Structured Language Modeling for Automatic Speech Recognition

Johanna Björklund, Loek Cleophas, and Marcus Karlsson

Department of Computing Science,
Umeå University, 90187 Umeå, Sweden
email: {johanna,cleophas,mk}@cs.umu.se

Abstract. Probabilistic lexicalized tree-insertion grammars (PLTIGs) are evaluated on a classification task relevant for automatic speech recognition. The baseline is a family of n-gram models tuned with Witten-Bell smoothing. The language models are trained on unannotated corpora, consisting of 10,000 to 50,000 sentences collected from the English section of Wikipedia. For the evaluation, an additional 150 random sentences were selected from the same source, and for each of these, approximately 3,200 variations were generated. Each variant sentence was obtained by replacing an arbitrary word by a similar word, chosen to be at most 2 character edits from the original. The evaluation task consisted in identifying the original sentence among the automatically constructed (and typically inferior) alternatives. In the experiments, the n-gram models outperformed the PLTIG model on the smaller data set, but as the size of data grew, the PLTIG model gave comparable results. While PLTIGs are more demanding to train, they have the advantage that they assign a parse structure to their input sentences. This is valuable for continued algorithmic processing, e.g., for summarization or sentiment analysis.

1 Introduction

Language models are a central concept in natural language processing. At an abstract level, they are parametrized families of functions that assign probabilities to natural language sentences. A 'good' language model should assign a higher probability to natural-sounding sentences and sentence fragments, compared to unlikely or even ungrammatical alternatives. Language models are commonly used in machine translation (MT) to rank candidate translations, in automatic speech recognition (ASR) to narrow the search-space by discarding unlikely transcriptions, and in summarization to shorten text while preserving readability.

The reigning approach to practical language modeling is to train structure-agnostic n-grams on large sets of data. In the n-gram model, the likelihood of a sequence of words $s = w_1, w_2, \ldots, w_n$ is the product of the likelihood of every subsequence of n consecutive words in s, with respect to some given set of training data. When greater precision is needed, the size of n can be increased, or some weighting or smoothing technique can be introduced to better leverage the training data. These language models have the advantages of being surprisingly

powerful considering their simplicity, easy to train from data, and can be applied in linear or even log-linear time (Klakow, 1998).

On the downside, the size of n-gram models grows exponentially in n. This means that for large values of n, only a fraction of all possible n-grams will appear in the training data, and to keep the model at a reasonable size, only the most frequent n-grams can be taken into account at any rate. For these reasons, Goodman (2001) concluded that proceeding beyond 5-grams is unlikely to be practically motivated. This conjecture has still not been contradicted a decade later. The flagship of the field, Google's n-gram viewer, contains 500 billion words compiled from 20 million books. In its construction, the value of n had to be restricted to 5 to limit the model's size, and all n-grams encountered fewer than 40 times were discarded without updating the model (Michel et al., 2010).

The fact that n has to be kept small causes problems. Take the following pair of sentences:

> She *rowed* between Yellowknife and Benchoko in a weathered *canoe*.
>
> She *rowed* between Yellowknife and Benchoko in a weathered *car*.

The first sentence is arguably more likely due to the semantic relation between *rowed* and *canoe*. However, no n-gram model could make this connection for a value of n less than 9, since no n-gram of shorter length contains both *rowed* and *canoe*, or *rowed* and *car*. Such simpler models would instead prefer the second sentence, *car* being a more common form of transport than *canoe*.

Another thing that is missing is a syntactical analysis of the input sentence, that is, n-grams do not support parsing. Parsing is useful in itself, as it tells us whether a sentence is likely to be perceived as grammatical. Furthermore, parse trees are suitable for continued algorithmic processing, for example, to obtain a semantic analysis. If we want to understand who does what to whom in a sentence, it helps to know how the sentence is put together, in particular, what the central verbs and their arguments are.

Context-free grammars (CFG) can be used for syntactical analysis, but are typically worse at predicting word sequences than n-grams, unless they are lexicalized. In a lexicalized CFG, each production rule produces at least one lexical item (Schabes & Waters, 1993a). CFGs can be lexicalized using Greibach normal form, but this skews the syntactical structure. Another alternative is to use Tree-adjoining Grammars (TAG). TAGs represent a language of parse trees as a set of parse-trees fragments, together with rewrite rules that regulate how the fragments may be pieced together into larger structures. TAGs were developed by linguists and have several appealing properties; they are expressive, straightforward to lexicalize, and offer parsing. However, the parsing complexity is as high as $O(n^6)$, which cannot be considered practical.

Aiming for the middle ground, Hwa (2001) suggested the use of probabilistic lexicalized tree insertion grammars (PLTIGs), a family of TAGs with simplified rewrite rules. PLTIGs are as good as trigrams at predicting word sequences, but also offer syntax-aware language modeling. Their parsing com-

plexity is $O(n^3)$, which is equal to that of probabilistic context-free grammars (PCFGs) and a factor n^3 faster than TAGs. PLTIGs also have the advantage that their expectation-maximization training converges faster than for similarly sized PCFGs (see Chapter 3 of (Hwa, 2001)).

In this article we evaluate PLTIGs as an alternative for n-grams in Automatic speech recognition, more precisely, the translation of spoken words into text. Modern ASR systems are typically built around an acoustic model, a lexical model, and a language model. The acoustic model maps utterances into phonemes, the lexical model concatenates phonemes to match a dictionary of known words, and the language model combines words into sentences. Today, n-grams and Hidden Markov Models (Leonard E. Baum, 1966) are commonly used language models in ASR systems, and we are interested to learn how PLTIGs stand up to these. To isolate the influence of the language models from the ASR system at large, we focus on a classification task in which the models pick out a sentence s from a set of alternatives, generated from s by replacing a word by a similar but inappropriate word.

1.1 Background and related work

Automatic speech recognition originates from the post-war era. The scope was initially limited to very small vocabularies, for example, single numeric digits (Marill, 1961). An early attempt to build a hardware device capable of recognizing speech was made by Smith (1951). His machine passed the input acoustic signal through a sequence of contiguous band-pass filters, generating 32 signals that were fed to a switch selector panel, which in turn mapped their energy distributions to phonemes. Forgie and Forgie (1959) presented a system similar to Smith's, with the exception that the recognition was done by a computer program running on a general-purpose machine. This meant that, unlike its predecessors, it could easily be adapted to new speakers. The success rate was now up to 93 %, but the program was only capable of differentiating between ten English vowels. Progress continued to be slow throughout the 1960s, and the entire field was heavily criticized for not making any substantial results. Real-world applications were considered distant (Pierce, 1969).

Language models came into focus during the 1970s. Jelinek (1976) introduced the *New Raleigh* language model, which was essentially a Hidden Markov Model. A similar system had also been presented by Baker (1975) the year before. The advantage of these models was that they allowed the ASR system to work comparably well under high error rates. From then on, probabilistic models and specifically n-grams were a commodity in the field.

Interest in other types of statistical language models increased during the late 1980s, with an emphasis on tree-based models. Tree-adjoining grammars had already been extensively studied in the context of formal languages (Joshi, Levy, & Takahashi, 1975), but in 1990 Shieber et al. (Abeillé, Schabes, & Joshi, 1990; Shieber & Schabes, 1990) successfully used a variant of them for modeling natural languages. Their intention was to use TAGs for a wider range of applications, including semantic interpretation and machine translation. For the latter

application, the idea was to use so called *synchronous TAGs* where elementary trees are mapped to a semantic representation unrelated to syntax. Two practical problems with this approach were the computational cost of parsing, and the lack of suitably annotated data sets.

Early work on lexicalized tree insertion grammars (lexicalized TIGs, or LTIGs) was conducted in 1994 by Schabes and Waters (Schabes & Waters, 1994). In their article they mention that the probabilities with which adjunctions and substitutions occur can be controlled by adding a stochastic parameter, as previously done for context-free grammars (Schabes & Waters, 1993b). Hwa (Hwa, 1998) later provided a semi-supervised expectation-maximisation algorithm for probabilistic lexicalized TIGs, and discussed how to infer high-quality grammars with minimal data annotation. As mentioned previously, parsing with respect to TIGs is substantially easier than with TAGs, but still expensive compared to n-grams.

In the 2000s, the advent of grid, massively parallel, and later cloud computing moved the boundaries for what could be considered computationally feasible. As the number of high-quality syntactically and semantically annotated corpora continues to grow, new possibilities for machine learning open up. This article aims to reassess the practical value of TIGs in light of these developments.

1.2 Outline

This article is organized as follows. Section 2 revises the relevant language models. Section 3 describes the experimental setup, and Section 4 reports and discusses the results. Section 5 concludes the article by outlining directions for future work.

2 Theory

We begin by recalling the definitions of n-grams, TAGs (as an intermediary step), and finally PLTIGs. Since the theoretical results that support this study are already in place, including the parsing complexity and the correctness of the EM algorithm, we describe the models at a fairly high level and refer to (Hwa, 1998) for the formal definitions. Those familiar with n-grams and PLTIG's may want to proceed directly to Section 3.

2.1 *n*-grams

The likelihood $P(w_{1,n})$ of a sequence of words $w_{1,n} = w_1, \ldots, w_n$ can be computed using the chain rule of probability

$$P(w_{1,n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1,2}) \cdots P(w_n|w_{1,n-1}) \tag{1}$$

$$= \prod_{i=1}^{n} P(w_i|w_{1,i-1}) \ . \tag{2}$$

The idea behind n-grams is to approximate the likelihood of w_i in $w_{1,n}$ by

$$P(w_i|w_{1,i-1}) \approx P(w_i|w_{i-n+1,i-1}) \ . \tag{3}$$

Substituting Equation 3 into Equation 2, the likelihood of the entire sequence is approximated by

$$P(w_{1,n}) \approx \prod_{i=1}^{n} P(w_i|w_{i-n+1,i-1}) \ .$$

When the value of n is 1, 2, 3, or 4, the model is often referred to as unigram, bigram, trigram, or quadgram, respectively. Similarly, for $n =$ the model may be referred to as quintgram.

n-grams are straight-forward to learn from training data through a Maximum Likelihood Estimation (MLE) process. The probability of a word w_i is

$$P_{MLE}(w_i|w^{i-n+1,i-1}) = \frac{c(w_{i-n+1,i})}{c(w_{i-n+1,i-1})} \ ,$$

where $c(u_{1,k})$ counts how often the sequence of words $u_{1,k}$ appears in the data. Simply put, a $P_{MLE}(w_i)$ computed with unigrams only considers the relative frequency of w_i in the data, whereas bigrams take a history of one word into account, and trigrams a history of two words.

For most applications, the training data cannot be expected to contain all n-grams that will later be encountered in the application data. As can be expected, this problem of data sparsity diminishes as the size of the training data increases (Banko & Brill, 2001), but it seldom disappears completely (Allison, Guthrie, & Guthrie, 2006). For this reason, the MLE learning is usually followed by a round of *smoothing*; a redistribution of the probability mass assigning a non-zero probability also to here-to unseen n-grams. There are several popular smoothing methods to choose between, for example, Laplace, Good-Turing, and Kneser-Ney. In this article, we use Witten-Bell smoothing. At the core of Witten-Bell is the observation that when we see some words, we can easily guess what the next word will be, whereas for others, it is more difficult. The word *want* for instance is followed by *to* more often than not, but the word *to*, in turn, tells us less about the future. With Witten-Bell, the nth order smoothed model is a linear interpolation between the nth order unsmoothed model and the $(n-1)$th order smoothed model, and the shape of the interpolation is affected by the *predictiveness* of the words (Bell, Cleary, & Witten, 1990).

2.2 Tree Adjoining Grammars

As previously mentioned, tree-insertion grammars are a restricted form of tree adjoining grammars (Joshi et al., 1975). In contrast to n-grams, probabilistic TAGs assign likelihoods to parse trees rather than their surface forms, that is, the generated sentences. A TAG has rules in form of *elementary trees* of which there are two types; *elementary initial trees* and *elementary auxiliary trees*. Each elementary initial tree has a number of nonterminal and terminal leaf nodes. Auxiliary trees also have a special nonterminal leaf node of the same type as the

root node called the *foot node*, marked with the special symbol ∗. The sequence of nodes on the path from the root node to the foot node is called the *spine*. Figure 1 shows a few examples of elementary trees.

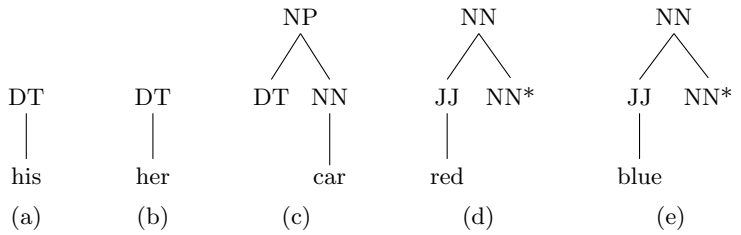


Fig. 1: Examples of elementary trees in a TAG. Elementary trees (a-c) show initial trees, while (d-e) are auxiliary trees. The auxiliary trees each have a foot node marked with a ∗ which is of the same type as the root node.

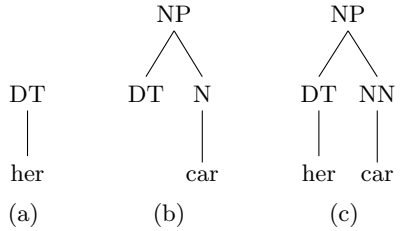


Fig. 2: The initial tree in (a) is substituted into a nonterminal node of the initial tree in (b), with the result of the new tree in (c). This is possible since the root node of (a) is of the same type as the nonterminal leaf node in (b).

If the root node of an initial tree matches a nonterminal leaf node of some other tree, then the first tree can be *substituted* into the nonterminal leaf node of the other tree, corresponding to a context-free derivation step. Figure 2 shows an example of this. The determiner *her* in the initial tree in Figure 2a is substituted into the leaf node of the initial tree for the noun *car* in Figure 2b. The result is the tree in Figure 2c, generating the construct *her car*.

An auxiliary tree can be inserted into an intermediate node of another tree through what is called *adjunction*. Since the root node and the foot node of an auxiliary tree have to have the same symbol, auxiliary trees can be inserted into another tree where such a symbol exists. An example of this is shown in Figure 3. Note that the auxiliary tree in Figure 3a has the same root and foot node as

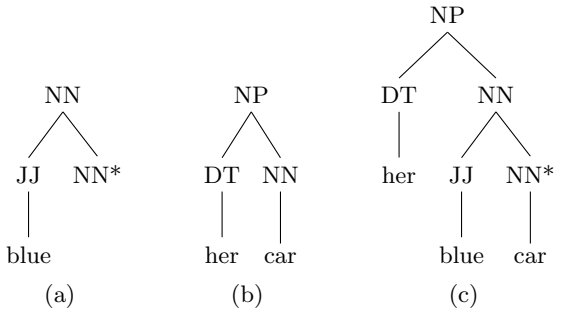


Fig. 3: The auxiliary tree in (a) is adjoined into the tree in (b), with the result of the new tree in (c). The tree from (a) can be inserted into (b) at the node `NN` since (a) has both a root and a foot node of that type.

one of the intermediary nodes in the tree in Figure 3b. This allows the adjective *blue* to be inserted into the sentence *her car*, producing the new sentence *her blue car*. TAGs are expressive due to the number of configurations they allow, but have the drawback of high parsing complexity in the order of $O(n^6)$.

More formally, a TAG is a tuple (Σ, V, I, A), where Σ is a set of terminal symbols, V is a set of nonterminal symbols, I is a set of initial trees, and A is a set of auxiliary trees. If every elementary tree structure in a TAG has at least one non-empty terminal leaf node with a lexical item, then that is called a *lexicalized TAG* (LTAG).

There is also a probabilistic version of TAGs (and hence of LTAGs), in which three probability distributions are included in the definition. These distributions are defined on the set of initial trees, and on the set Ω of possible substitution and adjunction events. A Probabilistic TAG is thus a tuple $(\Sigma, V, I, A, P_I, P_S, P_A)$ where (Σ, V, I, A) is a TAG, P_I is a mapping $I \to [0, 1]$ or the probability that an initial tree is used as the start of a derivation, P_S is a mapping $\Omega \to [0, 1]$ for the probability of a substitution, and P_A is a mapping $\Omega \to [0, 1]$ for the probability of an adjunction (Resnik, 1992).

2.3 Tree Insertion Grammars

Tree insertion grammars (TIGs) are a restricted form of TAG in which the rewrite steps are more regulated. A TIG is a quintuple (Σ, V, I, A, S) where (Σ, V, I, A) is a TAG and S is a distinguished nonterminal symbol. The semantics of TIGs are defined as for TAGs, with the following additions (Schabes & Waters, 1994).

– Every auxiliary tree has to be either a *left auxiliary tree* or a *right auxiliary tree*, meaning that they only have lexical items on the indicated side of the foot node.

– Left auxiliary trees are not allowed to be adjoined on a node that is *on* the path from the root to the foot of a right auxiliary tree, and vice versa.
– No more than one left auxiliary tree and one right auxiliary tree is allowed to be simultaneously adjoined into the same node.
– Adjunction on nodes that are *to the right of* the path from the root node to the foot node of a *left* auxiliary tree is not allowed, and vice versa.
– Adjunction on root nodes and foot nodes of auxiliary trees is not allowed.
– Adjunction on nodes that can be used for substitution is not allowed.

When combined, these restrictions only allow us to express context-free languages, but they also bring the parsing complexity down to $O(n^3)$.

A TIG is lexicalized (LTIG) if each elementary tree carries a lexical item. A TIG can also be probabilistic (PTIG or PLTIG) if it is parameterized by probabilities that describe how likely it is that operations are applied. A PTIG is defined as a TIG, with the addition of as many as eight probability distributions, $P_I, P_S, P_L, P_R, P_{NL}, P_{NR}, P_{RL}, P_{LR}$, defined as follows:

1. The probability $P_I(t)$ that the derivation starts with the initial tree t, so that $\Sigma_{t \in I} P_I(t) = 1$.
2. The probability $P_S(n, t)$ that an initial tree t is substituted into a leaf node n, so that $\Sigma_{t \in I} P_S(n, t) = 1$.
3. The probabilities $P_L(n, t)$ that a left auxiliary tree t is inserted into the internal nonterminal node n, and the probability $P_{NL}(n)$ that n takes no left adjunction, so that $P_{NL}(n) + \Sigma_{t \in A_L} P_L(n, t) = 1$ where A_L is the set of left auxiliary trees. The probability distributions P_R and P_{NR} are defined analogously to cover the same situation for the right-hand side.
4. The probabilities $P_{LR}(n)$ and $P_{RL}(n)$ that a simultaneous adjunction into the internal nonterminal node n makes a left-adjunction first or a right-adjunction first respectively, so that $P_{LR}(n) + P_{RL}(n) = 1$.

The configuration of the elementary trees affects the possible parse trees that can be derived. Figure 4a shows a parse tree where the elementary trees only allow right-adjunction, which effectively simulates an n-gram. Figure 4b shows another parse tree where the elementary trees allow both left-adjunction and right-adjunction. In this case, the tree-insertion grammar is able to represent hierarchical language structures that n-grams are unable to capture.

3 Method

The experiments were conducted within the ASR framework CMU Pocket-Sphinx (Lamere et al., 2003). It was chosen because it is released as open source, is light-weight for an ASR system, and supports n-grams up to the level of quint-grams. In the initial experiments, the language models were trained on an unannotated corpus, consisting of $10,000$ sentences collected from the English section of Wikipedia. The corpus contains $190,600$ words, divided over $28,500$ unique tokens. The 100 most common tokens (e.g., *the*, *of*, and *and*) make up half of

the corpus, whereas two-thirds of the tokens are seen only once (e.g., *sideward*, *cylindrical*, and *boreal*).

In later experiments, a medium-sized corpus of 20,000 sentences (i.e. an additional 10,000 beyond the original small dataset) and a larger corpus of 50,000 sentences (i.e. with 30,000 sentences added to the medium-sized corpus) were similarly collected. The 20,000 sentence set used over 44,000 unique tokens, the 50,000 sentence one used over 79,000.

3.1 Training

The n-gram models for $n = 1 - 5$ were read off the corpus by counting frequencies, and tuned with Witten-Bell smoothing. Inter-sentence punctuation was discarded and no distinction was made between uppercase and lowercase characters, because these cues are not accessible from the input audio in most speech recognition systems. This means that the words and abbreviations *AU*, *Au*, and *au* were all represented as *au*, and the sentence "Guatemala qualified a full team of 4 athletes, 2 men and 2 women." as "guatemala qualified a full team of 4 athletes 2 men and 2 women".

For the PLTIG, we started from a set of prototypical trees. The set contains a single initial tree connected to the empty lexical, representing the start of a sentence. For each lexical entry we also included a left-auxiliary and a right-auxiliary tree (see Figure 5 for an illustration). Like Hwa et al., we use the left-one right-two (L1R2) insertion paradigm, permitting (but not enforcing) one left adjunction and two right adjunctions into each auxiliary tree (see Figure 6 for an illustration). The PLTIG was then trained through an Expectation-Maximization (EM) process. The EM training algorithm (Dempster, Laird, & Rubin, 1977) is a hill-climbing algorithm which is used when inducing PLTIGs. The algorithm is based on maximum likelihood estimation (MLE) and determines the adjunction probabilities of a locally optimal grammar. A parser based on the Cocke-Younger-Kasami (CYK) algorithm was used to parse the induced grammar. Data sparsity is also a problem for PLTIGs, so the resulting model is smoothed through linear interpolation (Hwa, 2001).

3.2 Evaluation

For the evaluation, a fresh set of 150 sentences were randomly selected from the English version of Wikipedia. Then, a separate corpus was created for each sentence s, containing on average approx. 3,200 similar sentences (a total of 479,213 sentences). Each alternative sentence was obtained by replacing a word in s with a different word, also in the original vocabulary and at most two edits (i.e., letter insertions, deletions, or replacements) from the original. This would for example turn the template sentence

guatemala qualified a full team of 4 athletes 2 men and 2 women

into the alternatives

guatemala qualified a mule team of 4 athletes 2 men and 2 women

guatemala qualified 88 full team of 4 athletes 2 men and 2 women

guatemala qualified ham full team of 4 athletes 2 men and 2 women

The corpora were manually checked to make sure that the substituted words were inappropriate for their context. A few exceptions may have been overlooked, but in the vast majority of the cases, the alternatives were inferior.

The six language models, i.e. the unigram through quintgram ones and the PLTIG one, were used to compute the probability of each original sentence and its alternatives. The probabilities of the alternatives where then compared against the probability of the original sentence, and the number of sentences with higher and lower probability, respectively, were recorded.

4 Results

Table 1: The percentage of correctly completed evaluation tasks for differently sized data sets (measured in no. sentences).

	10,000	20,000	50,000
PLTIG	96.6 %	97.2 %	97.8 %
unigram	96.5 %	96.9 %	97.0 %
bigram	97.5 %	97.9 %	98.2 %
trigram	97.4 %	97.8 %	98.3 %
quadgram	97.5 %	97.8 %	98.3 %
quintgram	97.5 %	97.8 %	98.3 %

The first column of Table 1 shows the outcome of the experiments on the $10,000$-sentence dataset. The number reported for each language model (LM) is the percentage of correctly completed evaluation tasks. As the reader may recall from Section 3.2, these consist in selecting the most likely sentence out of a set of similar but inferior alternatives. The PLTIG language model performs worse than most n-gram models, achieving 96.6 % correct results compared to unigrams with 96.5 % and quintgrams with 97.5 % correct results.

As the data set was comparatively small, it cannot be expected saturate the more sophisticated language models. In particular the higher-level n-gram models and the PLTIG model are sensitive to over training because of their complexity. The experiments were therefore repeated for two larger datasets, a medium-sized one of $20,000$ sentences and the larger one of $50,000$ sentences (described in Section 3). For the medium-sized dataset, the difference between

PLTIGs and n-grams shrunk to less than one percentage point—97.2 % for PLTIGs versus 97.9 % for the best-performing n-gram model. For the largest dataset, PLTIGs perform almost as well as the best-performing n-gram model.

The scores of the PLTIG and n-gram language models for different training data sets as given in Table 1 are depicted visually in the left graph of Figure 7. The right graph shows the *difference* between the score of each model and the best-performing model trained on the same data set. This graph shows that the bigram model scores best for the small and medium-sized data set, but that the trigram model comes out on top for the larger data set. The graphs also suggest that while the PLTIG LM is initially inferior to the higher order n-gram models, it benefits more from added data, so the gap to the best-scoring n-gram decreases rather rapidly.

Another way to evaluate the language models is to look at their perplexity relative to the test sentences. This is simply the cross-entropy between the probability distributions predicted by the models and those embodied by the test data. Although perplexity is more loosely correlated with performance in speech recognition system, the general rule is that lower perplexity implies better prediction. Given the likelihood $P(w)$ of a sentence w with respect to a language model M, we compute the perplexity $PP(w)$ of M on w as $\sqrt[n]{1/P_M(w)}$ where $n = |w|$.

The average perplexities of the PLTIG and n-gram models are given in Table 8. We see that on average, the 'surprise' expressed by the PLTIG model is a factor 25 times higher that expressed by the n-grams models. Despite this, the PLTIG model is competitive in terms of prediction, at least for the larger data sets. Among the n-gram models, the correlation between preplexity and predictive power is stronger, though not perfect: For example, for the largest data set, the quadgram has the lowest perplexity, but the trigram the highest accuracy.

We must also remember that perplexity is strongly affected by smoothing, which decides how much probability mass to reserve for unseen constructions. The n-gram models in CMU sphinx are smoothed with Witten-Bell, but this method is not applicable to PLTIGs, so linear interpolation is used instead (see Section 3). We leave it as an open question to investigate what impact different choices of smoothing has on the perplexity.

In the cases where the models assigned a higher probability to a modified sentence, it is instructive to look at the words that differed between the sentences, and which caused the variation to receive a higher probability. We consider in particular words most often erroneously replaced or substituted in, for the PLTIG model and the trigram model. The top favored words substituted in are *'in', 'of', 'a', 'to', 'the', 'is', 'on', 'he', 'as', 'and'* for the PLTIG model, and for the trigram model they are *'a', 'in', 'of', 'on', 'to', 'at', 'is', 'as', 'the', 'he'*. The top favored words for each consist mostly of common two-letter words, and to a lesser degree single-letter or three-letter ones. This is because with only two edit operations, there are more ways of turning short lexical entries into other short lexical entries, than there are of turning long lexical entries into other

valid entries. For instance, every two-letter word can be turned into every other two-letter word.

The top unfavored words erroneously replaced on the other hand, mostly consist of short uncommon words. For the PLTIG model, the top unfavored words erroneously replaced are *'au', 'pan', 'fit', 'jun', 'on', 'bit', 'mono', 'it', 'bad', 'pit'*, for the trigram model they are *'au', 'cpr', 'jun', 'mono', 'bit', 'pan', 'ani', 'pit', 'fit', 't'*. These are also short, as our generation process puts them at within edit distance two from the original, but most of them are rare in literary text. Wikipedia is full of acronyms and abbreviations[1], but these are typically much rarer than common short words such as those at the top of the favored lists, yet often within edit distance two from such words.

5 Conclusion and future work

In our experiments, the n-gram models out-performed the PLTIG model on the smaller data set, but as the size of data grew, the PLTIG model gave comparable results, and its score grew faster with increasing data set size than that of the n-gram models. A natural question to ask is whether the PLTIG model will eventually overtake the n-grams. Judging by the rate with which the accuracies are improving, this will likely happen when the data set contains a couple of hundred thousand sentences, if it happens at all. As parts of the code base used for the experiments are old and not very efficient, experiments on this scale will require a substantial but likely rewarding re-implementation. If it indeed turns out that PLTIGs surpass n-grams at language modeling, then the fact that they provide structural information about sentences will make them an attractive alternative for practical language processing.

The introduction of finite-state machinery has led to improvements in several NLP tasks, not least part-of-speech tagging and parsing (Maletti, 2015). In the current attempt, all internal nodes are labelled with the same non-terminal symbol (i.e., state), so the possiblity of distinct internal symbols is not exploited. We are therefor interested in learning strategies that infer richer auxiliary trees, in the sense that they make use of different internal labels to propagate information and improve performance. This could possibly be accomplished through a split and merge approach (Petrov, 2012), given that the computational complexity does not run out of hand. Similarly, one could attempt to infer the number of adjunction sites and the insertion-strategy that offers the best trade-offs between parsing complexity and precision.

[1] For example, 'au' is the name of, or short form for, 70 different entities, including gold, absorbance unit, Australia, an audio format by Sun's Microsystem, Aarhus University in Denmark, a district in Munich, a Japanese telecom company, a creature in Vietnamese mythology, a Chinese family name, the Hawaiian name for the Pacific blue marlin, and the series "The Age of Ultron" by Marvel Comics.

References

Abeillé, A., Schabes, Y., & Joshi, A. K. (1990). Using lexicalized tags for machine translation. In *Proceedings of the 13th conference on computational linguistics* (pp. 1–6). Stroudsburg, PA, USA: Association for Computational Linguistics.

Allison, B., Guthrie, D., & Guthrie, L. (2006). Another look at the data sparsity problem. In *Text, speech and dialogue* (pp. 327–334). Berlin Heidelberg: Springer Verlag.

Baker, J. (1975). The Dragon system – an overview. *IEEE Transactions on Acoustics, Speech and Signal Processing*, *23*(1), 24–29.

Banko, M., & Brill, E. (2001). Mitigating the paucity-of-data problem: Exploring the effect of training corpus size on classifier performance for natural language processing. In *Proceedings of the first international conference on human language technology research* (pp. 1–5). Stroudsburg, PA, USA: Association for Computational Linguistics.

Bell, T. C., Cleary, J. G., & Witten, I. H. (1990). *Text compression.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 1–38.

Forgie, J. W., & Forgie, C. D. (1959). Results obtained from a vowel recognition computer program. *The Journal of the Acoustical Society of America*, *31*(11), 1480–1489.

Goodman, J. T. (2001). A bit of progress in language modeling. *Computer Speech & Language*, *15*(4), 403 – 434.

Hwa, R. (1998). An empirical evaluation of probabilistic lexicalized tree insertion grammars. In *Proceedings of the 36th annual meeting of the association for computational linguistics 1998* (pp. 557–563). Stroudsburg, PA, USA: Association for Computational Linguistics.

Hwa, R. (2001). *Learning probabilistic lexicalized grammars for natural language processing* (Unpublished doctoral dissertation). Harvard University, Cambridge, MA, USA.

Jelinek, F. (1976). Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, *64*(4), 532-556.

Joshi, A. K., Levy, L. S., & Takahashi, M. (1975). Tree adjunct grammars. *Journal of computer and system sciences*, *10*(1), 136–163.

Klakow, D. (1998). Log-linear interpolation of language models. In *Proceedings of the international symposium on chinese spoken language processing* (pp. 1695–1698). ISCA.

Lamere, P., Kwok, P., Walker, W., Gouvea, E., Singh, R., Raj, B., & Wolf, P. (2003). Design of the CMU Sphinx-4 decoder. In *Eight European conference on speech communication and technology.* ISCA.

Leonard E. Baum, T. P. (1966). Statistical inference for probabilistic functions of finite state markov chains. *The Annals of Mathematical Statistics*, *37*(6), 1554-1563.

Maletti, A. (2015, August). *Finite-state technology in natural language processing.* Umeå, Sweden. (Keynote Address at the 20th International Conference on Implementation and Application of Automata (CIAA 2015))

Marill, T. (1961). Automatic recognition of speech. *IRE Transactions on Human Factors in Electronics*(1), 34–38.

Michel, J.-B., Shen, Y. K., Aiden, A. P., Veres, A., Gray, M., The Google Books Team, ... Aiden, E. L. (2010). Quantitative analysis of culture using millions of digitized books. *Science*.

Petrov, S. (2012). *Coarse-to-fine natural language processing.* Berlin Heidelberg: Springer Verlag.

Pierce, J. R. (1969). Whither speech recognition? *The Journal of the Acoustical Society of America*, *46*(4B), 1049-1051.

Resnik, P. (1992). Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In *Proceedings of the 14th conference on computational linguistics* (pp. 418–424). Stroudsburg, PA, USA: Association for Computational Linguistics.

Schabes, Y., & Waters, R. C. (1993a). Lexicalized context-free grammars. In *Proceedings of the 31st annual meeting on association for computational linguistics* (pp. 121–129). Stroudsburg, PA, USA: Association for Computational Linguistics.

Schabes, Y., & Waters, R. C. (1993b). *Stochastic lexicalized context-free grammar* (Tech. Rep. No. 12). Cambridge, MA: Mitsubishi Electric Research Laboratories.

Schabes, Y., & Waters, R. C. (1994). *Tree insertion grammar: a cubic-time parsable formalism that lexicalizes context-free grammar without changing the trees produced* (Tech. Rep. No. 13). Cambridge, MA: Mitsubishi Electric Research Laboratories.

Shieber, S. M., & Schabes, Y. (1990). Synchronous tree-adjoining grammars. In *Proceedings of the 13th conference on computational linguistics* (pp. 253–258). Stroudsburg, PA, USA: Association for Computational Linguistics.

Smith, C. P. (1951). A phoneme detector. *The Journal of the Acoustical Society of America*, *23*(4), 446–451.

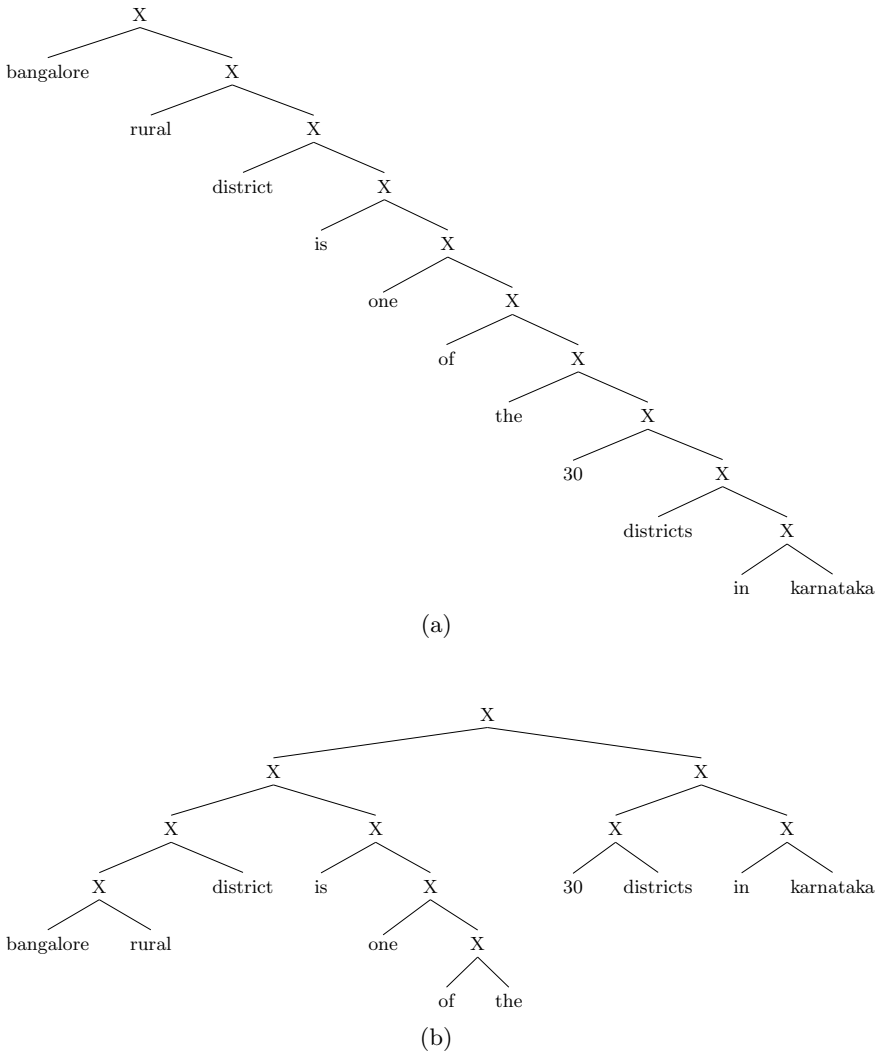
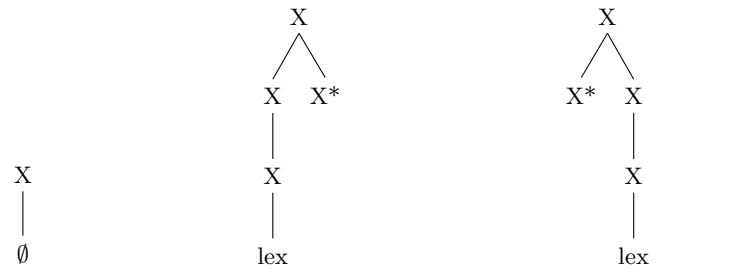
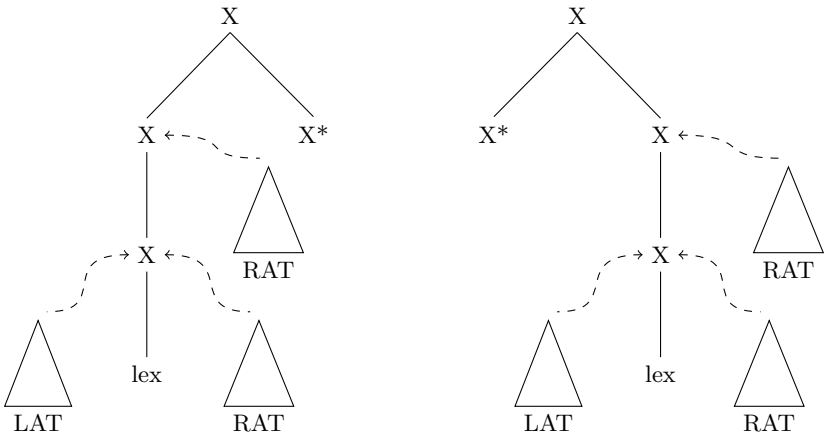


Fig. 4: Two alternative derivations of a structural tree for the sentence "Bangalore Rural District is one of the 30 districts in Karnataka."



(a) Initial tree (b) Left-auxiliary tree (c) Right-auxiliary tree

Fig. 5: The main elementary tree types. An initial tree (a) with a single empty lexical item, a left-auxiliary tree (b) and a right-auxiliary tree (c).



(a) L1R2 left-auxiliary tree (b) L1R2 right-auxiliary tree

Fig. 6: The L1R2 insertion paradigm allows at most one left and two right adjunctions into each auxiliary tree. In the figure, LAT and RAT are short for left-auxiliary and right-auxiliary tree, respectively.

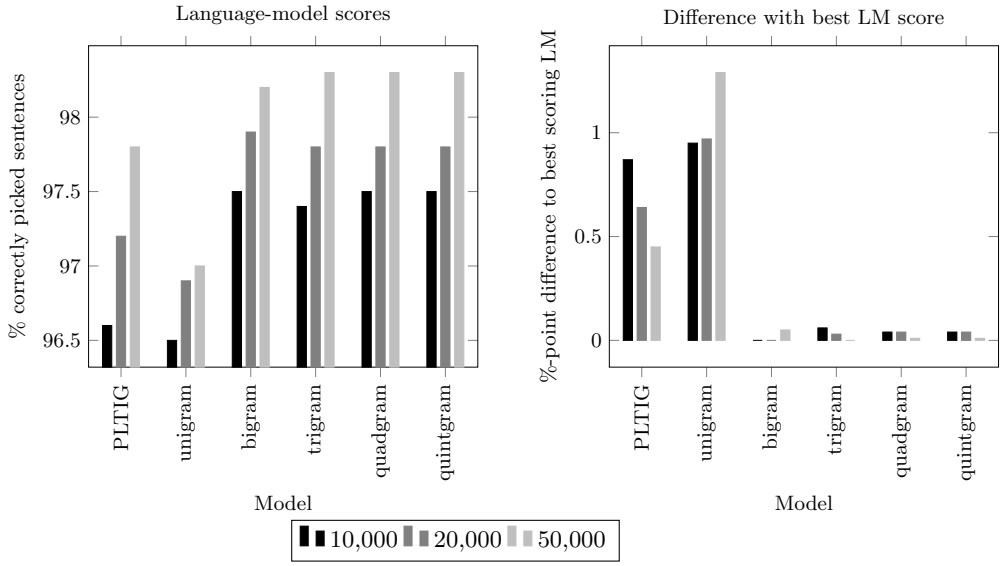


Fig. 7: LM scores picking correct sentence out of test set, and distance of particular LM's score to best-scoring LM's score, for LMs trained on data sets of $10,000$, $20,000$, $50,000$ sentences.

Fig. 8: The average perplexities of the n-gram and PLTIG language models on the sentences in the test set.

	PLTIG	unigram	bigram	trigram	quadgram	quintgram
10,000 sent.	49,787	2,126	1,710	1,809	1,826	1,828
20,000 sent.	50,499	2,096	1,208	1,295	1,320	1,323
50,000 sent.	51,505	2,146	1,013	1,079	1,105	1,109