A Tree-Based Statistical Language Model for Natural Language Speech Recognition

LALIT R. BAHL, MEMBER, IEEE, PETER F. BROWN, PETER V. DE SOUZA, AND ROBERT L. MERCER, MEMBER, IEEE

Abstract—**This paper is concerned with the problem of "predicting" the next word a speaker will say, given the words already spoken; specifically, the problem is to estimate the probability that a given word will be the next word uttered. Algorithms are presented for automatically constructing a binary decision tree designed to estimate these probabilities. At each node of the tree there is a yes/no question relating to the words already spoken, and at each leaf there is a probability distribution over the allowable vocabulary. Ideally, these nodal questions can take the form of arbitrarily complex Boolean expressions, but computationally cheaper alternatives are also discussed. The paper includes some results obtained on a 5000-word vocabulary with a tree designed to predict the next word spoken from the preceding 20 words. The tree is compared to an equivalent trigram model and shown to be superior.**

I. INTRODUCTION

GIVEN some acoustic evidence A derived from a spoken word sequence $\bar{W}$, the problem in automatic speech recognition is to determine $\bar{W}$. In one approach [1], [5], an estimate $\hat{W}$ of $\bar{W}$ is obtained as

$$\hat{W} = \underset{W}{\operatorname{argmax}} \Pr\{W|A\}. \quad (1)$$

By Bayes' rule, this can be rewritten as

$$\hat{W} = \underset{W}{\operatorname{argmax}} \frac{\Pr\{A|W\} \cdot \Pr\{W\}}{\Pr\{A\}}. \quad (2)$$

The denominator in (2) is independent of W and can be ignored in the search for $\hat{W}$. The first term in the numerator is the probability of observing the acoustic evidence A when the speaker says W. It is the job of an acoustic model to estimate this probability. The second term in the numerator is the probability that the speaker will say W. This is called the *prior* probability of W. The purpose of a language model is to estimate the prior probabilities.

The prior probability of a word sequence $W = w_1, w_2, \cdots, w_n$ can be written as

$$\Pr\{W\} = \prod_{i=1}^{n} \Pr\{w_i|w_1, w_2, \cdots, w_{i-1}\}, \quad (3)$$

so the task of language modeling can be reduced to the problem of estimating terms like $\Pr\{w_i|w_1, w_2, \cdots, w_{i-1}\}$: the probability that w_i will be said after $w_1, w_2, \cdots, w_{i-1}$. The number of parameters of a language model is the number of variables that must be known in advance in order to compute $\Pr\{w_i|w_1, w_2, \cdots, w_{i-1}\}$ for any word sequence $w_1, w_2, \cdots, w_i$.

In any practical natural-language system with even a moderate vocabulary size, it is clear that the language model probabilities $\Pr\{w_i|w_1, w_2, \cdots, w_{i-1}\}$ cannot be stored for each possible sequence $w_1, w_2, \cdots, w_i$. Even if the sequences were limited to one or two sentences in length, the number of distinct sequences would be so large that a complete set of probabilities could not be computed, never mind stored or retrieved. To be practicable, then, a language model must have many fewer parameters than the total number of possible sequences $w_1, w_2, \cdots, w_i$. An obvious way to limit the number of parameters is to partition the various possible word histories $w_1, w_2, \cdots, w_{i-1}$ into a manageable number of equivalence classes.

A simple-minded, but surprisingly effective, definition of equivalence classes can be found in the N-gram language model [5], [10]. In this model, word sequences are treated as equivalent if and only if they end with the same $N-1$ words. Typically $N = 3$, in which case the model is referred to as a 3-gram or trigram model. The trigram model is based upon the approximation

$$\Pr\{w_i|w_1, w_2, \cdots, w_{i-1}\} \simeq \Pr\{w_i|w_{i-2}, w_{i-1}\}, \quad (4)$$

which is clearly inexact, but apparently quite useful. Maximum-likelihood estimates of N-gram probabilities can be obtained from their relative frequencies in a large body of training text. But since many legitimate N-grams are likely to be missing from the training text, it is necessary to "smooth" the maximum-likelihood estimates so as to avoid probabilities of zero. The trigram model can be smoothed in a natural way using the bigram and unigram relative frequencies as described in [6].

The trigram language model has the following advantages: the equivalence classes are easy to determine, the relative frequencies can all be precomputed with very little computation, and the probabilities can be smoothed "on the fly" quickly and efficiently. The main disadvantage of the trigram model lies in its naive definition of

Manuscript received November 24, 1987; revised September 23, 1988.
The authors are with the Speech Recognition Group, IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598.
IEEE Log Number 8928123.

equivalence classes. All words prior to the most recent two are ignored, and useful information is lost. Additionally, word sequences ending in different pairs of words should not necessarily be considered distinct; they may be functionally equivalent from a language model point of view. Separating equivalent histories into different classes, as the trigram model does, fragments the training data unnecessarily and reduces the accuracy of the resulting probability estimates.

In Section II we describe a tree-based language model which avoids both of the above weaknesses in the trigram model. Although the tree-based model is as convenient to apply as the trigram model, it requires a massive increase in computation to construct. Some results are presented in Section III.

II. Constructing a Tree-Based Language Model

Let us now consider the construction of binary decision trees and their application to the language modeling problem. An example of a binary decision tree is shown in Fig. 1.

At each nonterminal node of the tree, there is a question requiring a yes/no answer, and corresponding to each possible answer there is a branch leading to the next question. Associated with each terminal node, i.e., leaf, is some advice or information which takes into account all the questions and answers which lead to that leaf. The application of binary decision trees is much like playing the venerable TV game "What's My Line?" where contestants try to deduce the occupation of a guest by asking a series of yes/no questions. In the context of language modeling, the questions relate to the words already spoken; for example: "Is the preceding word a verb?". And the information at each leaf takes the form of a probability distribution indicating which words are likely to be spoken next. The leaves of the tree represent language model equivalence classes. The classes are defined implicitly by the questions and answers leading to the leaves.

Notice that an N-gram language model is just a special case of a binary-tree model. With enough questions of the form "Is the last word w_i?" or "Is the second to last word w_j?" the N-gram language model can be represented in the form of a tree. Thus, an optimally constructed tree language model is guaranteed to be at least as good as an optimally constructed N-gram model, and is likely to be much better, given the weaknesses in N-gram models already discussed.

The object of a decision tree is to reduce the uncertainty of the event being decided upon, be it an occupation in a game show, a diagnosis, or the next word a speaker will utter. In language modeling, this uncertainty is measured by the entropy of the probability distributions at the leaves. We seek a decision tree which minimizes the average entropy of the leaf distributions. Let $\{l_1, l_2, \cdots, l_L\}$ denote the leaves of a decision tree, and let Y denote

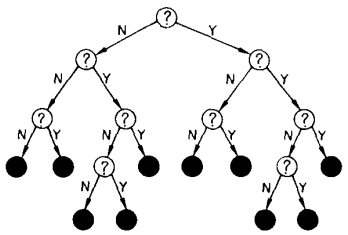


Fig. 1. Example of a binary decision tree.

the event being decided upon—the next word spoken. The average entropy of the leaf distributions is

$$\overline{H}(Y) = \sum_{i=1}^{L} H_i(Y) \cdot \Pr\{l_i\}, \tag{5}$$

where $H_i(Y)$ is the entropy of the distribution associated with leaf l_i, and $\Pr\{l_i\}$ is the prior probability of visiting leaf l_i. The entropy at leaf l_i, measured in bits, is

$$H_i(Y) = -\sum_{j=1}^{V} \Pr\{w_j|l_i\} \cdot \log_2 \Pr\{w_j|l_i\}, \tag{6}$$

where V is the size of the vocabulary, and $\Pr\{w_j|l_i\}$ is the probability that word w_j will be the next word spoken given that the immediate word history leads to leaf l_i.

Notice that the average entropy is defined in terms of the *true* leaf distributions, and not the relative frequencies as obtained from a sample of data. The sample entropy, i.e., the entropy on training data, can always be made arbitrarily small by increasing the number of leaves so as to make the leaf sample sizes arbitrarily small. The entropy on test data, however, cannot usually be made arbitrarily small. Typically, trees which perform extremely well on training data achieve their success by modeling idiosyncrasies of the training data themselves rather than by capturing generalizations about the process which gave rise to the data. In the algorithm described below, in order to avoid making such deceptively good trees, we divide the data into two independent halves. Improvements suggested by one half are verified against the other half. If they do not perform well there, they are rejected.

Although we would like to construct the tree with minimum entropy for test data, there is no way of doing so. The best hope is for a "good" tree, having low entropy on test data. Searching for a tree which is good is essentially a problem in heuristics, and many procedures are possible. These procedures include manual tree construction by linguistic experts, as well as automatic tree-growing algorithms. Automatic methods have some important advantages over manual construction: they are not generally language-dependent and may therefore be applied to any language, any domain, and any vocabulary of interest. In many cases, expert linguistic knowledge may not even exist.

In this paper we shall describe only one automatic method: a greedy algorithm, with restrictions on the form

of the questions. The algorithm is greedy in the sense that at any node in the tree the question selected is the one giving the greatest reduction in entropy at that node, without regard to subsequent nodes. Thus, the algorithm aims to construct a tree which is locally optimal, but very likely not globally optimal; the hope being that a locally optimal tree will be globally "good." This tree-construction paradigm has been advocated before [4], and has been used successfully in other applications [3], [8]. A dynamic programming algorithm for the determination of a truly optimal tree is described in [9], but is only suitable in restricted applications with relatively few variables; it is inappropriate for the present application. A treatise on the art and science of tree growing can be found in [2].

Because the search space of possible questions is enormous, we will place restrictions on the form of the questions. Initially, we shall only allow "elementary" questions of the form "$X \in S$?" where X denotes a discrete random variable with a finite number of possible values, and S is a subset of the values taken by X. For example, if X denotes the preceding word, and S is the set of all verbs in the permitted vocabulary, then "$X \in S$?" represents the question "Is the preceding word a verb?". Similarly, if X denotes the head of the most recent noun phrase, and S is the set of all plural nouns in the vocabulary, then "$X \in S$?" represents the question "Does the head of the most recent noun phrase have the attribute *plural*?". Later we shall discuss the possibility of relaxing these restrictions by allowing composite questions comprising several elementary questions.

Let $X_1, X_2, \cdots, X_m$ be the discrete random variables whose values may be questioned. We shall call these the *predictor* variables. Clearly the power of the decision tree depends on the number and choice of predictor variables. The trigram model makes use of only two predictors: the preceding word, X_1, and the word before that, X_2. But, obviously, the list of predictor variables need not be limited to those two; it can easily be extended to include the last N words spoken, $N > 2$, without encountering the difficulty of combinatorial explosion that besets the general N-gram model. Furthermore, if a parser that can handle partial sentences is available, then the list of predictors can also include information from the parser, like the head of the most recent noun phrase. As far as tree construction is concerned, it makes no difference how the predictors are defined; the construction algorithm only uses their values and the value of the next word, Y. As far as the number of equivalence classes is concerned, it makes no difference how many predictor variables there are; increasing the number of predictors does not, in itself, increase the number of equivalence classes. However, the amount of computation involved in tree construction will increase as the number of predictors increases, and therefore, the choice of predictors should be limited to those variables which provide significant information about the word being predicted.

The tree-growing algorithm can be summarized as follows.

1) Let c be the current node of the tree. Initially c is the root.

2) For each predictor variable X_i ($i = 1, 2, \cdots, m$), find the set S_i^c which minimizes the average conditional entropy at node c

$$\begin{aligned}\overline{H}_c(Y|\text{"}X_i \in S_i^c\text{?"}) &= -\Pr\{X_i \in S_i^c | c\} \sum_{j=1}^{V} \Pr\{w_j | c, X_i \in S_i^c\} \\ &\quad \cdot \log_2 \Pr\{w_j | c, X_i \in S_i^c\} \\ &\quad - \Pr\{X_i \notin S_i^c | c\} \sum_{j=1}^{V} \Pr\{w_j | c, X_i \notin S_i^c\} \\ &\quad \cdot \log_2 \Pr\{w_j | c, X_i \notin S_i^c\}. \end{aligned} \tag{7}$$

3) Determine which of the m questions derived in Step 2 leads to the lowest entropy. Let this be question k, i.e.,

$$k = \underset{i}{\operatorname{argmin}}\, \overline{H}_c(Y|\text{"}X_i \in S_i^c\text{?"}). \tag{8}$$

4) The reduction in entropy at node c due to question k is

$$R_c(k) = H_c(Y) - \overline{H}_c(Y|\text{"}X_k \in S_k^c\text{?"}), \tag{9}$$

where

$$H_c(Y) = -\sum_{j=1}^{V} \Pr\{w_j | c\} \cdot \log_2 \Pr\{w_j | c\}.$$

If this reduction is "significant," store question k, create two descendant nodes, c_1 and c_2, corresponding to the conditions $X_k \in S_k^c$ and $X_k \notin S_k^c$, and repeat Steps 2–4 for each of the new nodes separately.

The reduction in entropy in Step 4 is the mutual information between Y and question k at node c. Thus, seeking questions which minimize entropy is just another way of saying that questions are sought which are maximally informative about the event being predicted—an eminently reasonable criterion.

The true probabilities in Step 2 are generally unknown. In practice, they can be replaced by estimates obtained from relative frequencies in a sample of training text. This means, of course, that the algorithm must be conducted using estimates $\hat{\overline{H}}$, $\hat{H}$, and $\hat{R}$, instead of the true values $\overline{H}$, H, and R, respectively.

If the distribution of $\hat{R}$ were known, its statistical significance could be determined in Step 4 when assessing the utility of the selected question. In our case, however, we were unable to determine the distribution of $\hat{R}$ and resorted to an empirical test of significance instead. Using independent (held-out) training data, we computed the reduction in entropy due to the selected question, and retained the question if the reduction exceeded a threshold.

The threshold allows the arborist to state what is considered to be *practically* significant, as opposed to *statistically* significant. The definition of "significant" in Step 4 determines whether or not a node is subdivided into two subnodes, and is responsible, therefore, for the ultimate size of the tree, and the amount of computation required to construct it.

The only remaining issue to be addressed in the above tree-growing algorithm is the determination of the set S_i^c in Step 2. This amounts to partitioning the values taken by X_i into two groups: those in S_i^c and those not in S_i^c. Again, there is no known practical way of achieving a certifiably optimal partition, especially in applications like language modeling where X_i can take a large number of different values. As before, the best realistic hope is to find a "good" set S_i^c via some kind of heuristic search. Possible search strategies range from relatively simple greedy algorithms to the computationally expensive techniques of simulated annealing [7].

Let $\boldsymbol{X}$ denote the set of values taken by the variable X. In our case, $\boldsymbol{X}$ is the entire vocabulary. The following algorithm determines a set $\boldsymbol{S}$ in a greedy fashion.

1) Let $\boldsymbol{S}$ be empty.

2) Insert into $\boldsymbol{S}$ the $x \in \boldsymbol{X}$ which leads to the greatest reduction in the average conditional entropy (7). If no $x \in \boldsymbol{X}$ leads to a reduction, make no insertion.

3) Delete from $\boldsymbol{S}$ any member x, if so doing leads to a reduction in the average conditional entropy.

4) If any insertions or deletions were made to $\boldsymbol{S}$, return to Step 2.

Using this set-construction algorithm and the earlier tree-growing algorithm, a language-model decision tree could certainly be constructed, but because of the restrictive form of the questions it would be somewhat inefficient. For example, suppose that the best question to place at some node of the tree is actually a composite question of the form: "Is $(X_i \in S_i^c)$ OR $(X_j \in S_j^c)$?". This is still a binary question, but with the existing restrictions on the form of the questions, it can only be implemented as two separate questions in such a way that the data are split into three subgroups rather than two. The data for which the answer is "yes" are unavoidably fragmented across two nodes. This is inefficient in the sense that these two nodes may be equivalent from a language model point of view. Splitting data unnecessarily across two nodes leads to duplicated branches, unnecessary computation, reduced sample sizes, and less accurate probability estimates.

To avoid this inefficiency, the elementary questions at each node of the tree must be replaced by composite binary questions. We would like the structure of the composite questions to permit Boolean expressions of arbitrary complexity, containing any number of elementary conditions of the form $X \in \boldsymbol{S}$ and any number of AND, OR, and NOT operators, in any combination. This can be achieved if the questions are themselves represented by binary trees, but with the leaves tied in such a way that there are only two outcomes. Fig. 2 shows an example of such a structure.

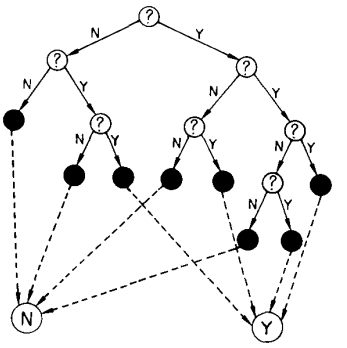


Fig. 2. Example of the topology of an unrestricted composite binary question.

The tree in Fig. 2 has eight leaves which are tied as shown by the dotted lines. The four leaves attached to no-branches are tied; they lead to the same final N-state. And the four leaves attached to yes-branches are tied similarly. Thus, the structure is still that of a binary question; there are only two possible final states. There are many different routes to each of the final states, each route representing a different series of conditions. Thus, the composite condition leading to either one of the final states can only be described with multiple OR operators: one for each distinct route. Similarly, the description of any particular route requires the use of multiple AND operators: one for each node on the route. Since there need be no limits on the number of leaves or the depth of a binary tree, it can be seen that any composite binary question, however complex, can be represented by a binary tree with tied leaves similar to that shown in Fig. 2. Note that NOT operators are superfluous here. The condition "NOT $(X \in \boldsymbol{T})$" can be rewritten in elementary form "$X \in \boldsymbol{S}$" by defining $\boldsymbol{S}$ to be the complement of $\boldsymbol{T}$.

Since composite questions are essentially just binary trees with elementary questions at each node, they can be constructed using the greedy tree-growing and set-construction algorithms already described. The entropies, however, must be computed from relative frequencies obtained after pooling all tied data.

Constructing a tree-based language model with composite questions involves a lot of computation. There are trees within trees: at each node of the global binary tree there is a local binary tree. If the computation appears to be excessive, it may be necessary to compromise on some form of question that is more general than an elementary question, but less general than a fully fledged binary tree. One compromise is provided by the pylon shown in Fig. 3.

The pylon has two states, N and Y. Starting at the top (level 1), all the data are placed in the N-state. An elementary question is then sought which splits the data in the N-state into two subgroups; the data answering "yes" being assigned to the Y-state, and the rest remaining in

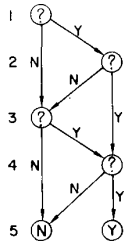


Fig. 3. Example of a pylon: a restricted form of composite binary question.

the *N*-state. At level 2, a refinement is applied to the data in the *Y*-state; an elementary question is sought which splits the data into two subgroups: the data answering "no" being returned to the *N*-state to rejoin the data already there, and the rest remaining in the *Y*-state. The procedure continues in this fashion, level by level, swapping data from one state to the other, until no further swaps can be found which lower the entropy.

The pylon is equivalent to a restricted binary tree which is constructed by determining a single question to apply simultaneously to all terminal nodes attached to no-branches, followed by a single question to apply simultaneously to all terminal nodes attached to yes-branches, and so on. Thus, the questions attached to several different no-branches are constrained to be identical, as are the questions attached to several different yes-branches.

The pylon has no memory: there are many different routes to most pylonic nodes, but no distinction is made between those different routes when the data are processed. The unconstrained binary tree has memory: different routes lead to different nodes, and since different nodes are subjected to different questions, a distinction is made between the different routes when the data are processed. Hence, the questions in the tree can be tailored to the questions already asked; in the pylon they cannot.

Although most composite questions cannot be represented in pylonic form, many useful composite questions can. For example, "Is the preceding word an adjective AND the one before that an article?". Or, "Is there any computer jargon in the last fifteen words?". The latter question requires a pylon 30 levels deep. As can be seen from these examples, the pylon can express certain types of semantic questions as well as grammatical questions. Both are important in determining what word will be spoken next.

Having constructed a tree, with whatever question topology can be afforded, there remains one important issue: the estimation of the probability distributions at the leaves of the decision tree. With a relatively small tree of 10 000 leaves, and a modest vocabulary of 10 000 words, there are one-hundred-million probabilities to estimate. Even with a reasonably generous one-billion words of training data, there would still be insufficient data to estimate the probabilities accurately. Thus, the probability distributions cannot simply be estimated from the relative frequencies in the training data, which are biased in any case. Just as the trigram probabilities are smoothed with lower order bigram and unigram distributions to ameliorate problems due to sparse data, so too can each leaf distribution be smoothed with the lower order nodal distributions between the root and the leaf.

Let $\{n_1, n_2, \cdots, n_r\}$ denote the set of nodes between the root, n_1, and a given leaf, n_r. Let $q_i(w)$ denote the relative frequency of word w at node n_i as obtained from the tree-growing training text, and let $\boldsymbol{q}_i$ denote the distribution of relative frequencies at node n_i. Further, let $\boldsymbol{q}_0$ denote the uniform distribution over the vocabulary. A smoothed leaf probability distribution may be obtained as

$$\tilde{\boldsymbol{q}}_r = \sum_{i=0}^{r} \lambda_i \boldsymbol{q}_i, \tag{10}$$

where the λ's are chosen to maximize the probability of some additional independent (held-out) training data, subject to the constraints that

$$\sum_{i=0}^{r} \lambda_i = 1 \tag{11}$$

and $\lambda_i \geq 0$, $(0 \leq i \leq r)$. The λ values may be determined from independent training data using the forward–backward parameter estimation algorithm, as described in [1].

It should be clear from the above prescription that tree-growing involves a great deal of computation; the vast majority being devoted to set construction. Certainly it involves a lot more work than the creation of a trigram model. It is also clear, however, that at any given node of the decision tree, the best question to ask is completely independent of the data and questions at any other node. This means that the nodes can be processed independently in any convenient order. Given enough parallel processors, tree growing need not take an excessive amount of time.

III. Results

We tested the foregoing ideas on tree-based language models in a pilot experiment involving a 5000-word vocabulary. The vocabulary consisted of the 5000 most frequent words in a database of IBM office correspondence.

The training and test data were drawn from 550 books and magazines which ranged from intellectually stimulating romantic novels to silly issues of *Datamation*. The books and magazines were divided randomly into four parts as follows:

1) training data for tree construction: approximately 10-million words;

2) data for testing the significance of a reduction in entropy: approximately 10-million words;

3) data for computing the smoothed leaf probability distributions: approximately 9-million words; and

4) test data; approximately 1-million words.

No book or magazine was split between more than one of the above categories. Words not in the vocabulary were treated as a single generic "unknown" word.

The purpose of the experiment was to predict the 21st word of a 21-gram, given the first 20 words. This is equivalent to predicting the next word a speaker will say given the last 20 words spoken. Thus, Y was a discrete random variable taking 5000 values. There were 20 predictor variables $X_1, X_2, \cdots, X_{20}$, each of which represented one of the preceding 20 words, and was, therefore, a discrete random variable taking 5000 values. Note that some of the preceding 20 words may be from earlier sentences or even earlier paragraphs or chapters.

We constructed a tree with pylonic questions using the tree-growing and set-construction algorithms of Section II, with two minor changes. First, instead of adding one word at a time to the set S in Step 2 of set construction, we added one group of words at a time. The groups were predefined to represent grammatical classes such as days of the week, months, nouns, verbs, etc. The classes were of differing coarseness and overlapped; many words belonged to several classes. For example, "July" belonged to the class of months, the class of nouns, as well as its own class, the class of "July," which contained only that one word. Adding groups of words simultaneously allows set construction to proceed more rapidly. Since individual words may be discarded from the set, little harm is done by inserting words several at a time.

Second, since this was only a pilot experiment, we limited the tree to 10 000 leaves. To ensure even growth, the nodes were processed in order of size.

The adequacy of a language model may be assessed by its ability to predict unseen test data. This ability is measured by the perplexity of the data given the model [1]. Put simply, a script with a perplexity of p with respect to some model has the same entropy as a language having p equally likely choices in all contexts. Clearly, the lower the perplexity, the better the model.

Ignoring those 21-grams in the test data which ended in the generic "unknown" word, the perplexity of the test data with respect to the tree was 90.7. This compares to a perplexity of 94.9 with respect to an equivalent trigram language model. Although the tree was by no means fully grown, it still outperformed the trigram model. The difference in perplexity, however, is not great.

A bigger difference is evident in the numbers of very bad predictions. Speech recognition errors are more likely to occur in words given a very low probability by the language model. Using the trigram model, 3.87 percent of the words in the test data were given a probability of less than 2^{-15}. (It is convenient to work in powers of 2 when performing perplexity and entropy calculations.) In the case of the tree, only 2.81 percent of the test words had such a low probability. Thus, the number of words with a probability of less than 2^{-15} was reduced by 27 percent, which could have a significant impact on the error rate of a speech recognizer.

The improvement of the tree over the trigram model was obtained despite the fact that the tree was incomplete and comprised substantially fewer distinct probability distributions than the trigram model. The tree has one distribution per leaf—10 015 in all. The trigram model, on the other hand, has at least one distribution per unique bigram in the training text, and there were 796 000 of them. Apparently, the vastly greater resolution of the trigram model is insufficient to compensate for its other weaknesses vis-à-vis the tree-based model.

Although the tree has 10 015 distributions as compared to 796 000 for the trigram model, the storage necessary for them is about the same. Many of the trigram distributions have very few nonzero entries, which is not true for the tree distributions.

We also created a language model which combined the tree and the trigram. The combined probabilities were obtained as

$$\Pr\{w_{21} | w_1, w_2, \cdots, w_{20}\} = \tilde{\lambda}_r \tilde{q}_r(w_{21}) + (1 - \tilde{\lambda}_r) \Pr\{w_{21} | w_{19}, w_{20}\} \quad (12)$$

with $\tilde{\lambda}_r$ determined from independent training data as discussed in [1]. Here r denotes the leaf corresponding to the sequence $w_1, w_2, \cdots, w_{20}$; $\tilde{q}_r(w_{21})$ and $\Pr\{w_{21} | w_{19}, w_{20}\}$ denote the tree and trigram probability estimates, respectively. The weight $\tilde{\lambda}_r$ was a function of the leaf only, and lay between 0 and 1 as always.

The perplexity of the test data with respect to the combined model was 82.5—13 percent lower than the trigram perplexity and 9 percent lower than the tree perplexity. Additionally, 2.73 percent of the correct words had a probability of less than 2^{-15}. Thus, the number of words with a probability of less than 2^{-15} was only slightly less than with the tree alone, but was almost 30 percent less than obtained with the trigram model on its own.

For interest, we tabulated the depths of the 10 014 pylons in the tree. The results are shown in Fig. 4. It can be seen from Fig. 4 that roughly one-half of the pylons represented a single elementary question, while the other half represented composite questions of varying complexity. This supports the argument for composite questions at the nodes.

We also tabulated how often each predictor was the subject of a question. The 10 014 pylons contained 22 160 questions in total; their subjects are tabulated in Fig. 5. As would be expected, most of the questions are directed toward the more recent words; these being more useful for determining the likely part of speech of the next word to be uttered. Earlier words have not been ignored, although they have been interrogated somewhat less; they are useful for determining the likely semantic category of the forthcoming word. Certainly, it is not the case that all questions are directed toward the two most recent words w_{19} and w_{20}; this underlines the limitations of the trigram model, and the potentially useful information it discards.

Depth	Number Of Pylons
1	5052
2	2267
3	1197
4	568
5	337
6	197
7	126
8	90
9	60
10	29
11	22
12	21
13	11
14	4
>14	33

Fig. 4. Histogram of the depths of 10 014 pylons.

Subject	Number Of Questions
w_{20}	3793
w_{19}	6451
w_{18}	5147
w_{17}	2501
w_{16}	1208
w_{15}	615
w_{14}	370
w_{13}	284
w_{12}	234
w_{11}	188
w_{10}	184
w_9	181
w_8	160
w_7	138
$w_1 - w_6$	706

Fig. 5. Histogram of the question subjects.

In summary, the tree-based language model provided a lower perplexity and fewer very bad probabilities than an equivalent trigram model. However, the combined trigram-tree model had appreciably lower perplexity than either model on its own. Thus, the most effective use of a tree-based model may be as an adjunct to a trigram model, rather than a replacement for it.

IV. Epilogue

A good language model ought to have the property that the probability assigned to real text is high, and the probability of nonsensical text is low. With this in mind, we probed the properties of the trigram and tree-based models by generating some text using the models. Words following quotation marks or a period were selected at random in accordance with their probabilities. In all other contexts, the most probable word was selected. The generic "unknown" word was excluded from the allowable vocabulary. The initial word history was selected at random from real text.

The excerpts below are included mainly for the reader's amusement, and are not claimed to be typical. The generated text reflects the language models' heritage in romantic novels. The punctuation has been left exactly as it was generated. Each punctuation symbol is considered to be a word and has a probability of being the next "word" spoken like any other word.

The following paragraph was obtained with the trigram model.

If you don't have to be a good deal of the world." "I said.
She was a good deal of the world. But the fact that the only one of the world. When the first time in the world.

Less monotonous are the following paragraphs obtained with the tree-based model.

"What do you mean?"
"I don't know. You know," said the man.
"Is it?" he asked.
"You know," said the man.
"They are not not to be a good idea. The first time I was a good idea." She was a good idea.

"Certainly," I said.
"What's the matter?"
"May I be able to get the money."
"Well," said the man. Scott was a good idea.

"Mrs. King," Nick said. "I don't know what I mean."
"Take a look at the door. He was a good idea. I don't know what I mean. Didn't you know," he said.

It is the collective unbiased opinion of the authors that these paragraphs are at least as stimulating as the average romantic novel.

The trigram sentences consist of very reasonable 3-grams, but the limited memory of the trigram model results in stilted 4-grams, 5-grams, etc., and hence leads to meaningless sentences. Quotation marks are unmatched because there is no mechanism to remember the number of unmatched quotes.

The tree-generated sentences exemplify the effects of the longer 20-word memory. Punctuation is vastly improved, and repetition is much reduced. Some of the component 3-grams, however, are not as reasonable as before. "Are not not" and "not not to" are pretty unusual constructions, but trigrams like those are not not to be found in scientific journals.

Acknowledgment

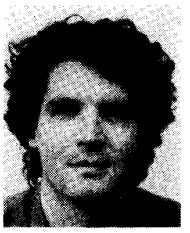
We would like to thank the other members of the Speech Recognition Group at the IBM Research Center for their help and encouragement. We are also indebted to the

American Printing House for the Blind who provided the books and magazines used in the experiments.

REFERENCES

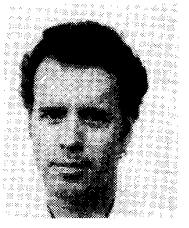
[1] L. R. Bahl, F. Jelinek, and R. L. Mercer, "A maximum likelihood approach to continuous speech recognition," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-5, pp. 179–190, Mar. 1983.
[2] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees.* Monterey, CA: Wadsworth, 1984.
[3] R. G. Casey and G. Nagy, "Decision tree design using a probabilistic model," *IEEE Trans. Inform. Theory*," vol. IT-30, pp. 93–99, Jan. 1984.
[4] C. R. P. Hartmann, P. K. Varshney, K. G. Mehrotra, and C. L. Gerberich, "Application of information theory to the construction of efficient decision trees," *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 565–577, July 1982.
[5] F. Jelinek, "Continuous speech recognition by statistical methods," *Proc. IEEE*, vol. 64, pp. 532–556, Apr. 1976.
[6] ——, "The development of an experimental discrete dictation recognizer," *Proc. IEEE*, vol. 73, pp. 1616–1624, Nov. 1985.
[7] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 1983.
[8] J. M. Lucassen and R. L. Mercer, "An information theoretic approach to the automatic determination of phonemic baseforms," in *Proc. 1984 IEEE Int. Conf. Acoust., Speech, Signal Processing*, San Diego, CA, Mar. 1984, pp. 42.5.1–42.5.4.
[9] H. J. Payne and W. S. Meisel, "An algorithm for constructing optimal binary decision trees," *IEEE Trans. Comput.*, vol. C-26, pp. 905–916, Sept. 1977.
[10] C. E. Shannon, "Prediction and entropy of printed English," *Bell Syst. Tech. J.*, vol. 30, pp. 50–64, Jan. 1951.

Lalit R. Bahl (S'66–M'68) is an ex-coding theorist who works for IBM on acoustic and language models for speech recognition.



Peter F. Brown is an ex-speech recognition researcher who works for IBM on automatic language translation.

Peter V. de Souza is an ex-biostatistician who works for IBM on acoustic and language models for speech recognition.



Robert L. Mercer (M'83) is an ex-physicist who works for IBM on acoustic and language models for speech recognition.