

---

# CSE 254 (Spring 2003)

## “Growing N-gram Trees for Language Modeling”

---

Dustin Boswell  
*dboswell [at] cs ucsd edu*

### Abstract

We implement variable n-grams using a word-tree data structure, where nodes represent sequences of words (“contexts”) and store how often that context appeared in a training corpus. We build the tree by growing it from the root up. Unlike other methods, there is no pruning step. Instead, we used the simple heuristic of maintaining a priority-queue of candidate leaves, sorted by how often those contexts occurred in the training text. The most popular leaves are then added to the tree, and this process repeats until a specified memory limit is reached. In this way, the tree was able to make branches for longer sentence fragments like ‘‘across the street from the’’ while saving the space from storing uncommon ones. We tested our system on samples from the North American News Text. Training/testing perplexities were comparable to that of a standard trigram model, and performed better in some cases.

## 1 Introduction

*Language Modeling* is the field of modeling how text is generated, so that we can assign a “likelihood” to a given string of words. For example, the string ‘‘he went home’’ is more likely than ‘‘abacus kindly flew’’. A typical application is in voice recognition, where a language model can help the system rank a set of candidate sentences by how likely they would have been said.

Formally, we consider a string of words  $W = w_1 \dots w_T$ . We are interested in creating an expression  $P(w_{j+1}|w_1 \dots w_j)$  - a probability distribution over the vocabulary set (of size  $|V|$ ), given the history of words. We sometimes refer to this as simply  $P(w|h)$ . Given this language model, the “likelihood” of a string of words can be calculated as  $P(W) = \prod_{j=1}^T P(w_j|h_j)$ .

## 2 Evaluation Criteria

If we have a language model  $P(w|h)$ , how do we measure how good it is, or if it’s better than another model? The ultimate measure of success is how well it improves the application it was made for. In speech recognition, the better language model

is the one that results in a lower word error rate for the recognizer. Unfortunately for a language model designer, it is difficult to 1) get a speech recognition system, 2) interface your model to the system, and 3) run the system on test speech data to observe the error rate.

Instead, a widely-accepted measure of language model success is obtained by measuring the model's *perplexity* on a text corpus. Perplexity  $P$  is related to the *average entropy*  $H$ :

$$\begin{aligned} H &\approx -\left(\frac{1}{T}\right) \lg(P(W)) && \text{for large } T \\ &= -\left(\frac{1}{T}\right) \sum_{\text{words } w} \lg(P(w|h)) \\ P &\equiv 2^H \end{aligned}$$

Thus, the model is assigned a “penalty” of  $\lg(P(w|h))$  for each word that appears. At each word, the model must essentially divide a total probability of 1.0 among each word in the vocabulary. Better models will have given higher probability to the words that actually came up, and will have a lower perplexity.

Perplexity can also be thought of as the “branching factor” of the model - the (geometric) average number of words that the system thinks may come next in the sentence. Typical values for  $P$  are in the range [50,300] depending on the strength of the model, the size of the vocabulary, and the diversity of the text - a set of documents from a limited domain is more “predictable”. Thus to compare two models, we fix the vocabulary and compute their perplexities for a given test set. Again, lower perplexity is better.

### 3 Previous Work

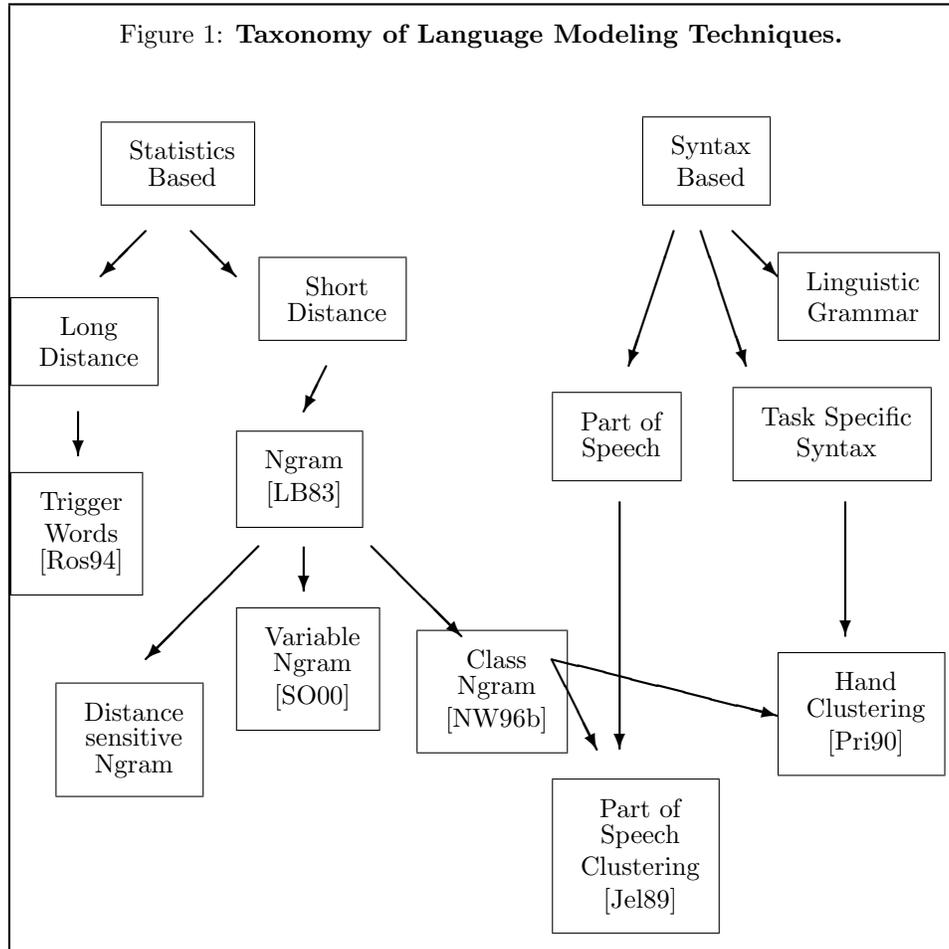
There are two classes of methods for constructing a language model. In a *statistical approach*, a training corpus is used to learn frequencies, co-occurrences, etc... of the data. In a *syntax-based approach*, there is usually no training corpus. Instead, an explicit model is created by the programmer based on his knowledge of the language or task syntax. A taxonomy of recent approaches can be seen in Figure 1.

On the statistics side, the most studied methods have been *n-gram* models. Simply put, they approximate an infinite history with a  $N - 1$  word history:

$$\hat{P}(w_{j+1} | w_j \dots w_{j-\infty}) = P(w_{j+1} | w_j \dots w_{j-N+1})$$

For a trigram model using a vocabulary of  $|V| = 10,000$  words, we see there are  $|V|^3 = 10^{12} = 1$  trillion possible trigrams  $P(w_3|w_1, w_2)$  to learn. In practice, only a vast minority of these would ever come up in a training corpus. But this leaves us with a problem - how do we estimate trigram probabilities from trigrams we've never seen? We can't just set them to 0 since obviously that's not the correct value (given that we see the trigram in the test data). Indeed, constructing an n-gram model is riddled with these types of “smoothing” issues. A complete four-gram would be a ridiculous feat, and so far it seems that no-one has seriously tried them.

However, instead of training an n-gram model for some fixed  $n$ , *variable n-grams* are used to store the n-grams in a tree-structure, so that n-grams of arbitrary length can be learned. Typically, a full tree of depth  $N_{max} \approx 5$  is grown, and then rare/un-useful branches are pruned. Alternatively, we can simply specify a memory limit,



and allow the tree to *grow* to maximum capacity. (This is the approach we take in this paper.)

N-grams are clearly a short-distance method - any correlations outside of the  $n$ -word window cannot be learned. Extensions to n-grams include *gappy n-grams* that allow words to be skipped, and *distance-sensitive n-grams* that use an n-gram with a fixed distance  $d$  between words. Another way to extend n-grams to further distance is to reduce the effective vocabulary size by clustering the vocabulary words into classes, and then using a *class based n-gram*. The classes may be hand-selected, for example: *part of speech* clustering (although this requires a part-of-speech tagger), or derived automatically. If the number of classes is small, n-grams for larger and larger  $N$  become computationally feasible. In [vSW94], a model was created to find common phrases such as “in the morning”, creating new words from those phrases. Unfortunately, the attempts were not very successful.

Even with methods like class-based n-grams, these short-distance methods can only be extended to perhaps a 10-word window. Instead, *long-distance* methods consider the history of the entire document currently being read. One approach is to use *trigger words*. For example, the word `bond` might have a higher likelihood given that `stock` preceded it in the document, than had `stock` not been in the document

history. In [Ros94], they report a decrease in perplexity from 170 (standard trigram model) to 153 when combining it with a trigger word model.

The problem with these methods is that they have no understanding of syntax or semantics. The trigger word `stock` would increase the probability of the word `bond` in the context “`stock ... The weather was ____`” despite `bond` having no meaningful place there.

Instead, *syntax-based methods* use grammatical analysis of (say) English to create models for that language. Strict grammar may not be as useful, since continuous speech is often grammatically incorrect. And simple *part of speech* models may not capture enough information to be useful.

## 4 Estimating Probabilities from Observed Frequencies

Every statistical method is faced with the problem of how to estimate  $P(w|h)$  from training data. We use the term  $h$  to refer generally to the history (also called the “context”) the system is making use of. For  $n$ -grams,  $h$  is a sequence of the  $n - 1$  previous words. For distance-sensitive bigrams,  $h$  refers to the word that came some fixed distance  $d$  before  $w$ .

Let  $c(h)$  be the number of times that context appeared in the training corpus of size  $T$ , and  $c(hw)$  be the number of times  $h$  was also followed by  $w$ . The maximum likelihood estimate is then

$$P(w|h) = \frac{P(hw)}{P(h)} \approx \frac{c(hw)/T}{c(h)/T} = \frac{c(hw)}{c(h)}$$

## 5 Storing Observed Frequencies in Trees

For  $n$ -grams, a tree (or *trie*) is a very natural and efficient data-structure for storing observed counts  $c(h)$ . (See figure 2 for an example.) Each node in the tree represents a history of  $k$  words, where  $k$  is the depth of the node. The root node corresponds to no history, and each child of a given node corresponds to a history with one more word than the parent. The edges are labeled with the word that is being added to the history. Each node also maintains a “traffic count”  $t(h_k)$  of how many times  $h_k$  occurred in the training corpus. (After training,  $t(h_k) = c(h_k)$  if the node exists.) Given these counts, there is an implied probability distribution over the next word ( $P(w|h_k)$ ) at each node  $h_k$ . (We address the issue of how to estimate probabilities for words that are not a child of  $h_k$  later.) And so the structure of the tree (and the associated probability distributions) are learned from the training corpus.

We should mention that there are two types of trees: *forward trees* and *backward trees*. In a forward tree (which we will refer to as just “tree”), going further down the tree corresponds to adding more words to the “end” of the history (as you would normally in reading a sentence). In a backward tree, each step down the tree corresponds to adding words to the front of the history. We use forward trees in this paper.

## 6 Notation

The training corpus consists of a vector of words  $\{w_j\}$ . For a given word  $w_j$ ,  $h_k$  refers to the  $k$  words  $w_{j-k} \dots w_{j-1}$  comprising the “history” of word  $j$ . Since each node in the tree corresponds to a history, we use the term  $h_k$  to refer both to the

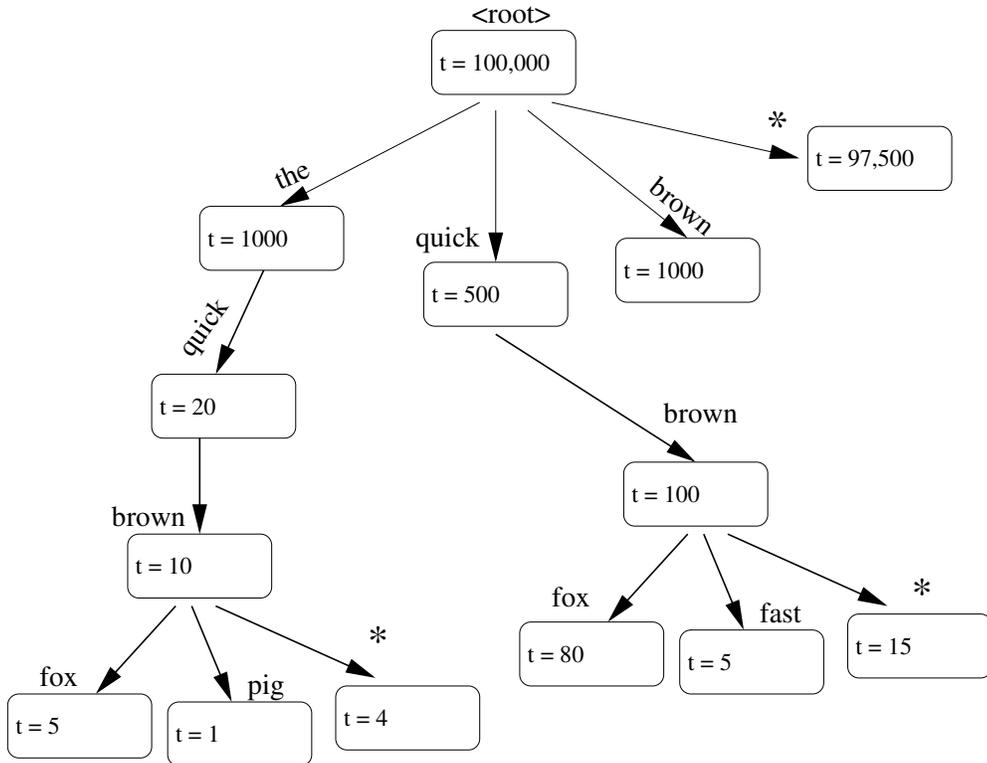


Figure 2: A hypothetical forward-tree for the vocabulary { **the**, **quick**, **fast**, **brown**, **fox**, **pig** }. A \* denotes all “other” words, so that each node’s count is the sum of its children ( $t(n) = \sum t(ch(n))$ ).

string of  $k$  words, and to the implied node in the tree (if it exists). Let  $ch(h_k)$  be the “children” of  $h_k$  (the set of words  $w$  for which  $h_k : w$  is also a node).

## 7 Problems with Zero-frequency events

What do we estimate as the probability for a sequence  $h : w$  we have never seen ( $c(h : w) = 0$ )? We cannot simply assume  $P(w|h) = 0$ , since that would cause our perplexity (which includes the term  $-\lg(P(w|h))$ ) to be infinite in the event  $h : w$  appears in the testing set. The reason  $c(hw)$  was zero might be because the true probability is  $P(w|h) \ll \frac{1}{c(h)}$ , so that  $c(hw)$  is observed to be zero for most training sets. Simply put, the issue is that the training corpus is only a sparse sample of the language, and our probability estimates (especially at the leaves of the tree) are not as reliable as we would hope. But there is no way around it: we must assign a positive probability for every combination  $P(w|h)$ .

To do this, we must shift some of the probability mass from the seen events to the unseen events. This is called *discounting*. The problem now becomes “how much probability do we assign to unseen events?” There is no clear answer to this, although there are many reasonable techniques for discounting.

## 8 Discounting Techniques

Suppose we encounter a history  $h_k$  that takes us to the corresponding node  $h_k$  in the tree. Recall  $t(h_k)$  is the number of times that history has occurred, and  $|ch(h_k)|$  is the number of child nodes. We are interested in computing some *discount factor*  $d$  such that the probability of an unseen word (a word that is not a child of  $h_k$ ) is  $1 - d$ . (How that probability is then divided up among those unseen words is a separate problem, addressed by *backing-off*)

### 8.1 Method A

This method simply sets  $d = t(h_k)/(t(h_k) + 1)$ . So the unseen word probability is  $1 - d = 1/(t(h_k) + 1)$ . This method effectively creates a new child node  $n'$  with count  $t(n') = 1$  that represents all the unseen words (and increases the count of the parent  $t(h_k)$  by 1).

### 8.2 Method B

This method can be described as “stealing one count away from each child”. That is, a new child  $n'$  representing the unseen words is created, and that node is given a count of  $t(n') = |ch(h_k)|$ . Each original child has its count decremented by 1. The probability of unseen words is thus  $1 - d = |ch(h_k)|/t(h_k)$ .

### 8.3 Method C (Witten-Bell)

This is one of the more popular methods, and the method used in this paper. This method can be described as “adding a count of 1 to the unseen node, for each child”. This generalization of method A sets  $d = t(h_k)/(t(h_k) + |ch(h_k)|)$ , and the unseen probability is  $1 - d = |ch(h_k)|/(t(h_k) + |ch(h_k)|)$ .

### 8.4 Good Turing

Let  $n_1$  be the number of children  $c$  that have count  $t(c) = 1$ . The unseen probability is then defined to be  $1 - d = n_1/t(h_k)$ , and each original child’s count  $t(c)$  is changed to  $t'(c) = (t(c) + 1) * \frac{n_1 t(c) + 1}{n_1 t(c)}$ . Clearly, the counts will no longer be integral. That these new counts still add up to  $t(h_k)$  is not obvious. The proof can be seen in [Kat87].

### 8.5 Katz Back-off

Notice that the above methods all decrease the probability of children with count  $t(c) \geq 1$  and increase the probability of virtual children with  $t(c) = 0$ . This can be generalized. For example [Kat87] treats all counts of  $t(c) \leq g$  as “unreliable” and smoothes the probability of these cases by combining this probability estimate with estimates from sub-histories (eg.  $h_{k-1}$ ). (To do this, we would have to start at the root of the tree again, and go down a path starting with one less word in the history.) The choice  $g \approx 5$  worked well in their case.

## 9 Implementing Backing-off

Using sub-histories to aid in averaging counts that are deemed too small (and therefore unreliable), is referred to as *backing off*. While combining overlapping informa-

tion sources in this way is definitely a “black art” of sorts, [Jel97] has shown how to compute the weighting coefficients in the optimal way for the training set.

For this system, we implemented the following “first-fit” back-off procedure: Given a history  $h_k$ , find that node in the tree (if it doesn’t exist, try shorter and shorter histories until we do find it). If our current word  $w$  is a child of the node  $h_k$ , then we output the usual maximum likelihood estimate  $t(h_k : w)/t(h_k)$  (but discounted of course). If  $w$  is not a child, we try the history  $h_{k-1}$  instead.

While that is the gist of it, there are a couple multiplicative factors that must be introduced to insure that the probability distribution  $P(w|h_k)$  sums to 1 over all words. (We will use the notation  $P_{h_k}(w = w^*)$  instead, for the next section.)

Formally, we are defining the probability distribution as follows:

$$P(w|h_k) \equiv \begin{cases} \frac{t(\mathbf{h}_k : \mathbf{w})}{t(\mathbf{h}_k)} * d(h_k) & : w \in ch(h_k) \\ P(w|h_{k-1} \text{ AND } w \notin ch(h_k)) * (1 - d(h_k)) & : w \notin ch(h_k) \end{cases}$$

where  $d(h_k)$  is the Witten-Bell discount factor associated with node  $h_k$ . The second case expression may look strange at first. The clause **AND**  $w \notin ch(h_k)$  is needed to make sure  $P(w|h_k)$  sums to 1. If we just used the simpler  $P(w|h_{k-1})$  in the second case, we would sum to less than 1. What we need is a way to convert  $P(w|h_{k-1})$  to  $P(w|h_{k-1} \text{ AND } w \notin ch(h_k))$ , since the latter correctly sums to 1 over  $w \notin ch(h_k)$ . The correct adjustment factor we need is:

$$P' = P(w|h_{k-1} \text{ AND } w \notin ch(h_k)) = P(w|h_{k-1}) * \left( \frac{1}{\sum_{w' \notin ch(h_k)} P(w'|h_{k-1})} \right)$$

A quick check easily shows that  $\sum_{w \notin ch(h_k)} P' = 1$ . Thus our recursive definition is complete.

## 10 Building the Tree - Going from Previous Work

Previous work [SO00] has typically been done with backward trees and is created using an algorithm of the form shown in figure 3.

Figure 3: **Typical Tree Pruning Algorithm**

```
Create tree of depth N-max
While size(tree) > SYSTEM_MEMORY:
    Prune the least-useful branch
```

This method has the undesirable features of requiring space for the full tree ahead of time, and having to choose another parameter: N-max. It would be desirable to have an algorithm that grows a tree from the root up, to an arbitrary depth (see figure 4).

Notice that this algorithm also has a pruning step. There are various ways to measure how “useful” a branch is. For backward trees (where each node defines an entire

Figure 4: **Typical Tree Growing Algorithm**

```
Create tree of depth 0
While  $size(tree) < SYSTEM\_MEMORY$ :
    Add children to every leaf.
    Prune least useful branches.
```

probability distribution), the typical measure is to compute the KL-Divergence of that node’s distribution with that of the parent distribution [Kne96] [SO00]. That is, if the probability  $P(w|h_k)$  is very different from  $P(w|h_{k-1})$ , then the context  $h_k$  supposedly tells us different (more) information about  $w$  than  $h_{k-1}$  did, and so should be kept in the tree. For forward trees, however, there is no explicit probability distribution, so such analysis is not possible.

A similar method to (figure 4) was used in [NW96b] and I will continue with it. Unfortunately, this algorithm requires an “exploration memory” of potentially unlimited size. That is, as the tree becomes large, the step in (figure 4) of adding children to every leaf may temporarily (but grossly) overshoot our memory limit.

Instead, we introduce an algorithm that maintains a fixed-size priority queue of “candidate children”, sorted by the number of times they are “requested” in scanning the training corpus. At the end of a scan, the top fraction (say 10%) of the queue is processed and added to the tree. This is repeated until the tree reaches a size limit. The algorithm is shown in figure 5.

Figure 5: **Modified Tree Growing Algorithm**

```
Tree  $T$ ;
PriorityQueue  $Q[EXPLORATION\_SIZE]$ ;

While  $size(T) < MAX\_TREE\_SIZE$ :
    for each word in the corpus  $w_j$  (along with its history  $h_\infty$ )
        for  $k = 0$  to  $maxDepth(T)$  ...
            if node  $n(h_k)$  exists and  $n(h_k w_j)$  doesn't,
                 $Q.request((n(h_k w_j)))$ 

     $Q.process( 10\% * EXPLORATION\_SIZE )$ 
     $Q.clear()$ 
```

## 11 Experiments

The training text used was the “North American News Text” (See figure 6.) For all experiments the vocabulary was set as the 64K most common words in the training set. This results in unknown words (which are all mapped to the same symbol UNK) occurring less than 1% of the time, typically. We adopt the convention that a language model is allowed to use UNK as part of the history, but for testing perplexity, instances of  $w = UNK$  are skipped.

Name	# Words	Description
LW94-97	78MW	Los Angeles Times, Washington Post
NY94-96	280MW	New York Times News Syndicate
RE94-96	106MW	Reuters News Service (General)
RF94-96	29MW	Reuters News Service (Financial)
WS94-96	61MW	Wall Street Journal
sample3	0.5MW	last $10^X$ lines of each file ( $\approx 500 * 10^X$ words total)
sample4	5MW	
sample5	50MW	
sampleX	$500 * 10^X$ W	

Figure 6: **Text Corpora used.** (MW is “million words”)

### 11.1 Looking at a sample Word Tree

Figure 7 shows a portion of the Word Tree obtained by running the growing algorithm on the *sample5* corpus. Each line represents a node of the tree, and is written in the form `WORD traffic child_traffic num_children`. The subtree of any node is written directly after that node, with indentation starting one level deeper. That is, the number of tabbings before a line is the depth of that node.

The `traffic` is simply the number of times that history (the sequence of words starting from the `<ROOT>` to the current child) occurred in the corpus. `traffic(<ROOT>)` is simply the number of words in the corpus - in this case 53 million.

The `child_traffic` is the sum of the `traffic`'s of each child node. In general,  $\text{child\_traffic}(n) \leq \text{traffic}(n)$ , but will be strictly less when one or more child nodes were not kept (because the child node did not occur enough times for the algorithm to grow it).

`num_children` is how many child nodes directly follow this one. Obviously,  $\text{num\_children} \leq |\text{Vocab}|$ , but in practice will always be much less. In fact, this tree shows  $\text{num\_children}(\text{<ROOT>}) = 32245$  while the vocabulary size is  $64K$ . Apparently, about half of the vocabulary words occurred so infrequently they didn't even merit an initial node. Not to worry, from  $\text{child\_traffic}(\text{<ROOT>}) / \text{traffic}(\text{<ROOT>}) = 98.8\%$ , we see those infrequent words occurred only 1.2% of the time.

The goal of this project was to build a system that maintains the power of the typical trigram model, but with the flexibility to shift its resources to n-grams of greater length where they are important. From the examples in the figure, we see some evidence of this. For instance, the 5-word phrase `across the street from the` is explicitly stored in the tree (and it occurred 77 times in the corpus).

Another example to point out is the subtree starting with `ronald`. The most common following words included `regan` and `goldman`. Interestingly, the 2-word history `ronald lyle` is a perfect predictor (49/49) of `goldman` in this news corpus.

Another long phrase discovered is `adjusted annual rate of 1 POINT .`<sup>1</sup> For comparison, the shorter subtree `of...` is also shown. From this long phrase, the maximum likelihood estimate of  $P(\text{POINT} | \text{adjusted annual rate of 1})$  is

$$P(\text{POINT} | \text{adjusted annual rate of 1}) =$$

<sup>1</sup>The `POINT` token was generated whenever a period followed a digit, and was followed by a non-space.

Figure 7: Portions of the Word Tree obtained from the sample5 corpus

NODE traffic child\_traffic num\_children

-----

```
<ROOT> 53771683 53156023 32245
  ...
  across 8514 6907 23
    america 114 0 0
    canada 48 0 0
    ...
    the 5310 3268 20
      board 388 145 2
        COMMA 89 0 0
        PERIOD 56 0 0
      ...
      street 284 198 3
        COMMA 37 0 0
        PERIOD 43 0 0
        from 118 77 1
          the 77 0 0
    ...
  adjusted 1613 887 7
    annual 271 235 1
      rate 235 216 1
        of 216 67 1
          1 67 55 1
            POINT 55 0 0
    ...
  of 1198167 1025437 2945
    1 24883 24749 16
      POINT 2455 2455 10
    ...
  having 8428 4478 30
    ...
    a 1190 90 2
      hard 50 46 1
        time 46 0 0
    ...
  ronald 1764 1064 8
    goldman 161 109 2
      ...
      lyle 49 49 1
        goldman 49 0 0
    reagan 480 201 3
      ...
      reagan's 125 0 0
        ...
        UNK 115 71 1
```

$$\frac{\text{traffic}(\text{adjusted annual rate of 1 POINT})}{\text{traffic}(\text{adjusted annual rate of 1})} = \frac{55}{67} \approx 0.82$$

But using a 2-word history (as a trigram would), we have

$$P(\text{POINT} \mid \text{of 1}) = \frac{\text{traffic}(\text{of 1 POINT})}{\text{traffic}(\text{of 1})} = \frac{2455}{24883} \approx 0.10$$

which is a lot different. Admittedly, this example is very anecdotal, but it shows the type of situation that a word tree can exploit.

## 11.2 Most common n-grams

It is also interesting to see which n-grams are the most common in the tree, and whether larger  $n$  allows it to capture useful information. Figure 8 shows the most common n-grams for a tree grown to 300,000 nodes (trained on various amounts of text). For small amounts of training (400KW), the grams  $n = 1, 2$  (“the” and “of the”) are intuitively “correct”, but for  $n \geq 3$ , we start to see the effects of over-fitting. However, this isn’t to say that all the longer n-grams are the effects of over-fitting. For example, the ones seen in figure 7 are much more sensible.

n	Trained on 400KW	4MW	15MW	Comment
1	the	(← same)	(← same)	
2	of the	(← same)	(← same)	
3	0 0 0	(← same)	1 9 9	(each digit is a word)
4	COMMA 0 0 0	(← same)	(← same)	
5	0 COMMA 0 0 0	(← same)	(← same)	
6	los angeles times washington post news	(← same)	[none]	occurred $\geq 100$ times in 400KW

Figure 8: Most common n-grams in WT(300,000) tree. (LW94 corpus).

## 11.3 Perplexity Results

We implemented the modified tree-growing algorithm (figure 5) in C++, and refer to this “Word Tree” system as WT(# nodes), where the tree was allowed to grow to that many nodes. To compare results, we used the CMU Language Modeling Toolkit [CR97], which implements smoothed n-grams (we set  $n=3$ ) in C. Unfortunately, our C++ implementation is currently very space inefficient: storing 250,000 nodes takes 200MB of memory. (The CMU implementation has amazingly squeezed it down to about 6 bytes per n-gram.) Thus, for fair comparison, we restricted the CMU system to use the same number of n-grams, as our system has nodes in the tree. The CMU system was also set to use Witten-Bell discounting (as our system does).

Figure 9 shows the perplexity for various training/testing combinations. The results are mixed. For the smallest training set (sample3), we see that the WT system had a lower self-perplexity (109 as compared to CMU’s 131), but slightly worse results when tested on sample4 (314 as compared to CMU’s 310). When trained on sample4, however, the WT system had better training *and* testing perplexity. Trained on sample5, though, resulted in worse performance for both training and testing. This might suggest that the current memory limit of 250,000 nodes is “tuned” to training sets of the size of sample4: less training resulted in over-fitting, too much training data was unable to be completely learned.

	training set ↓	testing set →			
		sample3	sample4	sample5	NY96
CMU	sample3	131	310		
	sample4		246	284	
	sample5			263	292
WT	sample3	109	314		
	sample4		233	278	
	sample5			274	305

Figure 9: **Perplexity Results for WT(250,000) vs. CMU**

## 12 Conclusions, Comments, and Future Work

We implemented variable-length n-grams in a tree structure, and grew the tree from the root up, adding the most frequent leaves (histories) from each scan of the training corpus. We then compared the perplexity results against a trigram model with the same memory resources. The two systems performed roughly the same.

Inherently, variable-length n-grams are a generalization of trigrams. Trigrams are just a special case where the growing algorithm refuses to grow paths of length  $> 3$ . And so we should expect variable-length n-grams to perform no worse than fixed-length n-grams.

Our implementation of the word tree was very space-inefficient, and this restricted our experimentation to word trees no bigger than 300,000 nodes. Also, we used a simple growing criteria: histories that occurred the most often were added to the tree. A better criteria would be one that assesses how much “excess information” growing a particular branch would bring. However, this would severely complicate the growing algorithm, and so we chose not to explore that issue here.

Despite mixed results, we still feel variable-length n-grams offer advantages over standard n-grams. From the experiments, we saw that the tree was able to learn useful phrases such as ‘ ‘ across the street from the’ ’ . Given a very large training corpus (perhaps the internet?) and sufficient resources, we feel our method would significantly outperform any trigram or four-gram.

## References

- [BP98] A. Berger and H. Printz. Recognition performance of a large-scale dependency-grammar language model, 1998.
- [CR97] Philip Clarkson and Ronald Rosenfeld. Statistical language modeling using the CMU-cambridge toolkit. In *Proc. Eurospeech '97*, pages 2707–2710, Rhodes, Greece, 1997.
- [HAHR93] Xuedong Huang, Fileno Allewa, Mei-Yuh Hwang, and Ronald Rosenfeld. An overview of the sphinx-ii speech recognition system. In *ARPA Human Language Technology Workshop*, pages 81–86. Morgan Kaufmann, March 1993. published as Human Language Technology.
- [Jel89] Fred Jelinek. Self-organized language modeling for speech recognition. *Readings in Speech Recognition*, 1989.
- [Jel97] *Statistical Methods for Speech Recognition*. MIT Press, 1997.
- [JMBB77] Fred Jelinek, Robert L. Mercer, Lalit R. Bahl, and James K. Baker. A measure of difficulty of speech recognition tasks. Technical report, 94th Meeting of the Acoustic Society of America, 1977.

- [Kat87] Slava M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-35:400–401, March 1987.
- [Kne96] Reinhard Kneser. Statistical language modeling using a variable context length. In *Proc. Int. Conf. Spoken Language Processing*, pages 494–497, 1996.
- [LB83] F. Jelinek and R.L. Mercer. L.R. Bahl. A maximum likelihood approach to continuous speech recognition. *IEEE Journal of Pattern Analysis and Machine Intelligence*, 1983.
- [LRR93] R. Lau, R. Rosenfeld, and S. Roukos. Trigger-based language models: A maximum entropy approach. In *Proc. ICASSP*, volume Vol II, pages 45–48, April 1993.
- [NW96a] T. Niesler and P. Woodland. Combination of word-based and category-based language models. In *Proc. ICSLP '96*, volume 1, pages 220–223, Philadelphia, PA, 1996.
- [NW96b] T. Niesler and P. Woodland. A variable-length category-based n-gram language model. In *Proc. ICASSP '96*, pages 164–167, Atlanta, GA, 1996.
- [NWW98] T. Niesler, E. Whittaker, and P. Woodland. Comparison of part-of-speech and automatically derived category-based language models for speech recognition, 1998.
- [PPMR92] Stephen Della Pietra, Vincent Della Pietra, Robert Mercer, and Salim Roukos. Adaptive language modeling using minimum discriminant estimation. In *International Conference on Acoustics, Speech and Signal Processing*, pages 633–636, San Francisco, March 1992. Also published in *Proceedings of the DARPA Workshop on Speech and Natural Language*, Morgan Kaufmann, pages 103106, February 1992.
- [Pri90] Patti Price. Evaluation of spoken language systems: the atis domain. In *Proceedings of the third DARPA Speech and Natural Language Workshop*, 1990.
- [Ros94] R. Rosenfeld. *Adaptive Statistical Language Modeling: A Maximum Entropy Approach*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 1994.
- [Ros96] R. Rosenfeld. A maximum entropy approach to adaptive statistical language modeling, 1996.
- [Ros00] R. Rosenfeld. Two decades of statistical language modeling: Where do we go from here, 2000.
- [Sha51] C. Shannon. Prediction and entropy of printed english. Technical Report 30, Bell Systems, 1951.
- [SO00] M. H. Siu and M. Ostendorf. Variable n-grams and extensions for conversational speech language modeling. *IEEE Transactions on Speech and Audio Processing*, 8(1):63–75, 2000.
- [UPE] UPENN. Linguistic data consortium.
- [vSW94] E. volume, Suhm, and B. Waibel. Toward better language models for spontaneous speech, 1994.