LANGUAGE MODELING WITH STOCHASTIC AUTOMATA

Jianying Hu *William Turin* *Michael K. Brown*

Bell Laboratories
Murray Hill, NJ 07974, USA
{jianhu, wt, mkb}@research.att.com

ABSTRACT

It is well known that language models are effective for increasing accuracy of speech and handwriting recognizers, but large language models are often required to achieve low model perplexity (or entropy) and still have adequate language coverage. We study three efficient methods for stochastic language modeling in the context of the stochastic pattern recognition problem and give results of a comparative performance analysis. In addition we show that a method which combines two of these language modeling techniques yields even better performance than the best of the single techniques tested.

1. INTRODUCTION

Language modeling has become widely recognized as a key component of high accuracy recognizers for speech, printed text and handwriting. Recognition accuracy is influenced heavily by language model perplexity (or entropy). Over the the past two decades, language models used in recognition applications have evolved from early finite-state models to bigrams and higher order N-gram models [6]. It has been known for some time that a fixed value of N does not yield the most efficient N-gram model for most real languages. Consequently, there has been increasing interest in *variable order* N-gram models where the value of N is conditioned on the local perplexity of the language [3, 10, 9].

In this paper, we compare and study three efficient variable-order stochastic language modeling methods. Two of the methods – Variable Length Markov Models (VLMM) [4] and Variable N-gram Stochastic Automata (VNSA) [8] are the most recent developments reported in the literature. The third method, called *Refined Probabilistic Finite Automata* (RPFA), which will be introduced in this paper, is based on the highly successful text compression algorithm called Dynamic Markov Compression (DMC) [3]. We compare performances of these methods using the Brown corpus [7] and show that a combination of two of these techniques yields higher performance than any single method alone.

In the next section we fix notation and review some key concepts in stochastic language modeling. The RPFA method is introduced in Section 3 and in Section 4 we compare and contrast performance of these methods with an emphasis on model efficiency.

2. PROBABILISTIC AUTOMATA IN LANGUAGE MODELING

A Probabilistic Finite State Automaton (PFSA) is a 5-tuple $M = (S, \Sigma, \tau, \gamma, s_0)$, where S is a finite set of *states*, Σ is a finite *alphabet*, $\tau : S \times \Sigma \rightarrow S$ is the *transition function*, $\gamma : S \times \Sigma \rightarrow [0, 1]$ is the *next symbol probability function* and $s_0 \in S$ is the initial state. While a PFSA can be considered as either a language generator or acceptor, in this paper we shall call it a generator without loss of generality. A PFSA is deterministic. Suppose that symbol string $R = (r_1, r_2, \ldots, r_N)$ is generated by state sequence $\xi = q_1, q_2, \ldots, q_{N+1}$. The probability of string R according to language model M is: $P_M(R) = \prod_{i=1}^{N} P(r_i|q_i)$, where $P(r_i|q_i) = \gamma(q_i, r_i)$. A commonly used extension to PFSA is PFSA-ε — PFSA with ε *transitions*. An ε transition (also referred to as *escape transition*) is a transition that does not generate any symbol. As a result of the escape transitions, a PFSA-ε is not deterministic — different state sequences may generate the same string. Thus, the probability of a given string R becomes: $P_M(R) = \sum_{\xi \in \Xi_R} P_M(R|\xi)$, where Ξ_R is the set of all state sequences that generate string R. A more restricted subset of PFSA often used in language modeling is Finite Context Automata (FCA), where each state can be uniquely labeled by a symbol string representing the *context* (or *memory*) at this state. N-grams and variable N-grams are both examples of FCA.

A well know problem that arises in constructing a probabilistic language model is the problem of *unseen events* [11]. To accommodate all possible events (seen or unseen) in a PFSA model, a transition with a non-zero probability has to be assigned for each symbol in the alphabet at each state, resulting in a model with $|S| \times |\Sigma|$ parameters. A more efficient model can be obtained if we replace all transitions representing unseen events at each state by an escape transition. The tradeoff is that the resulting model is a PFSA-ε model and thus non-deterministic, which could potentially cause problems in recognition applications. In a recognition problem, given a sequence of observations O and a language model M, we need to find efficiently among all possible strings R the one that maximizes the joint probability $P(O|R)P_M(R)$. A popular and successful approach in speech and, more recently, handwriting recognition, is to combine linguistic knowledge with the knowledge of a specific feature domain to form a integrated recognizer (decoder) [1, 5]. A dynamic programming algorithm (such as the Viterbi algorithm) is then applied to the combined network to find the state

sequence with the highest joint probability. Let the linguistic probability estimated by the Viterbi algorithm be $P_M(R|\xi_0)$, where: $\xi_0 = argmax_{\xi \in \Xi_R} P_M(R|\xi)$. For a non-deterministic language model $P_M(R) = \sum_{\xi \in \Xi_R} P_M(R|\xi) > P_M(R|\xi_0)$, in which case the success of the above recognition scheme relies on the assumption that the maximum term is prevailing in the sum and thus: $P_M(R) \approx P_M(R|\xi_0)$.

Consider a language source L and a model M trained using sample strings from L, the performance of the language model is measured by the *model cross entropy rate* (*model entropy* for short) $H_M = -\frac{1}{n} log P_M(R)$, where R is a test string of length n. When a non-deterministic language model is evaluated, we are also interested in the entropy rate computed along the optimal state sequence: $H_M(\xi_0) = -\frac{1}{n} log P_M(R|\xi_0)$, referred to as the *best-path entropy*. The difference between H_M and $H_M(\xi_0)$ is a measure of the "degree of non-determinism" in model M. As explained above, a lower degree of non-determinism is more desirable for most recognition applications.

Training of stochastic language models consists of two steps: selecting the model structure and estimating the transition probabilities. Since in FCA models each state corresponds to a specific string (context), transition probabilities can be easily estimated as [2]: $P(r_n|r_1, r_2, \ldots, r_{n-1}) = \nu(r_1, r_2, \ldots, r_n)/\nu(r_1, r_2, \ldots, r_{n-1})$, where $\nu(r_1, r_2, \ldots, r_n)$ is the number of times string $(r_1, r_2, \ldots, r_n)$ appears in the training corpus. In one common scheme, the model structure is first reached by creating a state for each possible substring (up to a predetermined memory length) seen in the training corpus. The size of the model is then reduced through pruning of *insignificant states*. This is the scheme adapted in both VLMM [4] and VNSA [8]. A very different training scheme has been used in the DMC technique for dynamic text compression. This technique starts with a small number of states and gradually grows the structure and adjusts the transition parameters as more symbols are processed. DMC models were first introduced to handle only binary strings and proved to give better results than variable order Markov models [3]. It was later extended to handle arbitrary vocabulary size, resulting in a technique called General Dynamic Markov Compression (GDMC) [10]. In the next section we extend the GDMC technique further to create language models for recognition. The resulting models are called Refined Probabilistic Finite Automata (RPFA).

3. THE RPFA TECHNIQUE

The GDMC technique was designed specifically for dynamic text compression. As a result, the parameter estimates are not optimal: by the time a new state is created, some of the training data is already lost. In extending this technique to training language models for recognition, the dynamic requirement is dropped because for recognition purposes a language model is usually trained beforehand on a training corpus. Thus the training of a RPFA model is composed of two stages. In the first stage, called *initial training*, a model is created using a similar algorithm as that used in GDMC [10]. In the second stage, called *retraining*, the model is modified through several iterations of pruning and maximum likelihood parameter retraining. In what follows, (s_i, s_j) denotes a transition from state s_i to state s_j; $Freq(s_i, s_j)$ denotes the frequency of transition (s_i, s_j) and $Fsum(s_i)$ denotes the sum of frequencies of the transitions from s_i. The symbol generated by transition (s_i, s_j) is referred to as the label of the transition and denoted $l(s_i, s_j)$ (the label of an escape transition is ε); s_j is called the escape target of s_i if (s_i, s_j) is an escape transition.

The initial model consists of the initial state s_0 and transitions and states $s_1, s_2, ..., s_m$ for all the symbols in the alphabet. Each transition from s_0 is labeled by a symbol in the alphabet, while all transitions back to s_0 are escape transitions. The initial transition frequences are all set to 1. Initial training consists of updating the transition frequencies after receiving each symbol, and "growing" the model structure if certain conditions are met. Suppose the current state is s_i and the current symbol is r_t, if a non-escape transition (s_i, s_j) with label r_t exists, then this transition is taken and the current symbol becomes r_{t+1}, otherwise the escape transition is taken and the current symbol is unchanged. After taking a transition, its frequency count is increased by one. The model is gradually grown through two structure-modifying operations: *cloning* which creates a new state and *shortcut generation* which adds a new transition (Fig. 1). In the cloning operation, the target state s_k of a non-escape transition (s_i, s_k) is cloned if the difference $Fsum(s_k) - Freq(s_i, s_k)$ exceeds the threshold value CC (Cloning Constant). The frequencies are then set to: $Freq(s_i, s_k') = Freq(s_k', s_k) = Freq(s_i, s_k)$. In Shortcut generation, when a non-escape transition (s_k, s_m) is taken after an escape transition (s_k', s_k), a shortcut (s_k', s_m) with the same label as $l(s_k, s_m)$ is created to bypass the combination. The initial frequency of the shortcut is then set to: $Freq(s_k', s_m) = \mu_{km} Freq(s_k', s_k)$, where $\mu_{km} = Freq(s_k, s_m)/Effsum(s_k|s_k')$, and $Effsum(s_k|s_k')$ is the sum of frequences of all the transitions from state s_k to all the states that do not have direct transitions from s_k'. More detailed explanations of these two operations can be found in [10]. The combination of cloning and shortcut generation has the effect of increasing the context used to estimate the next symbol distribution. In order to control the maximum context length in the initial training, the string is partitioned into overlapping substrings of a predetermined length N, which are scanned one after another. The current state is reset to the initial state at the beginning of each substring. The resulting model roughly corresponds to a variable N-gram model, or a variable length Markov model of memory length $N-1$.

Model retraining is carried out through iterations of the following two steps: structure pruning to remove insignificant states and maximum likelihood parameter retraining on the new structure. Pruning is carried out through two different operations. In the first operation, state s_i is removed along with all the transitions to s_i if its total frequency count $Freq(s_i)$ is lower than a threshold. In the second operation, state s_i is considered insignificant if $Rsum(s_i)/Fsum(s_i) < \epsilon_2$, where $Rsum(s_i)$ is the sum of the frequency of all non-escape transitions from s_i. In this case state s_i is removed, and all transitions to s_i are redirected to the escape target of s_i (notice that they are not removed as in the previous case). After pruning, all transition frequency counters are reset to 1. The training string is then rescanned and frequencies are accumulated in the same manner as in initial training. The probability of a transition (s_i, s_j) is then assigned: $Freq(s_i, s_j)/Fsum(s_i)$.

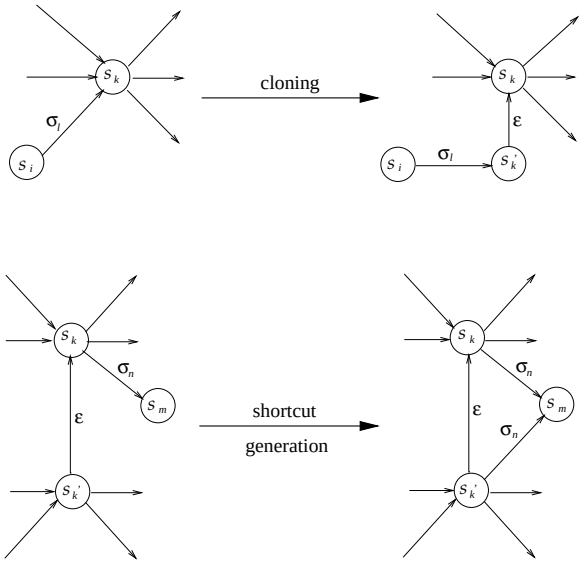

Figure 1: Cloning and short cut generation.

4. PERFORMANCE ANALYSIS AND IMPROVEMENT

First we give a brief overview of VNSA and VLMM to facilitate the comparison. In VNSA [8], the model structure is first reached by creating a state for each possible substring (up to length $N-1$) seen in the training corpus. To handle unseen events, an escape transition is created for each state. A special type of states called *null states* are created to serve as escape targets allowing elaborate estimation of separate back-off probabilities. Various heuristic interpolation methods are used to estimate escape probabilities as well as the transition probabilities from null states. Pruning is based on a single criterion — states with low frequency are removed. After pruning, parameters are not retrained, but re-estimated by redistributing previous counts using certain heuristics. In VLMM [4], a *prefix tree* is first created where each node corresponds to a substring (up to length $N-1$). The transition frequencies are counted by traversing the tree structure repeatedly with overlapping substrings of length N. The tree is first pruned by eliminating nodes with low frequency. Then another pruning procedure is applied while the prefix tree is converted to a *suffix tree* — for each node n_p in the prefix tree, a corresponding node n_s in the suffix is created if and only if the Kullback-Leibler divergence (KL divergence) between the probability distribution at n_p and the probability distribution at its ancestor node in the prefix tree is larger than a threshold. The suffix tree is then converted to an automaton. An escape transition is set from each state to the initial state and escape transition probabilities are estimated using simple Lagrange interpolation [4]. No retraining is carried out on the automaton after it is created.

The Brown corpus [7] is used to compare the performance of the above two techniques and the RPFA technique introduced in the previous section. To simplify the task only lower-case letters in the English alphabet and space are used, all upper-case letters are converted to lower-case and all other symbols are removed. After preprocessing, the corpus is divided into a training set containing 5,419,167 characters and a test set of 170,369 characters. Both model entropy and best-path entropy are computed as performance measures. The size of a model is measured by the number of transition probabilities (parameters).

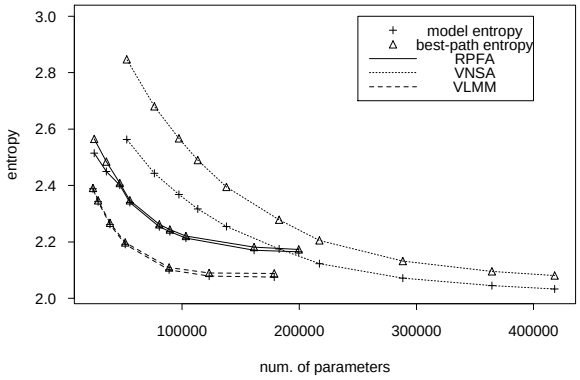

Figure 2: Performance plot for models with maximum memory length 4.

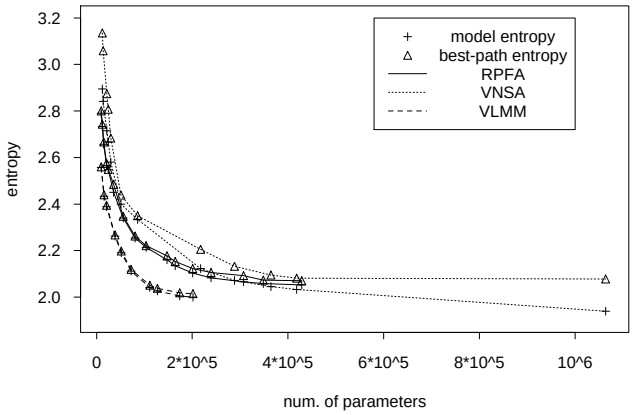

Figure 3: Overall performance comparison of VLMM, VNSA and RPFA.

Fig. 2 shows the performance plots of models created by the three techniques when the maximum memory length is set to 4, which means the models are all variable 5-grams. The following observations are made from the plots. 1) Both VLMM and RPFA models have very small degree of non-determinism; while the degree of non-determinism of the VNSA models is much larger. 2) The more complicated mechanism (with special null states) for back-off probability estimation used in VNSA does not seem to pay-off; with the simplest escape transitions, VLMM is able to achieve comparable entropy rates with much smaller model size. 3) The model growing strategy which works well in GDMC for dynamic text compression does not provide as good an initial model structure when applied in RPFA, as demonstrated by its inferior asymptotic performance. 4) VNSA has good asymptotic performance, however it degrades fast

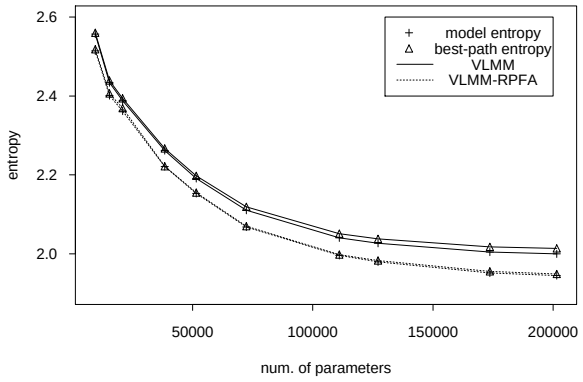

Figure 4: Performance comparison of VLMM and combined VLMM-RPFA.

under pruning. In comparison, the performance of the VLMM and RPFA models drops much slower as the number of parameters decreases. This shows that the KL distance measure used in VLMM is a more effective measure of the significance of a state than the simple frequency count. It also indicates that the parameter retraining procedure used in RPFA helps to reduce the performance loss after pruning. Fig. 3 shows the overall performance comparison of the three techniques. Here the maximum memory length, or order of the model, is not fixed, but rather used as an extra capacity control parameter and allowed to vary from 3 to 6. For each technique, at any given model size, the model which achieves the lowest entropy rates is chosen and the corresponding entropy rates are shown in the plot. Overall, VLMM has the best performance in both measures.

While VLMM proves to be the best modeling technique out of the three examined in this paper, we found there is still room for further improvement. As mentioned before, the parameters in a VLMM model are not retrained after pruning, as a result they are not optimal under the reduced model structure. To improve these parameters, we combine the VLMM technique and the RFSA technique by applying the maximum likelihood parameter retraining procedure described in section 3 to the model structure obtained using the VLMM technique. The results are plotted in Fig. 4 in comparison to the VLMM models. As shown clearly by the plots, the combined VLMM-RPFA technique gives a better performance as expected.

5. CONCLUSION

Three methods known as the Variable Length Markov Model (VLMM), the Variable N-gram Stochastic Automata (VNSA), and the Refined Probabilistic Finite Automata (RPFA) have been critically compared in terms of language modeling efficiency. Since all three methods generate non-deterministic models to achieve low model complexity, two estimates of entropy are used in performance evaluation: the model entropy which is the expected value of entropy taken over all possible model interpretations; and the best-path entropy which is the entropy of the most likely interpretation. The difference between these measures indicates the degree of non-determinism. Since many practical recognizers use only the most likely interpretation, it is desirable to have a low degree of non-determinism in the language model. The comparisons based on the Brown corpus indicate that the VLMM technique is superior to both RPFA and VNSA techniques, while a combination of the VLMM and RPFA methods simultaneously exhibits both the lowest ambiguity and the highest modeling efficiency among all of the methods tested over a wide range of model size and entropy values.

6. ACKNOWLEDGEMENTS

The authors would like to thank Giuseppe Riccardi and Isabelle Guyon for providing the the VNSA and VLMM programs as well as many helpful suggestions.

7. REFERENCES

1. L. R. Bahl and F. Jelinek. Maximum likehood approach to continuous speech recognition. *IEEE Trans. PAMI*, PAMI-5(3):250–256, March 1983.
2. P. Billinsley. Statistical methods in Markov chains. *Ann. Inst. Statis. Math.*, 32:12–40, 1961.
3. G. V. Cormack and R. N. S. Horspool. Data compression using dynamic Markov modelling. *The Computer Journal*, 30:541–550, 1987.
4. I. Guyon and F. Pereira. Design of a linguistic postprocessor using variable memory length Markov models. In *Proc. 3rd ICDAR*, pages 454–457, Montreal, Canada, August 1995.
5. J. Hu, M. K. Brown, and W. Turin. Handwriting recognition with hidden Markov models and grammatical constraints. In *Proc. 4th IWFHR*, pages 195–205, Taipei, Taiwan, December 1994.
6. F. Jelinek, R. L. Mercer, and S. Roukos. Principles of lexical language modeling for speech recognition. In S. Furui and M. M. Sondhi, editors, *Advances in Speech Signal Processing*, pages 651–699. Mercer Dekker Inc., 1992.
7. H. Kucera and W. Francis. *Computational analysis of present day American English*. Brown University Press, Providence, RI, 1967.
8. G. Riccardi, E. Bocchieri, and R. Pieraccini. Non-deterministic stochastic language models for speech recognition. In *Proc. ICASSP'95*, pages 237–240, Detroit, May 1995.
9. 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.
10. J. Teuhola and T. Raita. Application of a finite-state model to text compression. *The Computer Journal*, 36:607–614, 1993.
11. I. H. Witten and T. C. Bell. The zero frequence problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Trans. on Information Theory*, 37(4):1085–1093, 1991.