Inference and Estimation of a Long-Range Trigram Model

S. Della Pietra   V. Della Pietra   J. Gillett
J. Lafferty[†]   H. Printz   L. Ureš

IBM T. J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598 USA

Abstract

We describe an implementation of a simple probabilistic link grammar. This probabilistic language model extends trigrams by allowing a word to be predicted not only from the two immediately preceeding words, but potentially from any preceeding pair of adjacent words that lie within the same sentence. In this way, the trigram model can skip over less informative words to make its predictions. The underlying "grammar" is nothing more than a list of pairs of words that can be linked together with one or more intervening words; this word-pair grammar is automatically inferred from a corpus of training text. We present a novel technique for indexing the model parameters that allows us to avoid all sorting in the M-step of the training algorithm. This results in significant savings in computation time, and is applicable to the training of a general probabilistic link grammar. Results of preliminary experiments carried out for this class of models are presented.

1 Introduction

The most widely used statistical model of language is the so-called *trigram model*. In this simple model, a word is predicted based solely upon the two words which immediately precede it. The simplicity of the trigram model is simultaneously its greatest strength and weakness. Its strength comes from the fact that one can easily estimate trigram statistics by counting over hundreds of millions of words of data. Since implementation of the model involves only table lookup, it is computationally efficient, and can be used in real-time systems. Yet the trigram model captures the statistical relations between words by the sheer force of numbers. It ignores the rich syntactic and semantic structure which constrains natural languages, allowing them to be easily processed and understood by humans.

Research supported in part by NSF and ARPA under grants IRI-9314969, N00014-92-C-0189 and N00014-91-C-0135.

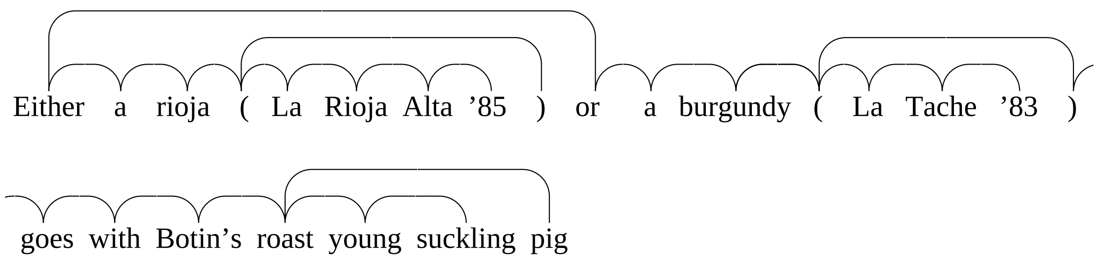
[†]Current address: School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213 USA, E-mail: `lafferty@cs.cmu.edu`

Probabilistic link grammar has been proposed as an approach which preserves the strengths and computational advantages of trigrams, while incorporating long-range dependencies and more complex information into a statistical model [LST92]. In this paper we describe an implementation of a very simple probabilistic link grammar. This probabilistic model extends trigrams by allowing a word to be predicted not only from the two immediately preceding words, but potentially from any preceding pair of adjacent words that lie within the same sentence. In this way, the trigram model can skip over less informative words to make its predictions. The underlying "grammar" is nothing more than a list of pairs of words that can be linked together with one or more intervening words between them. This paper presents an outline of the basic ideas and methods used in building this model.

Section 2 gives an introduction to the long-range trigram model and explains how it can be seen as a probabilistic link grammar. The word-pair grammar is automatically inferred from a corpus of training text. While mutual information can be used as a heuristic to determine which words might be profitably linked together, this measure alone is not adequate. In Section 3 we present a technique that extends mutual information to suit our needs. The parameter estimation algorithms, which derive from the EM algorithm, are presented in Sections 4 and 5. In particular, we present a novel technique for indexing the model parameters that allows us to avoid all sorting in the M-step of the training algorithm. This results in significant savings in computation time, and is applicable to the training of a general probabilistic link grammar. In Section 6 we present the results of preliminary experiments carried out using this approach.

2 A Long-Range Trigram Model

As a motivating example, consider the picture shown below. This diagram represents a *linkage* of the underlying sentence "Either a rioja ... suckling pig", as described in [ST91]. The important characteristics of a linkage are that the arcs, or *links*, connecting the various words do not cross, that there is no more than one link between any pair of words, and that the resulting graph is connected. Viewed probabilistically, we imagine that each word is generated from the bigram ending with the word that it is linked to on the left. Thus, the first right parenthesis is generated from the bigram (rioja, "(") while the word "suckling" is generated from the bigram (roast, young). The word "or" is generated from the bigram ($\perp$, Either), where $\perp$ is a *boundary word*.


Either a rioja (La Rioja Alta '85) or a burgundy (La Tache '83)
goes with Botin's roast young suckling pig

Another valid linkage would connect the first left parenthesis with the last right parenthesis, but this would preclude a connection between the words "Either" and "or" since the resulting links would cross.

To describe the model in more detail, consider the following description of standard trigrams. This model is viewed as a simple finite-state machine for generating sentences. The states in the machine are indexed by pairs of words. Adjoining the boundary word $\perp$ to our vocabulary, we suppose that the machine begins in the state $(\perp, \perp)$. When the machine is in any given state (w_1, w_2) it progresses to state (w_2, w_3) with probability $\mathtt{t}\,(\,w_3 \mid w_1\, w_2\,)$ and halts with probability $\mathtt{t}\,(\,\perp \mid w_1\, w_2\,)$, thus ending the sentence.

Our extended trigram model can be described in a similar fashion. Again states are indexed by pairs of words, but a state $s = (w_1, w_2)$ can now either halt, step, or branch, with probability $\mathtt{d}\,(\,\mathtt{halt} \mid s\,)$, $\mathtt{d}\,(\,\mathtt{step} \mid s\,)$, and $\mathtt{d}\,(\,\mathtt{branch} \mid s\,)$ respectively. In case either a `step` or a `branch` is chosen, the next word w is generated with the trigram probability $\mathtt{t}\,(\,w \mid w_1\, w_2\,)$. But in the case that `branch` was chosen for the state s, an additional word w' is generated from the *long-range* trigram distribution $\mathtt{l}\,(\,w' \mid w_1\, w_2\,)$.

For example, in generating the above linkage, the state indexed by $s = (\text{or}, \text{a})$ chooses to step, with probability $\mathtt{d}\,(\,\mathtt{step} \mid s\,)$, and the word "burgundy" is then generated with probability $\mathtt{t}\,(\,\text{burgundy} \mid \text{or a}\,)$. On the other hand, the state $s = (\perp, \text{Either})$ branches, with probability $\mathtt{d}\,(\,\mathtt{branch} \mid s\,)$, and from this state the words "a" and word "or" are then generated with probabilities $\mathtt{t}\,(\,\text{a} \mid \perp\ \text{Either}\,)$ and $\mathtt{l}\,(\,\text{or} \mid \perp\ \text{Either}\,)$ respectively.

This results in linkages, such as the one shown above, where every word is linked to exactly one word to its left, and to zero, one, or two words on its right. If we number the words in the sentence S from 1 to $|S|$, then it is convenient to denote by $\lhd i$ the index of the word which generates the i-th word of the sentence. That is, word i is linked to word $\lhd i$ on its left. For instance, in the linkage shown above we see that $\lhd 8 = 7$, $\lhd 9 = 4$, and $\lhd 10 = 1$. This notation allows us to write down the probability of a sentence as $P(S) = \sum_{\Lambda \in \mathcal{L}(S)} P(S, \Lambda)$, where $\mathcal{L}(S)$ is the set of all linkages of S, and where the joint probability $P(S, \Lambda)$ of a sentence and a linkage is given by

$$P(S,\Lambda) = \prod_{i=1}^{|S|} \mathtt{d}\,(\,d_i \mid w_i, w_{i-1}\,)\; \mathtt{t}\,(\,w_i \mid w_{i-1}\, w_{i-1}\,)^{\delta(i-1,\lhd i)}\; \mathtt{l}\,(\,w_i \mid w_{\lhd i}\, w_{\lhd i-1}\,)^{1-\delta(i-1,\lhd i)}\,. \tag{1}$$

Here $d_i \in \{\mathtt{halt}, \mathtt{step}, \mathtt{branch}\}$, $\delta(i,j)$ is equal to one if $i = j$ and is equal to zero otherwise, and the indices $\lhd i$ are understood to be taken with respect to linkage Λ. The indices $\lhd i$ determine which words are linked together, and so completely determine a valid linkage as long as they satisfy the no-crossing condition $\lhd j \le \lhd i$ whenever $i \le j$. In particular, specifying the indices $\lhd i$ determines the values of d_i, since d_i is equal to `halt`, `step`, or `branch` when $\sum_{j>i} \delta(i, \lhd j)$ is equal to zero, one, or two, respectively.

A full description of the model is best given in terms of a probabilistic pushdown automaton. The automaton maintains a stack of states s, where s is indexed by a word bigram, and a finite memory containing a state m. It is governed by a finite control that can read either `halt`, `step`, or `branch`. Initially the stack is empty, the finite memory contains the bigram $(\perp, \perp)$, and the finite control reads `step`. The automaton proceeds by carrying out three tasks. First, the finite control is read and a word is output with the appropriate distribution. If the control reads either `step` or `branch`, then word w is output with probability $\mathtt{t}\,(\,w \mid m\,)$. If the control reads `halt` then the automaton looks at the stack. If the stack is empty the machine halts. Otherwise, the state s is popped off the stack and word w is output with probability $\mathtt{l}\,(\,w \mid s\,)$. Second, the memory state is changed from $m = (w_1, w_2)$ to $m = (w_2, w)$. Third, the control is set to d with probability $\mathtt{d}\,(\,d \mid m\,)$, and state m is pushed onto the stack if the new setting is `branch`.

The probability with which this machine halts after outputting sentence S is precisely the sum $\sum_{\Lambda \in \mathcal{L}(S)} P(S, \Lambda)$ where $P(S, \Lambda)$ is given by equation (1).

In terms of link grammar, there is a natural equivalence between the values `halt`, `step`, and `branch` and three simple *disjuncts*, specifying how a word connects to other words. The value `halt` corresponds to a disjunct having a single (unlabeled) left connector, and no right connectors, indicating that a connection can be made to any word on the left, but to no word on the right. The value `step` corresponds to a disjunct having a single left connector and a single right connector, and the value `branch` corresponds to a disjunct having a single left connector and two right connectors. With this grammar, the probabilistic model described above is a simple variant of the general probabilistic model presented in [LST92].

In terms of phrase structure grammar, the constructive equivalence between link grammar and context free grammars given in [ST91] can be extended probabilistically. This shows how the above model is equivalent to the following standard probabilistic context-free model:

$$\begin{array}{ll} A^u_{v,w} \rightarrow B^u_{v,w} & \mathtt{d}(\,\mathtt{branch}\,|\,v,w\,) \\ A^u_{v,w} \rightarrow C^u_{v,w} & \mathtt{d}(\,\mathtt{step}\,|\,v,w\,) \\ A^w_{v,w} \rightarrow \epsilon & \mathtt{d}(\,\mathtt{halt}\,|\,v,w\,) \\ B^u_{v,w} \rightarrow C^x_{v,w}\; y\; A^u_{x,y} & \mathtt{l}(\,y\,|\,v\,w\,) \\ C^u_{v,w} \rightarrow z\; A^u_{w,z} & \mathtt{t}(\,z\,|\,v\,w\,)\ . \end{array}$$

Here x, y, z are vocabulary words with $x, y \neq \perp$ and A, B, and C are families of nonterminals parameterized by triples of words u, v, w. The corresponding rule probabilities are given in the second column. The start nonterminal of the grammar is $S = A^{-}_{-,-}$. This view of the model is unwieldy and unnatural, and does not benefit from the efficient link grammar parsing and pruning algorithms.

3 Inferring the Grammar

The probabilistic model described in the previous section makes its predictions using both long-range and short-range trigrams. In principle, we can allow a word to be linked to any word to its left. This corresponds to a "grammar" that allows a long-range link between any pair of words. The number of possible linkages for this grammar grows rapidly with sentence length: while a 10-word sentence has only 835 possible linkages, a 25-word sentence has 3,192,727,797 linkages (see Appendix A). Yet most of the long-range links in these linkages are likely to be spurious. The resulting probabilistic model has far too many parameters than can be reliably estimated.

Since an unrestricted grammar is impractical, we would like to restrict the grammar to allow those long-range links that bring the most improvement to the probabilistic model. Ideally, we would like to automatically discover pairs of words such as "(" and ")" with long-range correlations that are good candidates to be connected by a long-range link. We might find such pairs by looking for words with high mutual information. But if we imagine that we have already included all nearest neighbor links in our model, as is the case for the model (1), there is no point in linking up a pair of words L and R, no matter how high their mutual information, if R is already well-predicted by its immediate predecessor. Instead, we would like to find links between words that have the potential

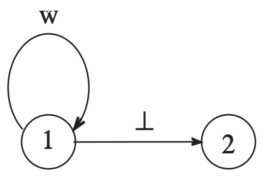


Figure 1: A bigram machine

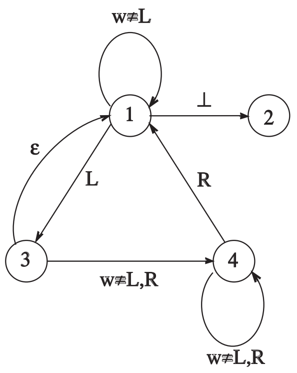


Figure 2: An (L, R) machine

of improving a model with only short-range links.

To find such pairs we adopt the following approach. Let V be the language vocabulary. For each pair $(L, R) \in V \times V$, we construct a model P_{LR} that contains all the bigram links together with only one additional long-range link: that from L to R. We choose the models P_{LR} to be simple enough so that the parameters of all the $|V|^2$ possible models can be estimated in parallel. We then rank the models according to the likelihood each assigns to the training corpus, and choose those pairs (L, R) corresponding to the highest ranked models. This list of word pairs constitutes the "grammar" as described in the previous section.

The model P_{LR} that we construct for a particular pair (L, R) is a simplification of the model of the previous section; it can be described as a probabilistic finite state automaton (and thus it requires no stack). Before explaining the details of the model, consider the standard bigram model $\mathtt{b}(\,w' \,|\, w\,)$ viewed as a probabilistic finite state machine. This machine maintains a finite memory m that contains the previously generated word, and can be in one of two states, as shown in Figure 1. The machine begins in state 1 with $m = \perp$. The machine operates by making a state transition, outputting a word w, and then setting the memory m to w. More precisely, the machine remains in state 1 with probability $\mathtt{d}(\,\mathtt{step} \,|\, m\,) = 1 \perp \mathtt{b}(\,\perp \,|\, m\,)$. Given that a transition to state 1 is made, the word $w \neq \perp$ is output with probability $\mathtt{b}'(\,w \,|\, m\,)$, where

$$\mathtt{b}'(\,w \,|\, m\,) = \frac{\mathtt{b}(\,w \,|\, m\,)}{1 \perp \mathtt{b}(\,\perp \,|\, m\,)}\,.$$

Alternatively, the machine outputs $\perp$ and proceeds to state 2, where it halts, with probability $\mathtt{d}(\,\mathtt{halt} \,|\, m\,) = \mathtt{b}(\,\perp \,|\, m\,)$.

The machine underlying our probabilistic model P_{LR} is depicted in a similar fashion in Figure 2. Like the bigram machine, our new automaton maintains a memory m of the most recently output word and begins in state 1 with $m = \perp$. Unlike the bigram machine, it enters a special state whenever word L is generated: from state 1, the machine outputs L, sets $m = L$, and makes a transition to state 3 with probability $\mathtt{b}(\,L \,|\, m\,)$. A transition from state 3 back to state 1 is made with probability $\mathtt{d}_{\mathrm{LR}}(\,\mathtt{step}|\, L\,)$. In this case, no word is output, and the memory m remains set to L. Alternatively, from state 3 the machine can output a word $w \neq L, R$ and proceed to state 4 with probability

$\mathtt{d}_{\mathrm{LR}}(\mathtt{branch} | L)\, \mathtt{b}_{\mathrm{LR}}(w | m)$, where

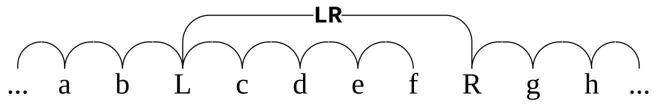
$$\mathtt{b}_{\mathrm{LR}}(w | m) = \frac{\mathtt{b}(w \,|\, m)}{1 \perp \mathtt{b}(L \,|\, m) \perp \mathtt{b}(R \,|\, m) \perp \mathtt{b}(\perp \,|\, m)}.$$

Once in state 4, the machine behaves much like the original bigram machine, except that neither an L nor a R can be generated. Word w is output and the machine remains in state 4 with probability $\mathtt{d}_{\mathrm{LR}}(\mathtt{step} | m)\, \mathtt{b}_{\mathrm{LR}}(w | m)$; it makes a transition back to state 1 and outputs word R with probability $\mathtt{d}_{\mathrm{LR}}(\mathtt{halt} | m)$.

According to this probabilistic finite state machine, words are generated by a bigram model except for the word R, which is generated either from its immediate predecessor or from the closest L to its left. Maximum-likelihood training of this machine yields an estimate of the reduction in entropy over the bigram model afforded by allowing long-range links between L and R in the general model presented in Section 2.

Training of the models P_{LR} for many (L, R) pairs in parallel is facilitated by two approximating assumptions on the parameters. First, we assume that $\mathtt{b}_{\mathrm{LR}}(w | m) = \mathtt{b}(w \,|\, m)$. Second, we assume that $\mathtt{d}_{\mathrm{LR}}(d | m) = \mathtt{d}_{\mathrm{LR}}(d)$ for $m \neq L$. Under this assumption, the parameter $\mathtt{d}_{\mathrm{LR}}(\mathtt{halt})$ encodes the distribution of the number of words between L and R; in the hidden model the number of words between them is geometrically distributed with mean $\mathtt{d}_{\mathrm{LR}}(\mathtt{halt})^{-1}$.

Each model P_{LR} can be viewed as a link grammar enhancement of the bigram model in the following way. In the bigram model each non-boundary word w has the single `step` disjunct, allowing only links between adjacent words. In the model P_{LR} we add additional disjuncts to allow long-range links between L and R. Specifically, we give all words the two disjuncts `step` and `halt`. In addition, we give L a third disjunct $\mathtt{branch}_{\mathrm{LR}}$, which like `step` allows connections to any word on the left and right, and in addition requires a long-range connection to R. Similarly, we give R a third disjunct $\mathtt{step}_{\mathrm{LR}}$ which connects to L on the left and any word on the right. This allows linkages such as



Now suppose that $S = w_1 \cdots w_N$ is a sentence containing a single (L, R) pair separated by at least one word. The probability of S is a sum over two linkages, $\mathcal{L}_{\mathrm{bigram}}$ and $\mathcal{L}_{\mathrm{LR}}$: $\mathtt{P}_{\mathrm{LR}}(S) = \mathtt{P}_{\mathrm{LR}}(\mathcal{L}_{\mathrm{bigram}}) + \mathtt{P}_{\mathrm{LR}}(\mathcal{L}_{\mathrm{LR}})$. If we let k be the the number of words between L and R, then under the two approximating assumptions made above it is easy to see that

$$\begin{aligned} \mathtt{P}_{\mathrm{LR}}(\mathcal{L}_{\mathrm{LR}}) &= c\, \mathtt{d}_{\mathrm{LR}}(\mathtt{branch}_{\mathrm{LR}} | L)\, \mathtt{d}_{\mathrm{LR}}(\mathtt{step})^{k-1}\, \mathtt{d}_{\mathrm{LR}}(\mathtt{halt}) \\ \mathtt{P}_{\mathrm{LR}}(\mathcal{L}_{\mathrm{bigram}}) &= c\, \mathtt{d}_{\mathrm{LR}}(\mathtt{step} | L)\, \mathtt{b}(R \,|\, R \perp 1) \end{aligned}$$

where c depends only on bigram probabilities. If, on the other hand, S contains a single L and no R then there is a unique linkage for the sentence whose probability is $\mathtt{d}_{\mathrm{LR}}(\mathtt{step} | L)$ times the bigram probability of the sentence. More generally, given the model just described, it is easy to write down the probability $\mathtt{P}_{\mathrm{LR}}(S)$ of any sentence with respect to the parameters $\mathtt{d}_{\mathrm{LR}}(d | w)$ and $\mathtt{b}(w_2 \,|\, w_1)$.

We train the parameters of this family of models in parallel, using the forward-backward algorithm. In order to do this, we first make a single pass through the training corpus, accumulating the following counts. For each (L, R) pair, we count $N(k, w|L, R)$, the number of times that L and R are separated by exactly $k \geq 1$ words, none of which is L or R, and the word immediately before R is w. We also count $N(\neg R|L)$, the number of times L appears in a sentence and either is not followed by an R in the same sentence or is followed by an L before an R. In terms of these counts, the increase in log-likelihood for model P_{LR} over the bigram model, in bits of information, is given by

$$\mathrm{Gain}_{\mathrm{LR}} = \sum_{k,w} N(k, w|L, R) \log \frac{\mathtt{P}_{\mathrm{LR}}\,(\,k, w, R|\,L\,)}{\mathtt{b}\,(\,R\,|\,w\,)} + N(\neg R|L) \log \mathtt{d}_{\mathrm{LR}}\,(\,\mathtt{step}|\,L\,)$$

where

$$\mathtt{P}_{\mathrm{LR}}\,(\,k, w, R|\,L\,) = \\ \mathtt{d}_{\mathrm{LR}}\,(\,\mathtt{branch}_{\mathrm{LR}}|\,L\,)\,\mathtt{d}_{\mathrm{LR}}\,(\mathtt{step})^{k-1}\,\mathtt{d}_{\mathrm{LR}}\,(\mathtt{halt}) + (1 - \mathtt{d}_{\mathrm{LR}}\,(\,\mathtt{branch}_{\mathrm{LR}}|\,L\,)\,\mathtt{b}\,(\,R\,|\,w\,)$$

and

$$N(\neg R|L) = c(L) - \sum_{k,w} N(k, w|L, R)$$

with $c(L)$ the unigram count of L. Using this formula, forward-backward training can be quickly carried out in parallel for all models, without further passes through the corpus. The results of this calculation are shown in Section 6.

4 The Mechanics of EM Estimation

In this section we describe the mechanics of estimating the parameters of our model. Our concern is not the mathematics of the inside-outside algorithm for maximum-likelihood estimation of link grammar models [LST92], but managing the large quantities of data that arise in training our model on a substantial corpus. We restrict our attention here to the short-range trigram probabilities $\mathtt{t}(z \mid x\ y)$ since these constitute the largest amount of data, but our methods apply as well to the long-range trigrams $\mathtt{l}\,(\,z\,|\,x\ y\,)$ and disjunct probabilities $\mathtt{d}\,(\,d\,|\,x\ y\,)$.

To begin, observe that the trigram probabilities must in fact be EM-trained. In a pure trigram model the quantity $\mathtt{t}(z \mid x\ y)$ is given by the ratio $c(x\ y\ z)/\sum_{a \in V} c(x\ y\ a)$ where $c(x\ y\ z)$ is the number of times the trigram $(x\ y\ z)$ appears in the training corpus $\mathcal{C}$. But in our link grammar model the trigram probabilities represent the conditional word probabilities in the case when the `step` linkage was used, and this is probabilistically determined. The same ratio determines $\mathtt{t}(z \mid x\ y)$, but the $c(x\ y\ z)$ are now *expected* counts, where the expectation is with respect to trainable parameters.

The EM algorithm begins with some initial set $\mathtt{t}(z \mid x\ y)$ of trigram probabilities. For each sentence S of the corpus, the algorithm labels the trigrams $(w_{i-2}\ w_{i-1}\ w_i)$ of S with these probabilities. From these and other parameters, the E-step determines partial estimated counts $\partial(w_{i-2}\ w_{i-1}\ w_i)$. The partial counts for a particular trigram $(x\ y\ z)$, accumulated over all instances of the trigram in the corpus, give the trigram's full estimated count $c(x\ y\ z)$.

Our difficulties occur in implementing the EM algorithm on the desired scale. To explain these difficulties, we will briefly sketch some naïve approaches. We assume the computer we will use has a substantial but not unlimited primary memory, which may be read and written at random, and a much larger secondary memory, which must be read and written sequentially. We will treat the corpus $\mathcal{C}$ as a series of words, $w_1,\ w_2,\ \ldots,\ w_{|\mathcal{C}|}$, recorded as indices into a fixed vocabulary V; this series is marked off into sentences. A word index is represented in 2 bytes, and a real value in 4 bytes.

Suppose we try to assign trigram probabilities $\mathtt{t}(w_i \mid w_{i-2}\ w_{i-1})$ to the sentence by looking them up in a table, and likewise accumulating the partial counts $\partial(w_{i-2}\ w_{i-1}\ w_i)$ into a table. Both must be randomly addressable and hence held in primary memory; each table must have space for $|V|^3$ entries. For realistic vocabularies of size $|V| \approx 5 \times 10^4$, the two tables together would occupy 10^{15} bytes, which far exceeds the capacity of current memory technologies, primary or secondary.

In a corpus of $|\mathcal{C}|$ words, no more than $|\mathcal{C}|$ distinct trigrams may appear. This suggests that we maintain the table by entering values only for the trigrams $(x\ y\ z)$ that actually appear in the corpus. A table entry will consist of $(x\ y\ z)$, and its $\mathtt{t}(z \mid x\ y)$ and $\partial(x\ y\ z)$. For fast access, the table is sorted by trigram. Unfortunately, this approach is also impractical. For a moderate training corpus of 25 million words, this table will occupy on the order of $25 \times 10^6 \times (6+4+4) = 350 \times 10^6$ bytes, which exceeds the primary memory of a typical computer.

Thus we are forced to abandon the idea of maintaining the needed data in primary memory. Our solution is to store the probabilities and counts in secondary memory; the difficulty is that secondary memory must be read and written sequentially.

We begin by dividing the corpus into R segments $\mathcal{C}^1,\ \ldots,\ \mathcal{C}^R$, each containing about $|\mathcal{C}|/R$ words. The number of segments R is chosen to be large enough so that a table of $|\mathcal{C}|/R$ real values can comfortably reside in primary memory. For each segment $\mathcal{C}^i$, which is a sequence of words $w_1,\ \ldots,\ w_{\mathcal{C}_i}$, we write an entry file E_i with structure

$$(w_1\ w_2\ w_3)\ \ 3,\ (w_2\ w_3\ w_4)\ \ 4,\ \ldots,\ (w_{|\mathcal{C}^i|-2}\ w_{|\mathcal{C}^i|-1}\ w_{|\mathcal{C}^i|})\ \ |\mathcal{C}^i|\,.$$

We sort E_i by trigram to yield SE_i, which has structure

$$(x\ y\ z_1)\ j_1^{(x\ y\ z_1)},\ \ldots,\ (x\ y\ z_1)\ j_{N(x\ y\ z_1)}^{(x\ y\ z_1)},\ (x\ y\ z_2)\ j_1^{(x\ y\ z_2)},\ \ldots,\ (x\ y\ z_2)\ j_{N(x\ y\ z_2)}^{(x\ y\ z_2)},\ \ldots.$$

Here we have written $N(x\ y\ z)$ for the number of times a trigram appears in the segment, and $j_1^{(x\ y\ z)},\ \ldots,\ j_{N(x\ y\ z)}^{(x\ y\ z)}$ for the sequence of positions where it appears. This sort is done one time only, before the start of EM training.

A single EM iteration proceeds as follows. First we perform an E-step on each segment $\mathcal{C}^i$. We assume the existence of a file AT_i that contains sequentially arranged trigram probabilities for $\mathcal{C}^i$. (For the very first EM iteration, it is easy to construct this file by writing out appropriate uniform probabilities.) Each segment's E-step sequentially writes a file PCT_i of partial estimated counts, $\partial(w_1\ w_2\ w_3),\ \partial(w_2\ w_3\ w_4),\ \ldots,\ \partial(w_{|\mathcal{C}^i|-2}\ w_{|\mathcal{C}^i|-1}\ w_{|\mathcal{C}^i|})$.

Next we sum these partial counts to obtain the segment counts. To do this we read PCT_i into primary memory. Then we read SE_i sequentially and accumulate the segment count $c_i(x\ y\ z_1) = \sum_{k=1}^{N(x\ y\ z_1)} \mathrm{PCT}_i[j_k^{(x\ y\ z_1)}]$, and so on for each successive trigram of $\mathcal{C}^i$. As each sum completes, we write it sequentially to a file SCT_i of segment counts, of format $c_i(x\ y\ z_1),\ c_i(x\ y\ z_2),\ \ldots$ The trigram that identifies each count can be obtained by sequential inspection of SE_i.

Now we merge across segments, by scanning all $2R$ files SE_i and SCT_i, and forming the complete counts $c(x\ y\ z) = \sum_{i=1}^{R} c_i(x\ y\ z)$. As we compute these sums, we maintain a list $c(x\ y\ z_1)$, $c(x\ y\ z_2)$, ... in primary memory—there will be no more than $|V|$ of them—until we encounter a trigram $(u\ v\ \cdot)$ in the input stream where $x \neq u$ or $y \neq v$. Then we compute $c(x\ y) = \sum_{z \in V} c(x\ y\ z)$, and dump the trigrams $(x\ y\ z)$ and quotients $\mathtt{t}'(z \mid x\ y) = c(x\ y\ z)/c(x\ y)$ sequentially to a file ST$'$. Note that ST$'$ is a sorted list of all the reestimated trigram probabilities.

To complete the process we must write a sequentially ordered file AT$'_i$ of the reestimated trigram probabilities for $\mathcal{C}^i$. First we create a table in primary memory of size $|\mathcal{C}^i|$. Then we read SE$_i$ and ST$'$ sequentially as follows. For each new trigram $(x\ y\ z)$ we encounter in SE$_i$, we search forward in ST$'$ to find $\mathtt{t}'(z \mid x\ y)$. Then for each $j_1^{(x\ y\ z)}, \ldots, j_{N(x\ y\ z)}^{(x\ y\ z)}$ listed for $(x\ y\ z)$ in SE$_i$, we deposit $\mathtt{t}'(z \mid x\ y)$ in AT$'_i[j_k^{(x\ y\ z)}]$. When SE$_i$ is exhausted we have filled each position in AT$'_i$. We write it sequentially to disk and are then ready for the next EM iteration.

5 Smoothing

The link grammar model given by (1) expresses the probability of a sentence in terms of three sets of more fundamental probability distributions $\mathtt{t}$, $\mathtt{l}$ and $\mathtt{d}$, so that $P(S) \equiv P(S; \mathtt{t}, \mathtt{l}, \mathtt{d})$. In the previous sections, we tacitly assumed 2-word history, non-parametric forms. That is, we allowed a separate *free* parameter for each 2-word history and prediction value subject only to the constraints that probabilities sum to one. In the case of the trigram distribution $\mathtt{t}$, for example, there are separate parameters $\mathtt{t}(w|w'w'')$ for each triple of words (w, w', w'') subject to the constraints $\sum_w \mathtt{t}(w|w'w'') = 1$ for all (w', w''). We will refer to such 2-word history distributions as 3-gram estimators, since they are indexed by triples, and will denote them by $\mathtt{t}_3$, $\mathtt{l}_3$ and $\mathtt{d}_3$. In the previous section we outlined an efficient implementation of EM (inside-outside) training, for adjusting the parameters of $\mathtt{t}_3$, $\mathtt{l}_3$ and $\mathtt{d}_3$ to maximize the log-likelihood of a large corpus of training text.

Unfortunately, we cannot expect the link grammar model using maximum likelihood distributions to work well when applied to new data. Rather, the distributions are likely to be too sharply determined by the training corpus to generalize well. This is the standard problem of overtraining and may be addressed by mixing the sharply defined distributions with less sharp ones to obtain smoother distributions. This procedure is referred to as *smoothing*.

The smoothing we employ in the link grammar is motivated by the smoothing typically used for the trigram language model [BBdSM91]. The idea is to linearly combine the 3-gram estimators $\mathtt{t}_3$, $\mathtt{l}_3$ and $\mathtt{d}_3$ with corresponding 2-gram, 1-gram and uniform estimators to obtain smooth distributions $\tilde{\mathtt{t}}_\lambda$, $\tilde{\mathtt{l}}_\lambda$ and $\tilde{\mathtt{d}}_\lambda$. In the case of $\tilde{\mathtt{t}}_\lambda$ we have

$$\tilde{\mathtt{t}}_\lambda(w|w'w'') = \lambda_3 \mathtt{t}_3(w|w'w'') + \lambda_2 \mathtt{t}_2(w|w') + \lambda_1 \mathtt{t}_1(w) + \lambda_0 \mathtt{t}_0\,. \tag{2}$$

Here $\mathtt{t}_3$, $\mathtt{t}_2$ and $\mathtt{t}_1$ denote 3-gram, 2-gram and 1-gram estimators for $\mathtt{t}$, and $\mathtt{t}_0$ denotes a uniform distribution. The 2-gram estimator $\mathtt{t}_2$ has a separate parameter $\mathtt{t}_2(w|w')$ for each 2-gram $(w\,w')$ subject to the constraint that $\sum_w \mathtt{t}_2(w|w') = 1$. The 1-gram estimator $\mathtt{t}_1$ has a separate parameter $\mathtt{t}_1(w)$ for each w. In general, an n-gram estimator depends on $n-1$ words of context. The parameters λ_i satisfy the constraint $\sum_i \lambda_i = 1$ to ensure that $\tilde{\mathtt{t}}_\lambda$ is a probability distribution. Equation (2) employs the same vector $(\lambda_0, \lambda_1, \lambda_2, \lambda_3)$, for each triple (w, w', w''). In practice, different vectors of λ's are used for different triples. We define $\tilde{\mathtt{l}}_\lambda$ and $\tilde{\mathtt{d}}_\lambda$ similarly. We then define the smooth link grammar model $\tilde{P}_\lambda$ using these smooth distributions: $\tilde{P}_\lambda(S) \equiv P(S; \tilde{\mathtt{t}}_\lambda, \tilde{\mathtt{l}}_\lambda, \tilde{\mathtt{d}}_\lambda)$.

To completely specify the smooth distributions, we must fix the values of the parameters of the individual n-gram distributions as well as the mixing parameters λ. Estimating all of these simultaneously using maximum likelihood training would defeat the purpose of smoothing: we would find that the only non-zero λ's would be those multiplying the 3-gram estimators, which would ultimately train to their maximum likelihood (and thus unsmooth) values! Instead we adopt the following procedure motivated by the *deleted interpolation* method sometimes used for the trigram model [BBdSM91]. We first divide our corpus of sentences into two parts: a large *training* corpus $\mathcal{T}$, and a smaller *smoothing* corpus $\mathcal{S}$. We estimate the n-gram estimators using the training corpus *only* according to the following scheme. The 3-gram estimators $\mathtt{t}_3$, $\mathtt{l}_3$ and $\mathtt{d}_3$ are chosen to maximize the log-likelihood $\sum_{S\in\mathcal{T}} \log P(S; \mathtt{t}_3, \mathtt{l}_3, \mathtt{d}_3)$ of the training corpus using the EM technique described in the previous section.

The 3-gram estimators are then used to "reveal" the hidden linkages of the training corpus, and the 2-gram and 1-gram estimators are chosen to maximize the likelihood of the training corpus *together with* these revealed linkages. Thus, for $i = 1, 2$, the distributions $\mathtt{t}_i$, $\mathtt{l}_i$ and $\mathtt{d}_i$ maximize $\sum_{S\in\mathcal{T}} \sum_\Lambda P(\Lambda | S; \mathtt{t}_3, \mathtt{l}_3, \mathtt{d}_3) \log P(S, \Lambda | \mathtt{t}_i, \mathtt{l}_i, \mathtt{d}_i)$. This procedure, while somewhat unwieldy to explain, is simple to implement, as it amounts to obtaining the 2-gram and 1-gram estimators as appropriate conditionals of the EM counts for the 3-gram estimators.

With the n-gram estimators thus determined, we adjust the mixing parameters λ to maximize the probability of the smoothing corpus *only*. The logarithm of this probability is

$$\mathcal{O}_{\text{outer}}(\lambda) = \sum_{S\in\mathcal{S}} \log P(S | \tilde{\mathtt{t}}_\lambda, \tilde{\mathtt{l}}_\lambda, \tilde{\mathtt{d}}_\lambda) = \sum_{S\in\mathcal{S}} \log \sum_\Lambda P(S, \Lambda | \tilde{\mathtt{t}}_\lambda, \tilde{\mathtt{l}}_\lambda, \tilde{\mathtt{d}}_\lambda)\,.$$

This maximization is complicated by the fact that the probability of a sentence now involves not only a sum over hidden linkages, but for each linkage, a sum over hidden λ indices as well. We deal with this by employing *nested* EM iterations, as follows.

1. Begin with some initial λ's.

2. By the inside-outside algorithm described in [LST92] and the previous section, reveal the hidden linkages of the *smoothing* corpus using the smooth distributions $\tilde{\mathtt{t}}_\lambda$, $\tilde{\mathtt{l}}_\lambda$ and $\tilde{\mathtt{d}}_\lambda$ and accumulate the EM counts $c_{\mathtt{t},\lambda}(t)$, $c_{\mathtt{t},\lambda}(l)$ and $c_{\mathtt{d},\lambda}(d)$ for the parameters t, l, d of the distributions $\mathtt{t}$, $\mathtt{l}$ and $\mathtt{d}$. These are the counts obtained by maximizing the auxiliary function

$$\sum_{S\in\mathcal{S}} \sum_\Lambda P(\Lambda | S; \tilde{\mathtt{t}}_\lambda, \tilde{\mathtt{l}}_\lambda, \tilde{\mathtt{d}}_\lambda) \log P(S, \Lambda | \tilde{\mathtt{t}}_{\lambda'}, \tilde{\mathtt{l}}_{\lambda'}, \tilde{\mathtt{d}}_{\lambda'})$$

   with respect to λ'. Their accumulation is the E-step of the outermost EM iterations.

3. Form the objective function

$$\mathcal{O}_{\text{inner}}(\lambda') = \sum_t c_{\mathtt{t},\lambda}(t) \log \tilde{\mathtt{t}}_{\lambda'}(t) + \sum_l c_{\mathtt{l},\lambda}(l) \log \tilde{\mathtt{l}}_{\lambda'}(l) + \sum_d c_{\mathtt{d},\lambda}(d) \log \tilde{\mathtt{d}}_{\lambda'}(d)\,.$$

Notice that the λ' indices are hidden in $\mathcal{O}_{\text{inner}}(\lambda')$. Use the forward-backward algorithm to find the λ's that maximize $\mathcal{O}_{\text{inner}}(\lambda')$ subject to the appropriate constraints. These nested EM iterations are the M-step of the outermost EM iterations.

4. Using these λ's as new guesses for the λs, return to step 1, and iterate until converged.

Note that the outermost EM steps use the inside-outside algorithm for link grammars; the hidden parses are in general context-free in generative power. However, step 3, which is the M-step for the inside-outside algorithm, is itself an EM estimation problem. Here, however, the hidden structure is regular, so the estimation can be carried out using the forward-backward algorithm for probabilistic finite state machines. The general EM algorithm technology guarantees that each iteration of the above algorithm increases the log-likelihood $\mathcal{O}_{\text{outer}}$ of the smoothing corpus with respect to the smooth model so far. In practice, we have observed that roughly three iterations of the outer EM iterations and 15 iterations of the inner EM iterations suffice to smooth the parameters of our models.

6 Sample Results

This section presents the results of inferring and training our long-range trigram model on a corpus of Wall Street Journal data.

Figure 3 lists examples of the word pairs that were discovered using the inference scheme discussed in Section 3. Recall that these pairs are discovered by training a link grammar that allows long-range links between a single, fixed, pair of words. A given pair is judged by the reduction in entropy that its one-link model achieves over the bigram model. In the table, this improvement, measured in bits of information, is shown in the third column. The first section of the table lists the pairs that resulted in the greatest reduction in entropy. The fourth column of the table gives the values of the probability $\mathtt{d}(\mathtt{branch}_{\text{LR}} \mid L)$ after forward-backward training. This number indicates the frequency with which L generates R from long range, according to the trained model. The second section of the table lists examples of pairs with high $\mathtt{d}(\mathtt{branch}_{\text{LR}} \mid L)$. The fifth column of the table gives the values of the probability $\mathtt{d}_{\text{LR}}(\mathtt{halt})^{-1}$ after forward-backward training. Recall that since the number of words between L and R is geometrically distributed with mean $\mathtt{d}_{\text{LR}}(\mathtt{halt})^{-1}$ in the hidden model, a large value in this column indicates that L and R are on average widely separated in the training data. The third section of the table gives examples of such pairs. Finally, the fourth section of the table shows the results of the word-pair calculation applied to the corpus after it was tagged with parts-of-speech. The search was restricted to verb-preposition pairs, and some of the pairs which yielded the greatest reduction in entropy are shown here.

In Figures 4 and 5 we show plots of perplexity as a function of iteration in the EM training of the long-range trigram model described in Section 2, using the word-pair "grammar" that was automatically extracted. These graphs plot the perplexity as a function of iteration, with the trigram perplexity shown as a horizontal line. In the first plot, carried out over a training set of 2,521,112 words, the perplexity falls approximately 12.7% below the trigram perplexity after 9 iterations. After smoothing as described in Section 5, the perplexity on test data was approximately 4.3% below the smoothed trigram

L	R	$\text{Gain}_{\text{LR}} \times 10^5$	$\texttt{d}(\texttt{branch}_{\text{LR}} \mid L)$	$\texttt{d}_{\text{LR}}(\texttt{halt})^{-1}$
(	)	472.944	0.808	2.277
”	”	80.501	0.089	3.041
between	and	57.097	0.674	2.002
[	]	54.287	0.907	2.644
neither	nor	22.883	0.588	2.030
either	or	16.892	0.496	3.083
both	and	14.915	0.277	1.786
–	–	14.909	0.074	5.309
’	’	14.039	0.117	3.845
from	to	13.021	0.044	1.931
tit	tat	0.344	0.835	2.049
to_preheat	oven	1.663	0.773	1.084
to_whet	appetite	0.521	0.709	1.943
nook	cranny	0.618	0.619	2.426
to_flex	muscle	0.702	0.548	1.784
sigh	relief	0.624	0.411	2.123
loaf	bread	0.434	0.308	2.795
quarterback	touchdown	0.167	0.027	5.715
inning	hit	0.097	0.018	5.673
farmer	crop	0.347	0.023	5.609
investor	stock	0.270	0.014	5.149
firefighter	blaze	0.513	0.071	4.955
whether	or	5.123	0.124	4.925
she	her	9.672	0.078	4.007
to_describe	as	9.022	0.457	3.275
to_rise	to	7.654	0.261	2.437
to_prevent	from	7.491	0.407	3.743
to_turn	into	6.642	0.174	3.566
to_attribute	to	5.679	0.904	4.189
to_view	as	5.193	0.524	3.425
to_bring	to	4.960	0.237	3.836
to_range	to	4.864	0.660	5.356

Figure 3: Sample word pairs

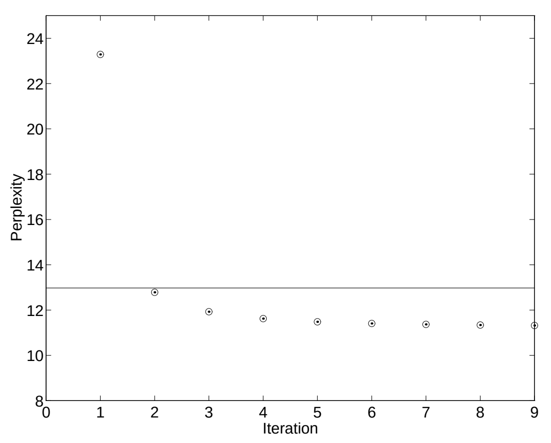


Figure 4: 2.5m corpus

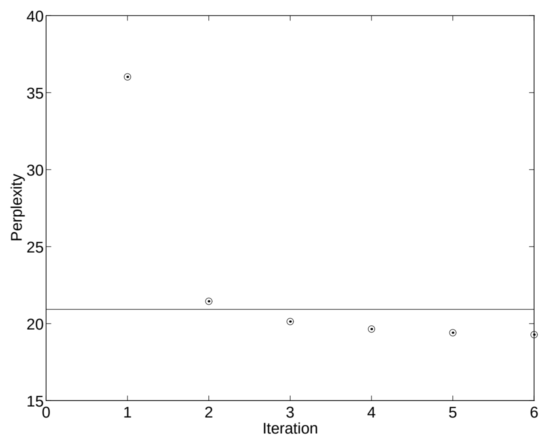


Figure 5: 25m corpus

perplexity. In the second plot, carried out over a training set of 25,585,580 words, the perplexity falls approximately 8% below the trigram perplexity after 6 iterations. After smoothing this model, the perplexity on test data was approximately 5.3% under the smoothed trigram perplexity. The fact that the magnitude of the entropy reduction on training data is not preserved after smoothing and evaluating on test data is an indication that the smoothing may be sensitive to the "bucketing" of the λ's.

These perplexity results are consistent with the observation that for a fixed word-pair grammar, as the training corpus grows in size the long-range trigram model becomes a small perturbation of the standard trigram model. This is because the number of disjunct parameters $\mathtt{d}(\,d\,|\,w_1\,w_2\,)$ and long-range trigram parameters $\mathtt{l}(\,w\,|\,w_1\,w_2\,)$ is on the order of the number of bigrams, which becomes negligible compared to the number of trigram parameters as the training set grows in size.

The smoothed models were incorporated into the *Candide* system for machine translation [BBP+94]. When compared with translations obtained with the system using the standard trigram model, our long-range model showed a slight advantage overall. For example, the French sentence "Manille a manqué d'électricité pendant dix heures mercredi," which was translated as "Manila has run out of electricity for ten hours Wednesday" using the standard language model, was translated as "Manila lacked electricity for ten hours Wednesday" using the link grammar model.

While the long-range trigram model that we have described in this paper represents only a small change in the trigram model itself, we believe that the techniques we develop here demonstrate the viability of more complex link grammar models, and show that significant improvements can be obtained using this approach.

References

[BBdSM91] L.R. Bahl, P.F. Brown, P.V. de Souza, and R.L. Mercer. Tree-based smoothing algorithm for a trigram language speech recognition model. *IBM Technical Disclosure Bulletin*, 34(7B):380–383, December 1991.

[BBP+94] A. Berger, P. Brown, S. Della Pietra, V. Della Pietra, J. Gillett, J. Lafferty, R. Mercer, H. Printz, and L. Ureš. The Candide system for machine translation. In *Human Language Technologies*, Morgan Kaufman Publishers, 1994.

[BT73] T. Booth and R. Thompson. Applying probability measures to abstract languages. *IEEE Transactions on Computers*, C-22:442–450, 1973.

[LST92] J. Lafferty, D. Sleator, and D. Temperley. Grammatical trigrams: A probabilistic model of link grammar. In *Proceedings of the AAAI Fall Symposium on Probabilistic Approaches to Natural Language*, Cambridge, MA, 1992.

[ST91] D. Sleator and D. Temperley. Parsing English with a link grammar. Technical Report CMU-CS-91-196, School of Computer Science, Carnegie Mellon University, 1991.

Appendix A: Enumerating Linkages

In this appendix we derive a formula for the number of linkages of the model described in Section 1 when the grammar allows long-range connections between any pair of words.

There is a natural correspondence between the linkages of model (1) and trees where each node has either zero, one, or two children. A node having one child will be called *unary* and a node having two children will be called *binary*. Let $a_{m,n}$ be the number of trees having m unary nodes and n binary nodes. Then $a_{m,n}$ satisfies the recurrence

$$a_{m,n} = a_{m-1,n} + \sum_{0 \le k \le m} \sum_{0 \le l \le n-1} a_{k,l}\, a_{m-k,n-l-1}\,.$$

Thus, the generating function $T(x,y) = \sum_{m,n \ge 0} a_{m,n}\, x^m\, y^n$ satisfies the equation

$$T(x,y) = 1 + x\,T(x,y) + y\,T^2(x,y)\,.$$

Since $T(0,0) = 1$ we have that

$$T(x,y) = \frac{1 - x - \sqrt{(1-x)^2 - 4y}}{2y}\,.$$

The total number of nodes in a tree that has m unary nodes and n binary nodes is $2n + m + 1$. Therefore, if $S(z) = \sum_{k \ge 0} s_k z^k$ is the generating function given by $S(z) = z\,T(z, z^2)$, then

$$s_k = \sum_{2n+m+1=k} a_{m,n}$$

and s_k is the number of trees having a total of k nodes. S is given by

$$\begin{aligned}
S(z) &= \frac{1 - z - \sqrt{(1-z)^2 - 4z^2}}{2z} \\
&= \frac{1 - z - \sqrt{1-3z}\sqrt{1+z}}{2z} \\
&= \frac{1}{2z} - \frac{1}{2} - \frac{1}{2z} \sum_{i \ge 0} \binom{1/2}{i} (-3)^i\, z^i \sum_{j \ge 0} \binom{1/2}{j} z^j \\
&= \frac{1}{6} \sum_{k \ge 0} \sum_{0 \le i \le k+1} \binom{1/2}{i} \binom{1/2}{k+1-i} (-3)^{i+1}\, z^k\,.
\end{aligned}$$

While we are unable to find a closed form expression for the coefficients

$$s_k = \frac{1}{6} \sum_{0 \le i \le k+1} \binom{1/2}{i} \binom{1/2}{k+1-i} (-3)^{i+1}$$

a few of the values are displayed below.

k	1	2	3	4	5	6	7	8	9	10	20	25
s_k	1	1	2	4	9	21	51	127	323	835	18,199,284	3,192,727,797

Appendix B: Deficiency

We say that a language model is *deficient* if it assigns a probability that is smaller than one to the set of strings it is designed to model. There are several ways in which a probabilistic link grammar can be deficient. One such way is if the total probability of finite linkages is smaller than one. In this appendix we derive conditions under which this type of deficiency can occur for a simplified version of our model. The general analysis is similar, but more intricate [BT73].

Following the notation of Appendix A, suppose that we generate trees probabilistically with a node having zero children with probability p_0, one child with probability p_1, and two children with probability p_2, irrespective of the label of the node. These probabilities correspond to the disjunct probabilities $\mathtt{d}(\mathtt{halt} \mid s)$, $\mathtt{d}(\mathtt{step} \mid s)$, and $\mathtt{d}(\mathtt{branch} \mid s)$. We ignore the short and long-range trigram probabilities in this simplified model. The probability of generating a tree with m unary nodes and n binary nodes is then $p_0^{n+1}\, p_1^m\, p_2^n$. The total probability assigned to finite trees is

$$T_{\text{finite}} = \sum_{m,n \geq 0} a_{m,n}\, p_0^{n+1}\, p_1^m\, p_2^n = p_0\, T(p_1, p_0\, p_2)\,.$$

Using the calculations of Appendix A, this leads directly to the relation

$$\begin{aligned} T_{\text{finite}} &= \frac{1 - p_1 - \sqrt{(1-p_1)^2 - 4 p_0\, p_2}}{2 p_2} \\ &= \frac{p_0 + p_2 - |p_0 - p_2|}{2 p_2}\,. \end{aligned}$$

In terms of the expected number of children $E[n] = p_1 + 2p_2$, we can state this as

$$T_{\text{finite}} = \begin{cases} 1 & E[n] \leq 1 \\ p_0/p_2 & E[n] \geq 1 \end{cases} \,.$$

More generally, for n-ary trees with probability p_i of generating i children, with $0 \leq i \leq n$, T_{finite} is the smallest root of the equation

$$T = \sum_{0 \leq i \leq n} p_i\, T^i$$

and $T_{\text{finite}} = 1$ in case $E[n] < 1$. This is a well-known result in the theory of branching processes.