Beyond Word N-Grams

Fernando C. Pereira
AT&T Bell Laboratories
600 Mountain Ave.
Murray Hill, NJ 07974
pereira@research.att.com

Yoram Singer
Institute Computer Science
Hebrew University
Jerusalem 91904, Israel
singer@cs.huji.ac.il

Naftali Tishby
Institute of Computer Science
Hebrew University
Jerusalem 91904, Israel
tishby@cs.huji.ac.il

Abstract

We describe, analyze, and experimentally evaluate a new probabilistic model for word-sequence prediction in natural languages, based on *prediction suffix trees* (PSTs). By using efficient data structures, we extend the notion of PST to unbounded vocabularies. We also show how to use a Bayesian approach based on recursive priors over all possible PSTs to efficiently maintain tree mixtures. These mixtures have provably and practically better performance than almost any single model. Finally, we evaluate the model on several corpora. The low perplexity achieved by relatively small PST mixture models suggests that they may be an advantageous alternative, both theoretically and practically, to the widely used n-gram models.

1 Introduction

Finite-state methods for the statistical prediction of word sequences in natural language have had an important role in language processing research since Markov's and Shannon's pioneering investigations (C.E. Shannon, 1951). While it has always been clear that natural texts are not Markov processes of any finite order (Good, 1969), because of very long range correlations between words in a text such as those arising from subject matter, low-order alphabetic n-gram models have been used very effectively for such tasks as statistical language identification and spelling correction, and low-order word n-gram models have been the tool of choice for language modeling in speech recognition. However, low-order n-gram models fail to capture even relatively local dependencies that exceed model order, for instance those created by long but frequent compound names or technical terms. Unfortunately, extending model order to accommodate those longer dependencies is not practical, since the size of n-gram models is in principle exponential on the order of the model.

Recently, several methods have been proposed (Ron et al., 1994; Willems et al., 1994) that are able to model longer-range regularities over small alphabets while avoiding the size explosion caused by model order. In those models, the length of contexts used to predict particular symbols is adaptively extended as long as the extension improves prediction above a given threshold. The key ingredient of the model construction is the *prediction suffix tree* (PST), whose nodes represent suffixes of past input and specify a predictive distribution over possible successors of the suffix. It was shown in (Ron et al., 1994) that under realistic conditions a PST is equivalent to a Markov

process of variable order and can be represented efficiently by a probabilistic finite-state automaton. For the purposes of this paper, however, we will use PSTs as our starting point.

The problem of sequence prediction appears more difficult when the sequence elements are *words* rather than characters from a small fixed alphabet. The set of words is in principle unbounded, since in natural language there is always a nonzero probability of encountering a word never seen before. One of the goals of this work is to describe algorithmic and data-structure changes that support the construction of PSTs over unbounded vocabularies. We also extend PSTs with a wildcard symbol that can match against any input word, thus allowing the model to capture statistical dependencies between words separated by a fixed number of irrelevant words.

An even more fundamental new feature of the present derivation is the ability to work with a *mixture* of PSTs. Here we adopted two important ideas from machine learning and information theory. The first is the fact that a mixture over an ensemble of experts (models), when the mixture weights are properly selected, performs better than almost any individual member of that ensemble (DeSantis et al., 1988; Cesa-Bianchi et al., 1993). The second idea is that within a Bayesian framework the sum over exponentially many trees can be computed efficiently using a recursive structure of the tree, as was recently shown by Willems et al. (1994). Here we apply these ideas and demonstrate that the mixture, which can be computed as almost as easily as a single PST, performs better than the most likely (maximum aposteriori — MAP) PST.

One of the most important features of the present algorithm that it can work in a fully online (adaptive) mode. Specifically, updates to the model structure and statistical quantities can be performed adaptively in a single pass over the data. For each new word, frequency counts, mixture weights and likelihoods associated with each relevant node are appropriately updated. There is not much difference in learning performance between the online and batch modes, as we will see. The online mode seems much more suitable for adaptive language modeling over longer test corpora, for instance in dictation or translation, while the batch algorithm can be used in the traditional manner of n-gram models in sentence recognition and analysis.

From an information-theoretic perspective, prediction is dual to compression and statistical modeling. In the coding-theoretic interpretation of the Bayesian framework, the assignment of priors to novel events is rather delicate. This question is especially important when dealing with a statistically open source such as natural language. In this work we had to deal with two sets of priors. The first set defines a prior probability distribution over *all* possible PSTs in a recursive manner, and is intuitively plausible in relation to the statistical self-similarity of the tree. The second set of priors deals with novel events (words observed for the first time) by assuming a scalable probability of observing a new word at each node. For the novel event priors we used a simple variant of the Good-Turing method, which could be easily implemented online with our data structure. It turns out that the final performance is not terribly sensitive to particular assumptions on priors.

Our successful application of mixture PSTs for word-sequence prediction and modeling make them a valuable approach to language modeling in speech recognition, machine translation and similar applications. Nevertheless, these models still fail to represent explicitly grammatical structure and semantic relationships, even though progress has been made in other work on their statistical modeling. We plan to investigate how the present work may be usefully combined with models of those phenomena, especially local finite-state syntactic models and distributional models of

semantic relations.

In the next sections we present PSTs and the data structure for the word prediction problem. We then describe and briefly analyze the learning algorithm. We also discuss several implementation issues. We conclude with a preliminary evaluation of various aspects of the model on several English corpora.

2 Prediction Suffix Trees over Unbounded Sets

Let $U \subseteq \Sigma^\star$ be a set of *words* over the finite alphabet Σ, which represents here the set of actual and future words of a natural language. A *prediction suffix tree* (PST) T over U is a finite tree with nodes labeled by distinct elements of $U^\star$ such that the root is labeled by the empty sequence ϵ, and if s is a son of s' and s' is labeled by $a \in U^\star$ then s is labeled by wa for some $w \in U$. Therefore, in practice it is enough to associate each non-root node with the first word in its label, and the full label of any node can be reconstructed by following the path from the node to the root. In what follows, we will often identify a PST node with its label.

Each PST node s is has a corresponding *prediction function* $\gamma_s : U' \to [0,1]$ where $U' \subset U \cup \{\phi\}$ and ϕ represents a *novel* event, that is the occurrence of a word not seen before in the context represented by s. The value of γ_s is the *next-word probability* function for the given context s. A PST T can be used to generate a stream of words, or to compute prefix probabilities over a given stream. Given a prefix $w_1 \cdots w_k$ generated so far, the context (node) used for prediction is found by starting from the root of the tree and taking branches corresponding to $w_k, w_{k-1}, \ldots$ until a leaf is reached or the next son does not exist in the tree. Consider for example the PST shown in Figure 1, where some of the values of γ_s are:

$$\gamma_{\text{`and the first'}}(\texttt{world}) = 0.1, \quad \gamma_{\text{`and the first'}}(\texttt{time}) = 0.6 \ ,$$
$$\gamma_{\text{`and the first'}}(\texttt{boy}) = 0.2 \ , \quad \gamma_{\text{`and the first'}}(\phi) = 0.1 \ .$$

When observing the text '`... long ago and the first`', the matching path from the root ends at the node '`and the first`'. Then we predict that the next word is `time` with probability 0.6 and some other word not seen in this context with probability 0.1. The prediction probability distribution γ_s is estimated from empirical counts. Therefore, at each node we keep a data structure to track of the number of times each word appeared in that context.

A *wildcard symbol*, '`*`', is available in node labels to allow a particular word position to be ignored in prediction. For example, the text '`... but this was`' is matched by the node label '`this *`', which ignores the most recently read word '`was`'. Wildcards allow us to model conditional dependencies of general form $P(x_t | x_{t-i_1}, x_{t-i_2}, \ldots, x_{t-i_L})$ in which the indices $i_1 < i_2 < \ldots < i_L$ are not necessarily consecutive.

We denote by $C_T(w_1 \cdots w_n) = w_{n-k} \cdots w_n = s$ the context (and hence a corresponding node in the tree) used for predicting the word w_{n+1} with a given PST T. Wildcards provide a useful capability in language modeling since syntactic structure may make a word strongly dependent on another a few words back but not on the words in between.

One can easily verify that every standard n-gram model can be represented by a PST, but the opposite is not true. A trigram model, for instance, is a PST of depth two, where the leaves are

all the observed bigrams of words. The prediction function at each node is the trigram conditional probability of observing a word given the two preceding words.

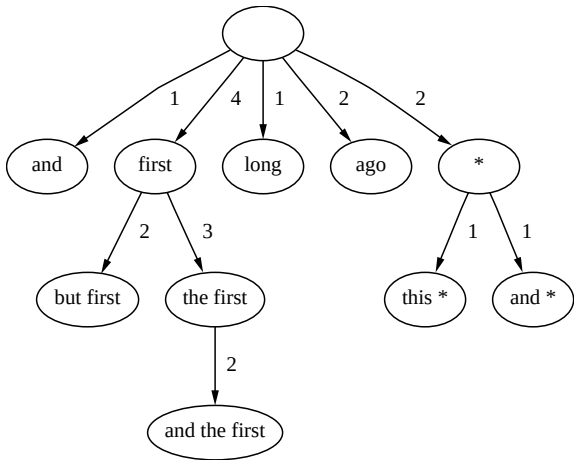

Figure 1: A small example of a PST of words for language modeling. The numbers on the edges are the weights of the sub-trees starting at the pointed node. These weights are used for tracking a mixture of PSTs. The special string * represents a 'wild-card' that can be matched with any observed word.

3 The Learning Algorithm

Within the framework of online learning, it is provably (see e.g. (DeSantis et al., 1988; Cesa-Bianchi et al., 1993)) and experimentally known that the performance of a weighted ensemble of models, each model weighted according to its performance (the posterior probability of the model), is not worse and generally much better than any single model in the ensemble. Although there might be exponentially many different PSTs in the ensemble, it has been recently shown (Willems et al., 1994) that a mixture of PSTs can be efficiently computed for small alphabets.

Here, we will use the Bayesian formalism to derive an *online* learning procedure for mixtures of PSTs of words. The mixture elements are drawn from some pre-specified set $\mathcal{T}$, which in our case is typically the set of all PSTs with maximal depth $\leq D$ for some suitably chosen D.

To each possible PST $T \in \mathcal{T}$ and observation length n observation sequence $w_1, \ldots, w_n$ we associate a confidence value given by T's likelihood (or evidence) on the past n observations $P(w_1, \ldots, w_n|T)$:

$$P(w_1, \ldots, w_n|T) = \prod_{i=1}^{n} \gamma_{C_T(w_1,\ldots,w_{i-1})}(w_i) \; , \tag{1}$$

where $C_T(w_0) = \epsilon$ is the null (empty) context. The probability of the next word, given the past n observations, is provided by Bayes formula,

$$P(w_n|w_1, \ldots, w_{n-1}) = \frac{P(w_1, \ldots, w_{n-1}, w_n)}{P(w_1, \ldots, w_{n-1})} \tag{2}$$

$$= \frac{\sum_{T\in\mathcal{T}} P_0(T)P(w_1, \ldots, w_{n-1}, w_n|T)}{\sum_{T\in\mathcal{T}} P_0(T)P(w_1, \ldots, w_{n-1}|T)} \; , \tag{3}$$

where $P_0(T)$ is the prior probability of the PST, T.

A naïve computation of (3) would be infeasible, because of the size of $\mathcal{T}$. Instead, we use a recursive method in which the relevant quantities for a PST mixture are computed efficiently from related quantities for sub-PSTs. In particular, the PST prior $P_0(T)$ is defined as follows. A node s has a probability α_s of being a leaf and a probability $1-\alpha_s$ of being an internal node. In the latter case, its sons are either a single wildcard, with probability β_s, or actual words with probability $1-\beta_s$. To keep the derivation simple, we assume here that the probabilities α_s are independent of s and that there are no wildcards, that is, $\beta_s = 0, \alpha_s = \alpha$ for all s. Context-dependent priors and trees with wildcards can be obtained by a simple extension of the present derivation. Let us also assume that all the trees have maximal depth D. Then $P_0(T) = \alpha^{n_1}(1-\alpha)^{n_2}$, where n_1 is the number of leaves of T *not* of less than maximal depth and n_2 is the number of internal nodes of T *not* at the maximal depth D.

To evaluate the likelihood of the whole mixture we build a tree of maximal depth D containing all observation sequence suffixes of length up to D. Thus the tree contains a node s iff $s = (w_{i-k+1}, \ldots, w_i)$ with $k \leq D$, $i \leq n$. At each node s we keep two variables.[1] The first, $L_n(s)$, accumulates the likelihood of the node seen as leaf. That is, $L_n(s)$ is the product of the predictions of the node on all the observation-sequence suffixes that ended at that node:

$$L_n(s) \quad = \prod_{\{i \,|\, C_T(w_1,\ldots,w_{i-1})=s,\, 1\leq i\leq n\}} P(w_i|s) \quad = \prod_{\{i \,|\, C_T(w_1,\ldots,w_{i-1})=s,\, 1\leq i\leq n\}} \gamma_s(w_i) \ . \tag{4}$$

For each new observed word w_n, the likelihood values $L_n(s)$ are derived from their previous values $L_{n-1}(s)$. Clearly, only the nodes labeled by w_{n-1}, $w_{n-2}w_{n-1}$, $\ldots$, $w_{n-D}\cdots w_{n-1}$ will need likelihood updates. For those nodes, the update is simply multiplication by the node's prediction for w_n:

$$L_n(s) = \begin{cases} L_{n-1}(s)\,\gamma_s(w_n) & s = C(w_1,\ldots,w_{n-1}),\ |s| \leq D \\ L_{n-1}(s) & \text{otherwise} \end{cases} \ , \tag{5}$$

The second variable, denoted by $Lmix_n(s)$, is the likelihood of the mixture of *all* possible trees that have a subtree rooted at s on the observed suffixes (all observations that reached s). $Lmix_n(s)$ is calculated recursively as follows:

$$Lmix_n(s) = \alpha L_n(s) \quad + \quad (1-\alpha)\prod_{u\in U} Lmix_n(us)\,, \tag{6}$$

The recursive computation of the mixture likelihood terminates at the leaves:

$$Lmix_n(s) = L_n(s) \text{if} |s| = D\,. \tag{7}$$

In summary, the mixture likelihoods are updated as follows:

$$Lmix_n(s) = \begin{cases} L_n(s) & s = C(w_1,\ldots,w_{n-1}),\ |s| = D \\ \alpha L_n(s) + (1-\alpha)\prod_{u\in U} Lmix_n(us) & s = C(w_1,\ldots,w_{n-1}),\ |s| < D \\ Lmix_{n-1}(s) & \text{otherwise} \end{cases} \ , \tag{8}$$

At first sight it would appear that the update of $Lmix_n$ would require contributions from an arbitrarily large subtree, since U may be arbitrarily large. However, only the subtree rooted at

[1] In practice, we keep only a ratio related to the two variables, as explained in detail in the next section.

$(w_{n-|s|-1}\, s)$ is actually affected by the update. so the following simplification holds:

$$\prod_{u \in U} Lmix_n(us) \;=\; Lmix_n(w_{n-|s|-1}\, s) \;\times \prod_{u \in U, u \neq w_{n-|s|-1}} Lmix_n(us) \;. \tag{9}$$

Note that $Lmix_n(s)$ is the likelihood of the weighted mixture of trees rooted at s on *all* past observations, where each tree in the mixture is weighted with its proper prior. Therefore,

$$Lmix_n(\epsilon) = \sum_{T \in \mathcal{T}} P_0(T) P(w_1, \ldots, w_n | T) \;, \tag{10}$$

where $\mathcal{T}$ is the set of trees of maximal depth D and ϵ is the null context (the root node). Combining Equations (3) and (10), we see that the prediction of the whole mixture for next word is the ratio of the likelihood values $Lmix_n(\epsilon)$ and $Lmix_{n-1}(\epsilon)$ at the root node:

$$P(w_n | w_1, \ldots, w_{n-1}) = Lmix_n(\epsilon) / Lmix_{n-1}(\epsilon) \;. \tag{11}$$

A given observation sequence matches a unique path from the root to a leaf. Therefore the time for the above computation is linear in the tree depth (maximal context length). After predicting the next word the counts are updated simply by increasing by one the count of the word, if the word already exists, or by inserting a new entry for the new word with initial count set to one. Based on this scheme several n-gram estimation methods, such as Katz's backoff scheme (Katz, 1987), can be derived. Our learning algorithm has, however, the advantages of not being limited to a constant context length (by setting D to be arbitrarily large) and of being able to perform online adaptation. Moreover, the interpolation weights between the different prediction contexts are automatically determined by the performance of each model on past observations.

In summary, for each observed word we follow a path from the root of the tree (back in the text) until a longest context (maximal depth) is reached. We may need to add new nodes, with new entries in the data structure, for the first appearance of a word. The likelihood values of the mixture of subtrees (Equation 8) are returned from each level of that recursion from the root node. The probability of the next word is then the ratio of two consecutive likelihood values returned at the root.

For prediction without adaptation, the same method is applied except that nodes are not added and counts are not updated. If the prior probability of the wildcard, β, is positive, then at each level the recursion splits, with one path continuing through the node labeled with the wildcard and the other through the node corresponding to the proper suffix of the observation. Thus, the update or prediction time is in that case $O(2^D)$. Since D is usually very small (most currently used word n-grams models are trigrams), the update and prediction times are essentially linear in the text length.

It remains to describe how the probabilities, $P(w|s) = \gamma_s(w)$ are estimated from empirical counts. This problem has been studied for more than thirty years and so far the most common techniques are based on variants of the Good-Turing (GT) method (Good, 1953; Church and Gale, 1991). Here we give a description of the estimation method that we implemented and evaluated. We are currently developing an alternative approach for cases when there is a known (arbitrarily large) bound on the maximal size of the vocabulary U.

Let $n_1^s, n_2^s, \ldots, n_r^s$, respectively, be the counts of occurrences of words $w_1, w_2, \ldots, w_{r^s}$ at a given context (node) s, where r^s is the total number of different words that have been observed at node s. The total text size in that context is thus $n^s = \sum_{i=1}^{r^s} n_i^s$. We need estimates of $\gamma_s(w_i)$ and of $\gamma_s(w_0)$, the probability of observing a new word w_0 at node s. The GT method sets $\gamma_s(w_0) = \frac{t_1^s}{n^s}$, where t_1 is the total number of words that were observed only once in that context. This method has several justifications, such as a Poisson assumption on the appearance of new words (Fisher et al., 1943). It is, however, difficult to analyze and requires keeping track of the rank of each word. Our learning scheme and data structures favor instead any method that is based only on word counts. In source coding it is common to assign to novel events the probability $\frac{r}{n+r}$. In this case the probability $\gamma_s(w_i)$ of a word that has been observed n_i^s times is set to $\frac{n_i^s}{n^s+r^s}$. As reported in (Witten and Bell, 1991), the performance of this method is similar to the GT estimation scheme, yet it is simpler since only the number of different words and their counts are kept.

Finally, a careful analysis should be made when predicting novel events (new words). There are two cases of novel events: (a) an occurrence of an entirely new word, that has never been seen before in any context; (b) an occurrence of a word that has been observed in some context, but is new in the current context.

The following coding interpretation may help to understand the issue. Suppose some text is communicated over a channel and is encoded using a PST. Whenever an entirely new word is observed (first case) it is necessary to first send an indication of a novel event and then transfer the identity of that word (using a lower level coder, for instance a PST over the alphabet Σ in which the words in U are written. In the second case it is only necessary to transfer the identity of the word, by referring to the shorter context in which the word has already appeared. Thus, in the second case we incur an additional description cost for a new word in the current context. A possible solution is to use a shorter context (one of the ancestors in the PST) where the word has already appeared, and multiply the probability of the word in that shorter context by the probability that the word is new. This product is the probability of the word.

In the case of a completely new word, we need to multiply the probability of a novel event by an additional factor $P_0(w_n)$ interpreted as the prior probability of the word according to a lower-level model. This additional factor is multiplied at all the nodes along the path from the root to the maximal context of this word (a leaf of the PST). In that case, however, the probability of the next word w_{n+1} remains independent of this additional prior, since it cancels out nicely:

$$P(w_{n+1}|w_1, \ldots, w_n) = \frac{Lmix_{n+1}(\epsilon) \times P_0(w_n)}{Lmix_n(\epsilon) \times P_0(w_n)} = \frac{Lmix_{n+1}(\epsilon)}{Lmix_n(\epsilon)} \ . \tag{12}$$

Thus, an entirely new word can be treated simply as a word that has been observed at all the nodes of the PST. Moreover, in many language modeling applications we need to predict only that the next event is a new word, without specifying the word itself. In such cases the update derivation remains the same as in the first case above.

4 Efficient Implementation of PSTs of Words

Natural language is often bursty (Church, this volume), that is, rare or new words may appear and be used relatively frequently for some stretch of text only to drop to a much lower frequency of

use for the rest of the corpus. Thus, a PST being build online may only need to store information about those words for a short period. It may then be advantageous to prune PST nodes and remove small counts corresponding to rarely used words. Pruning is performed by removing all nodes from the suffix tree whose counts are below a threshold, after each batch of K observations. We used a pruning frequency K of 1000000 and a pruning threshold of 2 in some of our experiments.

Pruning during online adaptation has two advantages. First, it improves memory use. Second, and less obvious, predictive power may be improved power. Rare words tend to bias the prediction functions at nodes with small counts, especially if their appearance is restricted to a small portion of the text. When rare words are removed from the suffix tree, the estimates of the prediction probabilities at each node are readjusted reflect better the probability estimates of the more frequent words. Hence, part of the bias in the estimation may be overcome.

To support fast insertions, searches and deletions of PST nodes and word counts we used a hybrid data structure. When we know in advance a (large) bound on vocabulary size, we represent the root node by arrays of word counts and possible sons subscripted by word indices. At other nodes, we used *splay* trees (Sleator and Tarjan, 1985) to store both the counts and the branches to longer contexts. Splay trees support search, insertion and deletion in amortized $O(\log(n))$ time per operation. Furthermore, they reorganize themselves to so as to decrease the cost of accessing to the most frequently accessed elements, thus speeding up access to to counts and subtrees associated to more frequent words. Figure 2 illustrates the hybrid data structure:

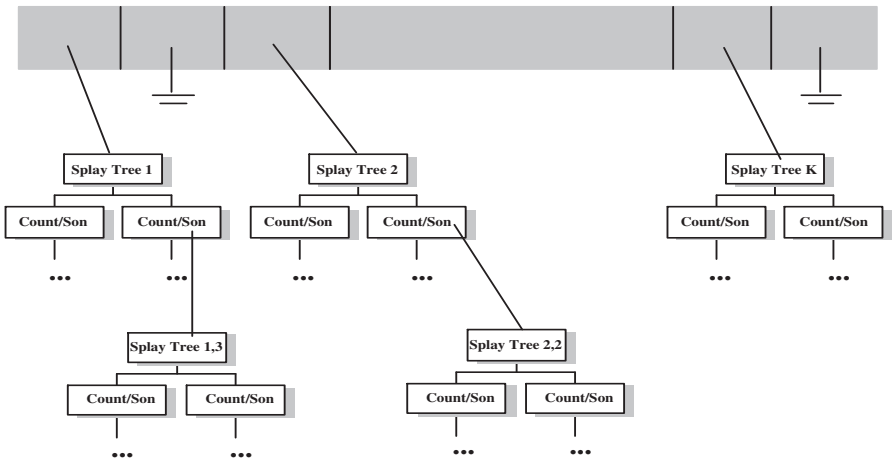

Figure 2: The hybrid data structure that represents the suffix tree and the prediction functions at each node.

The likelihoods $Lmix_n(s)$ and $L_n(s)$ decrease exponentially fast with n, potentially causing numerical problems even if log representation is used. Moreover, we are only interested in the predictions of the mixtures; the likelihoods are only used to weigh the predictions of different nodes. Let $\tilde{\gamma}_s(w_n)$ be the prediction of the *weighted* mixture of all subtrees rooted below s (including s itself) for w_n. By following the derivation presented in the previous section it can be verified that,

$$\tilde{\gamma}_s(w_{n+1}) = q_n(s)\gamma_s(w_{n+1}) \; + \; (1 - q_n(s))\tilde{\gamma}_{(w_{n-|s|}\, s)}(w_{n+1}) \;\; , \tag{13}$$

where

$$q_n(s) \;\; = \;\; \alpha L_n(s) / \left(\alpha L_n(s) + (1-\alpha) \prod_{u \in U} Lmix_n(us) \right) \tag{14}$$

$$= \quad 1/\left(1 + \frac{(1-\alpha)\prod_{u \in U} Lmix_n(us)}{\alpha L_n(s)}\right) \quad . \tag{15}$$

Define

$$R_n(s) = \log\left(\frac{\alpha L_n(s)}{(1-\alpha)\prod_{u \in U} Lmix_n(us)}\right) \quad . \tag{16}$$

Setting $R_0(s) = \log(\alpha/(1-\alpha))$ for all s, $R_n(s)$ is updated as follows:

$$R_{n+1}(s) = R_n(s) + \log\left(\gamma_s(w_{n+1})\right) - \log\left(\tilde{\gamma}_{(w_{n-|s|}s)}(w_{n+1})\right) \quad , \tag{17}$$

and $q_n(s) = 1/(1+e^{-R_n(s)})$. Thus, the probability of w_{n+1} is propagated along the path corresponding to suffixes of the observation sequence towards the root as follows,

$$\tilde{\gamma}_s(w_{n+1}) = \begin{cases} \gamma_s(w_{n+1}) & s = C(w_1, \ldots, w_n),\ |s| = D \\ q_n \gamma_s(w_{n+1}) \ + \ (1-q_n)\tilde{\gamma}_{(w_{n-|s|}s)}(w_{n+1}) & s = C(w_1, \ldots, w_n),\ |s| < D \end{cases} \quad . \tag{18}$$

Finally, the prediction of the complete mixture of PSTs for w_n is simply given by $\tilde{\gamma}_\epsilon(w_n)$.

5 Evaluation

We tested our algorithm in two modes. In online mode, model structure and parameters (counts) are updated after each observation. In batch mode, the structure and parameters are held fixed after the training phase, making it easier to compare the model to standard n-gram models. Our initial experiments used the Brown corpus, the Gutenberg Bible, and Milton's Paradise Lost as sources of training and test material. We have also carried out a preliminary evaluation on the ARPA North-American Business News (NAB) corpus.

For batch training, we partitioned randomly the data into training and testing sets. We then trained a model by running the online algorithm on the training set, and the resulting model, kept fixed, was then used to predict the test data.

As a simple check of the model, we used it to generate text by performing random walks over the PST. A single step of the random walk was performed by going down the tree following the current context and stop at a node with the probability assigned by the algorithm to that node. Once a node is chosen, a word is picked randomly by the node's prediction function. A result of such a random walk is given in Figure 3. The PST was trained on the Brown corpus with maximal depth of five. The output contains several well formed (meaningless) clauses and also cliches such as "conserving our rich natural heritage," suggesting that the model captured some longer-term statistical dependencies.

In online mode the advantage of PSTs with large maximal depth is clear. The perplexity of the model decreases significantly as a function of the depth. Our experiments so far suggest that the resulting models are fairly insensitive to the choice of the prior probability, α, and a prior which favors deep trees performed well. Table 1 summarizes the results on different texts, for trees of growing maximal depth. Note that a maximal depth 0 corresponds to a 'bag of words' model (zero order), 1 to a bigram model, and 2 to a trigram model.

every year public sentiment for conserving our rich natural heritage is growing but that heritage is shrinking even faster no joyride much of its contract if the present session of the cab driver in the early phases conspiracy but lacking money from commercial sponsors the stations have had to reduce its vacationing

Figure 3: Text created by a random walk over a PST trained on the Brown corpus.

Text	*Maximal Depth*	*Number of Nodes*	*Perplexity* ($\alpha = 0.5$)	*Perplexity* ($\alpha = 0.999$)	*Perplexity* ($\alpha = 0.001$)
Bible (Gutenberg Project)	0	1	282.1	282.1	282.1
	1	7573	84.6	84.6	84.6
	2	76688	55.9	58.2	55.5
	3	243899	42.9	50.9	42.5
	4	477384	37.8	49.8	37.5
	5	743830	36.5	49.6	35.6
Paradise Lost by John Milton	0	1	423.0	423.0	423.0
	1	8754	348.7	348.7	348.7
	2	59137	251.1	289.7	243.9
	3	128172	221.2	285.3	206.4
	4	199629	212.5	275.2	202.1
	5	271359	209.3	265.6	201.6
Brown Corpus	0	1	452.8	452.8	452.8
	1	12647	276.5	276.5	276.5
	2	76957	202.9	232.6	197.1
	3	169172	165.8	224.0	165.6
	4	267544	160.5	223.9	159.7
	5	367096	158.7	223.8	158.7

Table 1: The perplexity of PSTs for the online mode.

In our first batch tests we trained the model on 15% of the data and tested it on the rest. The results are summarized in Table 2. The perplexity obtained in the batch mode is clearly higher than that of the online mode, since a small portion of the data was used to train the models. Yet, even in this case the PST of maximal depth three is significantly better than a full trigram model. In this mode we also checked the performance of the single most likely (maximum aposteriori) model compared to the mixture of PSTs. This model is found by pruning the tree at the nodes that obtained the highest confidence value, $L_n(s)$, and using only the leaves for prediction. As shown in the table, the performance of the MAP model is consistently worse than the performance of the mixture of PSTs.

As a simple test of for applicability of the model for language modeling, we checked it on text which was corrupted in different ways. This situation frequently occurs in speech and handwriting recognition systems or in machine translation. In such systems the last stage is a language model, usually a trigram model, that selects the most likely alternative between the several options passed by the previous stage. Here we used a PST with maximal depth 4, trained on 90% of the text of *Paradise Lost*. Several sentences that appeared in the test data were corrupted in different ways.

Text	*Maximal Depth*	*Perplexity* ($\alpha = 0.5$)	*Perplexity* (MAP Model)
Bible	0	411.3	411.3
(Gutenberg	1	172.9	172.9
Project)	2	149.8	150.8
	3	141.2	143.7
	4	139.4	142.9
	5	139.0	142.7
Paradise Lost	0	861.1	861.1
by	1	752.8	752.8
John Milton	2	740.3	746.9
	3	739.3	747.7
	4	739.3	747.6
	5	739.3	747.5
Brown	0	564.6	564.6
Corpus	1	407.3	408.3
	2	396.1	399.9
	3	394.9	399.4
	4	394.5	399.2
	5	394.4	399.1

Table 2: The perplexity of PSTs for the batch mode.

We then used the model in the batch mode to evaluate the likelihood of each of the alternatives. In Table 3 we demonstrate one such case, where the first alternative is the correct one. The negative log likelihood and the posterior probability, assuming that the listed sentences are all the possible alternatives, are provided. The correct sentence gets the highest probability according to the model.

	Negative Log. Likl.	Posterior Probability
from god and over wrath grace shall abound	74.125	0.642
from god but over wrath grace shall abound	82.500	0.002
from god and over worth grace shall abound	75.250	0.295
from god and over wrath grace will abound	78.562	0.030
before god and over wrath grace shall abound	83.625	0.001
from god and over wrath grace shall a bound	78.687	0.027
from god and over wrath grape shall abound	81.812	0.003

Table 3: The likelihood induced by a PST of maximal depth 4 for different corrupted sentences.

Finally, we trained a depth two PST on randomly selected sentences from the NAB corpus totaling approximately 32.5 million words. The test set perplexity of the resulting PST was 168 on a separate set of sentences totaling around 2.8 million words drawn from the remaining NAB data. Further experiments using longer maximal depth and allowing comparisons with existing n-gram models trained on the full (280 million word) NAB corpus will require improved data structures

and pruning policies to stay within reasonable memory limits.

6 Conclusions and Further Work

PSTs are able to capture longer correlations than traditional fixed order n-grams, supporting better generalization ability from limited training data. This is especially noticeable when phrases longer than a typical n-gram order appear repeatedly in the text. The PST learning algorithm allocates a proper node for the phrase whereas a bigram or trigram model captures only a truncated version of the statistical dependencies among words in the phrase.

Our current learning algorithm is able to handle moderate size corpora, but we hope to adapt it to work with very large training corpora (100s of millions of words). The main obstacle to those applications is the space required for the PST. More extensive pruning may be useful for such large training sets, but the most promising approach may involve a batch training algorithm that builds a compressed representation of the PST final from an efficient representation, such as a suffix array, of the relevant subsequences of the training corpus.

References

T.C. Bell, J.G. Cleary, I.H. Witten. 1990. Text Compression. Prentice Hall.

P.F. Brown, V.J. Della Pietra, P.V. deSouza, J.C. Lai, R.L. Mercer. 1990. Class-based n-gram models of natural language. In *Proceedings of the IBM Natural Language ITL*, pages 283–298, Paris, France, March.

N. Cesa-Bianchi, Y. Freund, D. Haussler, D.P. Helmbold, R.E. Schapire, M. K. Warmuth. 1993. How to use expert advice. Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC).

K.W. Church and W.A. Gale. 1991. A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams. *Computer Speech and Language*, 5:19–54.

A. DeSantis, G. Markowski, M.N. Wegman. 1988. Learning Probabilistic Prediction Functions. Proceedings of the 1988 Workshop on Computational Learning Theory, pp. 312–328.

R.A. Fisher, A.S. Corbet, C.B. Williams. 1943. The relation between the number of species and the number of individuals in a random sample of an animal population. J. Animal Ecology, Vol. 12, pp. 42–58.

G.I. Good. 1953. The population frequencies of species and the estimation of population parameters. Biometrika, 40(3):237–264.

G.I. Good. 1969. Statistics of Language: Introduction. Encyclopedia of Linguistics, Information and Control. A. R. Meetham and R. A. Hudson, editors. pages 567–581. Pergamon Press, Oxford, England.

D. Hindle. 1990. Noun classification from predicate-argument structures. In *28th Annual Meeting of the Association for Computational Linguistics*, pages 268–275, Pittsburgh, Pennsylvania. Association for Computational Linguistics, Morristown, New Jersey.

D. Hindle. 1993. A parser for text corpora. In B.T.S. Atkins and A. Zampoli, editors, *Computational Approaches to the Lexicon*. Oxford University Press, Oxford, England. To appear.

S.M. Katz. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. IEEE Trans. on ASSP 35(3):400–401.

R.E. Krichevsky and V.K. Trofimov. 1981. The performance of universal encoding. IEEE Trans. on Inform. Theory, pp. 199-207.

P. Resnik. 1992. WordNet and distributional analysis: A class-based approach to lexical discovery. In *AAAI Workshop on Statistically-Based Natural-Language-Processing Techniques*, San Jose, California, July.

J. Rissanen. 1986. A universal prior for integers and estimation by minimum description length. The Annals of Statistics, 11(2):416–431.

D. Ron, Y. Singer, N. Tishby. 1994. The power of amnesia: learning probabilistic automata with variable memory length. Machine Learning (to appear).

C.E. Shannon 1951. Prediction and Entropy of Printed English. Bell Sys. Tech. J., Vol. 30, No. 1, pp. 50–64.

D.D. Sleator and R.E. Tarjan. 1985. Self-Adjusting Binary Search Trees. Journal of the ACM, Vol. 32, No. 3, pp. 653–686.

F.M.J. Willems, Y.M. Shtarkov, T.J. Tjalkens. 1994. The context tree weighting method: basic properties. Submitted to IEEE Trans. on Inform. Theory.

I.H. Witten and T.C. Bell. 1991. The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression. IEEE Trans. on Inform. Theory, 37(4):1085–1094.