguistics and language use would in principle be preferable, acquiring and representing such detailed knowledge has proven so far computationally too difficult. Instead, simple statistical techniques have yielded the most predictive models. At present, word n-gram models ($n-1$-order Markov models) built from very large ($> 10^8$ words) corpora have the best performance, for instance, the word trigram model of Brown *et al* [5].

We present here a language model that approaches the performance of the best word n-gram models while meeting the criteria below, demanded by character recognition applications and especially on-line handwriting recognition:

i. provide the user with immediate feedback,
ii. run on small machines,
iii. allow rapid customization,
iv. have unlimited vocabulary,
v. have adjustable size.

The outline of our paper follows our design methodology:

1. split the system into independent modules,
2. train the modules,
3. adjust model capacity,
4. implement graph composition,
5. test the system.

1 Graph factorization

Most language models used in practice can be viewed as probabilistic finite-state automata (PFSA) (Figure 1). A PFSA $\mathcal{M}$ can be thought of as graph with nodes representing states and arcs representing state transitions. Each arc from a state s to a state s' is labeled by a character σ from a finite alphabet that is emitted (or accepted) by the corresponding transition, and by a real number giving the probability of moving from s to s' while emitting (accepting) σ:

$$P_{\mathcal{M}}(s', \sigma|s) = P_{\mathcal{M}}(s'|\sigma, s)\ P_{\mathcal{M}}(\sigma|s) \quad . \qquad (1)$$

For the models we are concerned with, $P_{\mathcal{M}}(s'|\sigma, s) = 1$ (deterministic automata).

Deterministic PFSAs include word lists (or lexicons), word n-grams and character n-grams. Lexicons provide the flexibility to add and remove words easily (spec. iii). Encoded as appropriate trees (tries), lexicons provide predictions for the next character before an entire word is written (spec. i). Character n-grams are Markov models of order $n-1$ which use

the statistics of n adjacent characters. Character n-gram models represent only imperfectly the dictionary information implicit in the word set of a word n-gram model. However, they have several advantages: they generalize to words not present in the training data (spec. iv), provide predictions for the next character before an complete word is scanned (spec. i) and can be more easily represented in a compact form (spec. ii). Moreover, prediction accuracy can easily be traded for model size reduction (spec. v).

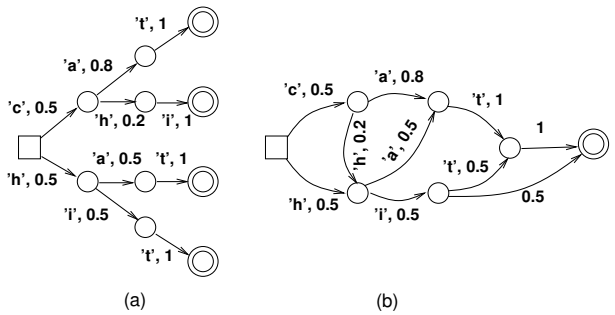

Figure 1: **Probabilistic Finite State Automata.** (a) Lexicon tree for the words "cat", "chi", "hat" and "hit". (b) n-gram model for the same words.

None of the classical language models fulfills all our requirements on its own. Therefore, we decompose our system into several independent PFSAs (graph factorization):

- main model
- capitalization model
- punctuation model
- number model
- symbol model
- proper noun model

2 Training

The modular structure of the model helps training. To train the main PFSA, we factor out capitalization, number strings, symbols and proper nouns (Figure 2). Therefore we can use standard English corpora to train the main module while allowing the the models for capitalization, numbers, symbols and proper nouns to change according to the needs of particular applications.

Lexicons and character n-grams are trained simply by calculating string frequencies. For n-grams, strings are obtained by sweeping a window of n characters over the text, with word boundaries represented by a single space character. The raw frequencies may be adjusted by some recipe to accommodate strings not seen in training [9]. For lexicons, strings are simply words.

WASHINGTON (AP) - After listening to President Reagan's televised address on channel 5, last week, Professor Eugene Jennings reached this conclusion: "He has the smarts. He's the best when he's on the ropes! He did it again".

(a)

C(C) - after listening to **C C**'s televised address on channel **N**, last week, professor **C C** reached this conclusion: 'he has the smarts. he's the best when he's on the ropes. he did it again'.

(b)

Figure 2: **Training text.** a - Text extracted from the training corpus. The words or characters that are going to be substituted by meta-characters are underlines. b - Text deprived of capitalization, with simplified punctuation, and containing the meta-characters C and N that replace proper nouns and numbers.

String statistics are efficiently gathered by growing a tree (trie) in which each node represents a string prefix w and its immediate descendants the strings $w\sigma$ for each σ in the alphabet. In training, a counter $\nu(w)$ of the node for string (prefix) w gets incremented each time a string $w' = wv$ is entered in the tree. At the end of training, the frequency of w is given by $\nu(w)/\nu(\epsilon)$, where $\nu(\epsilon)$ is the root counter. The prediction probability for the next character σ is estimated by:

$$\hat{P}(\sigma|w) = \frac{\nu(w\sigma)}{\nu(w)} \quad . \tag{2}$$

In our experiments, we limited the string window to $n = 6$.

3 Capacity tuning

The central module of our system is a variable memory length Markov model (VLMM) [2]. Like other Markov models, VLMMs predict the next character given a window of past characters. But, unlike n-grams which use a fixed window size, VLMMs optimize locally the size of the window.

Assume that w is a prefix of length $(n-1)$ used to predict the next character σ', according to $\hat{P}(\sigma'|w)$. If $\hat{P}(\sigma'|\sigma w)$ differs significantly from $\hat{P}(\sigma'|w)$, the longer prefix σw may be a better predictor than w. A possible decision criterion [2] is the Kullback-Leibler divergence between the next-character distributions for different prefixes, weighted by the prior distribution

of σw:

$$\Delta H(\sigma w, w) = \hat{P}(\sigma w) \sum_{\sigma'} \hat{P}(\sigma'|\sigma w) \log \frac{\hat{P}(\sigma'|\sigma w)}{\hat{P}(\sigma'|w)} \quad . \tag{3}$$

If $\Delta H(\sigma w, w)$ exceeds a given threshold ε, the longer prefix σw is used, otherwise the shorter prefix w is sufficient.

The threshold ε is a more fine-grained capacity control parameter than the order n of a regular Markov models (see Figure 3). For a given number of model parameters, better performance is achieved; for a given performance, fewer parameters are needed. In the figure, we have plotted the average and the standard deviation of the cross-entropy H between various validation sets and the model $\mathcal{M}$, given by:

$$H \ (per \ char) = -(1/n) \sum_{t=1}^{n} \log P_{\mathcal{M}}(\sigma_t | s_{t-1}) \quad , \tag{4}$$

where s_{t-1} is the last state of the machine and σ_t the current character in the text. The cross-entropy measures the minimum code length needed to encode the prediction errors of the model.

The validation sets used in these experiments have a total of 2 million characters and were chosen for their diversity: "Alice in Wonderland," electronic mail, Unix man pages, and various Tex files.

4 Graph composition

The overall system consists of several finite state automata, including the main VLMM and a proper noun VLMM. We use a special case of PFSA composition [3] to reconfigure flexibly the language model according to the needs of different applications, without having to retrain the core of the language model (Figure 4). For instance, the proper noun model can be replaced by a lexicon. Automata composition also allows us to use graph search techniques, including beam search, without having to build explicitly the whole composed automaton.

5 Results

The results on a 6 million character test set (the Brown corpus [4]) are summarized in Table 1.

Using cross-entropy as a performance measure allows us to make comparisons with the code lengths obtained with various coding schemes. For instance, the ASCII code requires 7 bits per character. On the Brown corpus, classical coding schemes like Huffman coding or Lempel-Ziv coding use between 4 and 4.5 bits/character. Estimates of the intrinsic entropy of English derived from human subjects range between 0.6 and 1.3 bits per character [6].

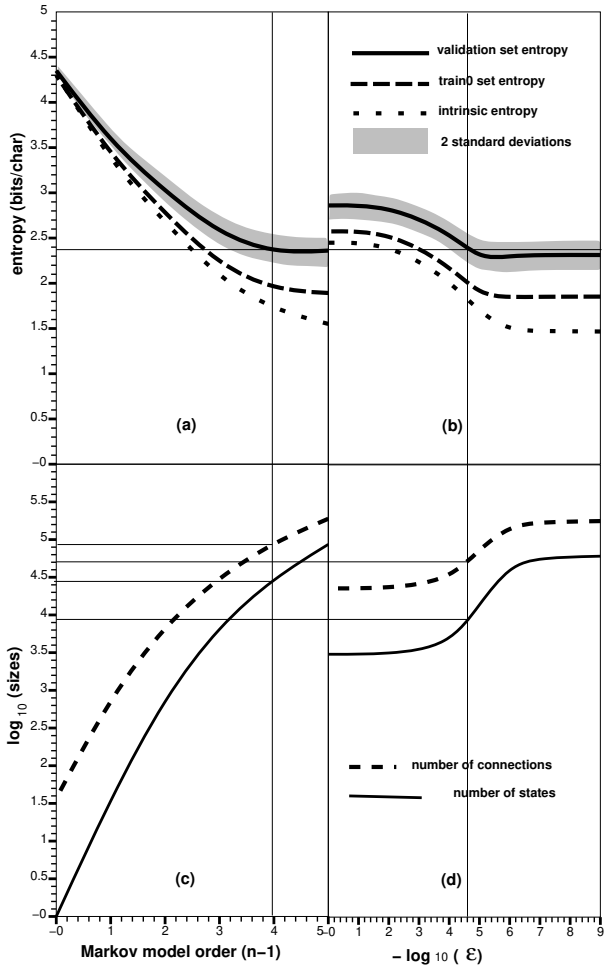

Figure 3: **Comparison of two capacity tuning methods.** a - Entropy *vs* $(n-1)$ for $\varepsilon = \varepsilon^{min} = 0$. b - Entropy *vs* ε for $n = n^{max} = 6$. c - Machine sizes *vs* $(n-1)$ for $\varepsilon = \varepsilon^{min} = 0$. d - Machine sizes *vs* ε for $n = n^{max} = 6$.

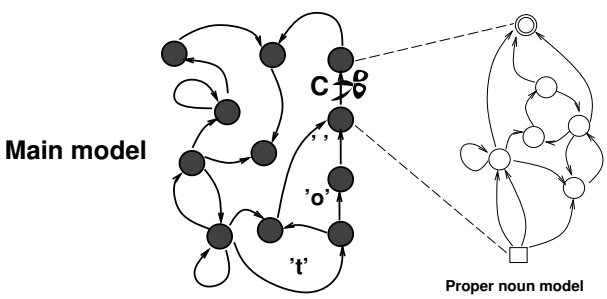

Figure 4: **Composition of two PFSAs.** The meta-character C in the main model is substituted by the proper noun model.

Model	Model freq.	Model Xentropy (bits/ch.)	Contrib. Xentropy (bits/ch.)	Number of weights
Main model	1	2.11	2.11	77,570
Capitalization	1	0.04	0.04	566
Punctuation	.026	0.66	0.017	16
Number	.0016	4.01	0.06	8,685
Symbol	.00006	5.22	0.03	7,225
Proper noun	.008	2.09	0.017	65,560
Total	-	-	2.19	159,622
Wd 3-gram [5]	-	-	1.75	$\simeq 27.10^6$
Human [10]	-	-	1	$\simeq 10^{14}$?

Table 1: **Parameters and performance of the various modules of the model.** Sub-models contribute to the total cross-entropy (Xentropy) only proportionaly to the frequency at which they are called from the main model.

We trained our model on 200 million characters of newswire, and tested it on the 6 million characters of the Brown corpus, achieving a cross-entropy of 2.19 bits/character with a state machine with approximately 160,000 parameters. The best model reported in the literature so far [5] achieves is 1.75 bits per character on the Brown corpus, using word trigrams with 27 million parameters estimated on a training set of 400 million words (over 2 billion characters). That is, our model is almost 200 times smaller and was trained on 10 times less data, and yet it approaches the performance of the much larger model.

Conclusions and Future Work

Word n-gram models have been for a while the leading method for effective language models. We demonstrated that VLMMs built at the character level are a viable alternative. They allow the construction of finite state automata of rather modest sizes, compared to word trigram models, and have good performance in terms of cross-entropy on a standard reference corpus. We control the tradeoffs among memory, speed and accuracy by varying a single information-theoretic parameter. Finite state automata composition allows us to reconfigure flexibly the language model according to the needs of different applications, without having to retrain the core of the model. Our design is targeted towards on-line handwriting recognition. Combining the model with neural-network handwriting recognizers [7, 8] shows substantial reduction in error rate (up to a factor of 2 to 3 depending on the writer). The modeling techniques described here may also be relevant to speech recognition, using phonetic units instead of characters. Other applications in discrete time series prediction may also include predicting the human genome, dynamic buffering, game playing and anomaly detection.

Acknowledgments

We gratefully acknowledge discussions with our colleagues at AT&T.

References

[1] H. Ney. Stochastic grammars and pattern recognition. In P. Laplace and R. De Mori, editors, *Speech recognition and understanding, recent advances*, volume F75 of *NATO ASI Series*, Berlin, Heidelberg, 1992. Springer Verlag.

[2] D. Ron, S. Singer, and N. Tishby. The power of amnesia. In J. Cowan et al, editor, *Advances in neural information processing systems*, volume 6. Morgan Kauffmann, 1994.

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

[4] H. Kucera and W. Francis. *Computational analysis of present day American English*. Brown University Press, Providence, RI, 1967.

[5] V. J. Brown, P. F. an Della Pietra, R. L. Mercer, S. A. Della Pietra, and J. C. Lai. An estimate of an upper bound for the entropy of English. *Computational Linguistics*, 18 (1), 1992.

[6] T. C. Bell, J. C. Cleary, and I. H. Witten. *Text Compression*. Prentice Hall, Englewood Cliffs, New Jersey, 1990.

[7] M. Schenkel, I. Guyon, and D. Henderson. On-line cursive script recognition using neural networks and hidden Markov models. In *ICASSP 1994*, Adelaide, Australia, 1994.

[8] Y. Bengio, Y. Le Cun, and D. Henderson. Globally trained handwritten word recognition using spatial representation, space displacement neural networks and hidden Markov models. In *Advances in Neural Information Processing Systems 6*. Morgan Kauffmann, 1994.

[9] S. M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(3):400–401, 1987.

[10] C. E. Shannon. Prediction and entropy of printed English. *The Bell System Technical Journal*, January 1951.

[11] F. Jelinek, R. L. Mercer, and S. Roukos. Principles of lexical language modeling for speech recognition. In S. Furui and M. M. Sondhi, eds., *Advances in Speech Signal Processing*, pages 651–699. Mercer Dekker, Inc., 1992.