A Structured Language Model based on Context-Sensitive Probabilistic Left-Corner Parsing

Dong Hoon Van Uytsel[†] donghoon@esat.kuleuven.ac.be
Filip Van Aelten[‡] filip.van.aelten@lhs.be
Dirk Van Compernolle[†] compi@esat.kuleuven.ac.be

[†]Katholieke Universiteit Leuven, ESAT, Belgium
[‡]Lernout & Hauspie, Belgium

Abstract

Recent contributions to statistical language modeling for speech recognition have shown that probabilistically parsing a partial word sequence aids the prediction of the next word, leading to "structured" language models that have the potential to outperform n-grams. Existing approaches to structured language modeling construct nodes in the partial parse tree after all of the underlying words have been predicted. This paper presents a different approach, based on probabilistic left-corner grammar (PLCG) parsing, that extends a partial parse both from the bottom up and from the top down, leading to a more focused and more accurate, though somewhat less robust, search of the parse space. At the core of our new structured language model is a fast context-sensitive and lexicalized PLCG parsing algorithm that uses dynamic programming. Preliminary perplexity and word-accuracy results appear to be competitive with previous ones, while speed is increased.

1 Structured language modeling

In its current incarnation, (unconstrained) speech recognition relies on a left-to-right language model L, which estimates the occurrence of a next word w_j given a sequence of preceding words $c_j = w_0^{j-1}$ (the context):[1]

$$L(w_j|c_j) = \hat{p}(w_j|c_j).$$

L is called a language model (LM).

Obviously the context space is huge and even in very large training corpora most contexts never occur, which prohibits a reliable probability estimation. Therefore the context space needs to be mapped to a much smaller space, such that only the essential information is retained. In spite of its simplicity the trigram LM, that reduces c_j to w_{j-2}^{j-1}, is hard to improve on and still the main language model component in state-of-the-art speech recognition systems. It is therefore commonly used as a baseline in the evaluation of other models, including the one described in this paper.

Structured language models (SLM) introduce parsing into language modeling by alternating between predicting the next word using features of partial parses of the context and extending the partial parses to cover the next word. Following this approach, Chelba and Jelinek (2000) obtained a SLM that slightly improves on a trigram model both in perplexity and recognition performance. The Chelba-Jelinek SLM is, to our knowledge, the first left-to-right LM using parsing techniques that is successfully applied to large vocabulary speech recognition. It is built on top of a lexicalized probabilistic shift-reduce parser that predicts the next word from the headwords ("exposed" heads) and categories of the last two predicted isolated constituents of the context. Then the predicted word becomes the last isolated constituent and the last two constituents are repeatedly recombined until the parser decides to stop.

A dynamic programming (DP) version of Chelba's parser, inspired on the CYK chart parser, was proposed in (Jelinek and Chelba, 1999). Our implementation is roughly quadratic in the length of the sentence, but not significantly faster than Chelba's non-DP parser. It scored somewhat lower in perplexity before reestimation (presumably by avoiding search errors), but remained roughly at the same level after full inside-outside reestimation (Van Aelten and Hogenhout, 2000).

An obvious weakness of the Chelba-Jelinek SLM is the bottom-up behavior of the parser: it creates isolated constituents and only afterwards is it able to check whether a constituent fits into a higher structure. Van Uytsel (2000) developed a top-down al-

[1]As a shorthand, w_a^b denotes a sequence $w_a w_{a+1} \dots w_b$ if $b \geq a$, else it is the empty sequence.

ternative along similar lines but based on a lexicalized and context-sensitive DP version of an efficient Earley parser (Stolcke, 1995; Jelinek and Lafferty, 1991). The Earley-based SLM performed worse than the Chelba-Jelinek SLM, mostly due to the fact that the rule production probabilities cannot be conditioned on the underlying lexical information, thus producing a lot of wrong parses.

The weaknesses of our Earley SLM have led us to consider probabilistic left-corner grammar (PLCG) parsing (Manning and Carpenter, 1997), which follows a mixed bottom-up and top-down approach. Its potential to enhance parsing efficiency has been recognized by Roark and Johnson (2000), who simulated a left-corner parser with a top-down best-first parser applying a left-corner-transformed PCFG grammar. For the language model described in this paper, however, we implemented a DP version of a native left-corner parser using a left-corner treebank grammar (containing projection rules instead of production rules). The efficiency of our implementation further allowed to enrich the history annotation of the parser states and to apply a lexicalized grammar.

The following section contains a brief review of Manning's PLCG parser. Section 3 describes how it was adapted to our SLM framework: we introduce lexicalization and context-sensitivity, present a DP algorithm using a chart of parser states and finally we define a language model based on the adapted PLCG parser. At the end of the same section we explain how the initial language model can be trained on additional plain text through a variant of inside-outside reestimation. In section 4 we evaluate a few PLCG-based SLMs obtained from the Penn Treebank and BLLIP WSJ Corpus. We present test set perplexity measurements and word accuracy after n-best list rescoring to assess their viability for speech recognition.

2 Classic PLCG parsing

The parameters of a PLCG are called *projection* probabilities. They are of the form

$$p(Z \to X\ \alpha | X, G),$$

to be read as "given a completed constituent X dominated by a goal category G, the probability that there is a Z that has X as its first daughter and α as its next daughters". A PLCG contains essentially the same rules as a probabilistic context-free grammar (PCFG), but the latter conditions the rule probabilities on the mother category Z (*production* probabilities). In both cases the joint probability of the entire parse tree and the parsed sentence is the product of the production resp. projection probabilities of the local trees it consists of.

While PCFG parsing proceeds from the top down or from the bottom up, PLCG naturally leads to a parsing scheme that is a mixture of both. The advantages of this are made clear in the subsections below. Formally, a PLCG parser has three elementary operations:

- SHIFT: given that an unexpanded constituent G starts from position i, shift the next word w_i with probability $p_s(w_i|G)$ (G is called the *goal category*);
- PROJECT: given a complete constituent X, dominated by a goal category G, starting in position i and ending in j, predict a mother constituent Z starting in position i and completed up till position j, and zero or more unexpanded sister constituents α starting in j with probability $p_p(Z \to X\ \alpha | X, G)$;
- ATTACH: given a complete constituent X dominated by a goal category G, identify the first as the latter with probability $p_a(X, G)$.

3 Extending the PLCG framework

3.1 Synchronous chart parsing with PLCG

In this subsection we present the basic parsing algorithm and its data structures and operations. In the subsections that follow, we will introduce lexicalization and context-sensitivity by extending this framework.

The PLCG parsing process is interpreted as a search through a network of states, a compact representation of the search space. The network nodes correspond to states and the arcs to operations (annotated with transition probabilities). A (partial) parse corresponds to a (partial) path through the network. The joint probability of a partial parse and the covered part of the sentence is equal to the partial path probability, i.e. the product of the probabilities of the transitions in the path.

3.1.1 PLCG states

We write a state q as

$$q = (G; Z \to_i X \star_j \beta; \mu, \nu) \qquad (1)$$

where G is the goal category, Z is the category of a constituent from position i complete up till position

j, X is the first daughter category, β denotes the remaining unresolved daughters of Z, and μ and ν are forward and inner probabilities defined below. The wildcard $\star$ symbolizes zero or more resolved daughter categories: we make abstraction of the identities of resolved daughters (except the first one), because further parser moves do not depend on them. If β is empty, q is called a *complete* state, otherwise q is a *goal* state.

3.1.2 Forward and inner probability

Given a state q as defined in (1). We define its *forward* probability $\mu = \mu(q)$ as the sum of the probabilities of the paths ending in q, starting in the initial state and generating w_0^{j-1}. As a consequence, $\mu(q) = p(w_0^{j-1}, q)$ (joint probability).

The *inner* probability $\nu = \nu(q)$ is the sum of the probabilities of the paths generating w_i^{j-1}, ending in q and starting with a SHIFT of w_i. As a consequence, $\nu(q) = p(w_i^{j-1}, q)$.

Note that the forward and inner probabilities of the final state should be identical and equal to $p(S)$.

3.1.3 Parser operations

In this paragraph we reformulate the classic PLCG parser operations in terms of transitions between states. We hereby specify update formulas for forward and inner probabilities.

Shift The SHIFT operation starts from a goal state

$$q = (G; Z \rightarrow_i X \star_j Y \beta; \mu, \nu) \tag{2}$$

and shifts the next word w at position j of the input by updating q' or generating a new state q' where[2]

$$q' = (Y; \mathsf{W} \rightarrow_j w \star_{j+1}; \mu' \mathrel{+}= \mu p, \nu' = p) \tag{3}$$

with transition probability

$$p = p_{\mathrm{s}}(w|Y). \tag{4}$$

If q' already lives in the chart, only its forward probability is updated. The given update formula is justified by the relation

$$\mu(q') = \sum_{q \Rightarrow_{\mathrm{s}} q'} \mu(q) p(q \Rightarrow q')$$

where the sum is over all SHIFT transitions from q to q' and $p(q \Rightarrow q')$ denotes the transition probability from q to q'. Computing $\nu(q')$ is a trivial case of the definition.

[2]The C-like shorthand notation $\mu' \mathrel{+}= \mu p$ means that μ' is set to μp if there was no q' in the chart yet, otherwise μ' is incremented with μp.

Projection From a complete state, two transitions are possible: ATTACH to a goal state with a probability p_{a} or PROJECT with a probability $1 - p_{\mathrm{a}}$. PROJECT starts from a complete state

$$q = (G; Z \rightarrow_i X \star_j; \mu, \nu) \tag{5}$$

and generates or updates a state

$$q' = (G; T \rightarrow_i Z \star_j \alpha; \mu' \mathrel{+}= \mu p, \nu' \mathrel{+}= \nu p) \tag{6}$$

with transition probability

$$p = p_{\mathrm{p}}(T, \alpha | Z, G) \cdot (1 - p_{\mathrm{a}}(Z, G)). \tag{7}$$

Again, the forward probability is computed recursively as a sum of products. Now ν' needs to be accumulated, too: the constituent Z in general may be resolved with more than one different X, which each time adds to ν'.

Note that a mother constituent inherits G from her first daughter (left-corner).

Attachment Given a complete state q as in (5) where $G = Z$ and some goal state q'' in the partial path leading to q

$$q'' = (G''; T \rightarrow_h U \star_i Z \beta; \mu'', \nu'') \tag{8}$$

then the ATTACH operation is a transition from q to q' with

$$q' = (G''; T \rightarrow_h U \star_j \beta; \mu' \mathrel{+}= \mu'' \nu p / \nu'', \nu' \mathrel{+}= \nu p) \tag{9}$$

and transition probability

$$p = p_{\mathrm{a}}(Z, G) \cdot \nu''. \tag{10}$$

Why can μ' not be updated from μ, similarly to (3) and (6)? The reason is that ATTACH makes use of non-local constraints: the transition from q to q' is only possible if a matching goal state q'' occurred in a path leading to q. Therefore computing μ as in (3) and (6) would include all paths that generate q', also those that do not contain q''. Instead, the update of μ' in (9) combines all paths leading to q'' with the paths starting from q'' and ending in q. The update of ν' follows an analogous reasoning.

3.1.4 Chart representation

The parser produces a set of states that can be conveniently organized in a staircase-shaped chart similar to the one used by the CYK parser. In the chart cell with coordinates (i, j) we store all the states starting in i and completed up till position j.

3.1.5 Synchronous parsing algorithm

Following (Chelba, 2000), we represent a sentence by a sequence of word identities starting with a sentence-begin token ⟨s⟩, that is used in the context but not predicted, followed by a sentence-end token ⟨/s⟩, that is predicted by the model. We are collecting the sentence proper together with ⟨/s⟩ under a node labeled TOP′, and the TOP′ node together with ⟨s⟩ under a TOP node. The parser starts from the initial state

$$q_{\mathrm{I}} = (\mathsf{TOP}; \mathsf{TOP}/\langle\mathsf{s}\rangle \rightarrow_{-1} \mathsf{SB}/\langle\mathsf{s}\rangle \star_0 \mathsf{TOP}'; 1, 1). \tag{11}$$

After processing the sentence $S = w_0^{N-1}$ and provided a full parse was found, the final state

$$q_{\mathrm{F}} = (\mathsf{TOP}; \mathsf{TOP}/\langle\mathsf{s}\rangle \rightarrow_{-1} \mathsf{SB}/\langle\mathsf{s}\rangle \star_N; p(S), p(S)) \tag{12}$$

is found in cell $(-1, N)$.

Now we are ready to formulate the parsing algorithm. Note that we treat an ATTACH operation as a special PROJECT, as explained in Sec. 4.1.

```
for $j \leftarrow 0, 1$ to $N$
  for $i \leftarrow j-1, j-2$ to $-1$
    foreach complete state $q$ in cell $(i, j)$
      foreach proj in projections($q$)
        if goal($q$) = cat($q$) and proj = 'attach'
          for $h \leftarrow i-1, i-2$ to $-1$
            foreach goal state $m$ in cell $(h, i)$ matching $q$
              $q' \leftarrow$ ATTACH$(q, m)$
              add $q'$ to cell $(h, j)$
        else
          $q' \leftarrow$ PROJECT$(q)$
          add $q'$ to cell $(i, j)$
        if $q'$ is complete, recursively add further projections/attachments
  if $j = N$
    break
  for $i \leftarrow -1, 0$ to $j-1$
    foreach goal state $q$ in cell $(i, j)$
      $q' \leftarrow$ SHIFT$(q, w_j)$
      add $q'$ to cell $(j, j+1)$
```

3.2 Lexicalization and context-sensitivity

Probably the most important shortcoming of PCFG's is the assumption of context-free rule probabilities, i.e. the probability distribution over possible righthand sides given a lefthand side is independent from the function or position of the lefthand side. This assumption is quite wrong. For instance, in the Penn Treebank an NP in subject position produces a personal pronoun in 13.7% of the cases, while in object position it only does so in 2.1% of the cases (Manning and Carpenter, 1997). Furthermore, findings from corpus-based linguistic studies and developments in functional grammar indicate that the lexical realization of a context, besides its syntactic analysis, strongly influences patterns of syntactic preference. Today's best automatic parsers are made substantially more efficient and accurate by applying lexicalized grammar (Manning and Schütze, 1999).

3.2.1 Context-sensitive and lexicalized states

In our work we did not attempt to find semantic generalizations (such as casting a verb form to its infinitive form or finding semantic attributes); our simple (but probably suboptimal) approach, borrowed from (Magerman, 1994; Collins, 1996; Chelba, 2000), is to percolate words upward in the parse tree in the form in which they appear in the sentence. In our experiments, we opted to hardcode the head positions as part of the projection rules.[3] The nodes of the resulting partial parse trees thus are annotated with a category label (the CAT feature) and a lexical label (the WORD feature).

The notation (1) of a state is now replaced with

$$q = (G, L_1, L_2; Z/z \rightarrow_i X/x \star_j \beta; \mu, \nu) \tag{13}$$

where z is the WORD of the mother (possibly empty), x is the WORD of the first daughter (not empty), and the extended context contains

- G = CAT of a goal state q_g;
- L_1 = (CAT, WORD) of the state q_1 projecting q_g;
- L_2 = (CAT, WORD) of the state q_2 projecting a goal state dominating q_1.

If the grammar only contains unary and binary rules, L_1 and L_2 correspond with Chelba's concept of *exposed heads* — which was in fact the idea behind the definition above. The mixed bottom-up and top-down parsing order of PLCG allows to condition q on a goal constituent G higher up in the partial tree containing q; this turns out to significantly improve efficiency with respect to Jelinek's bottom-up chart parser.

[3]Inserting a probabilistic head percolation model, as in (Chelba, 2000), may be an alternative.

3.2.2 Extended parser operations

In this section, we extend the parser operations of Sec. 3.1.3 to handle context-sensitive and lexicalized states. The forward and inner probability update formulas remain formally the same and are not repeated here.

The SHIFT operation $q \Rightarrow_{\rm s} q'$ is a transition from q to q' with probability p where

$$q = (G, L_1, L_2;\, Z/z \to_i X/x \star_j Y\,\beta;\, \mu, \nu) \quad (2')$$

$$q' = (Y, X/x, L_1;\, \mathsf{W}/w \to_j \mathsf{W}/w \star_{j+1};\, \mu', \nu') \quad (3')$$

$$p = p_{\rm s}(w_j|q). \quad (4')$$

The PROJECT operation $q \Rightarrow_{\rm p} q'$ is a transition from q to q' with probability p where

$$q = (G, L_1, L_2;\, Z/z \to_i X/x \star_j;\, \mu, \nu) \quad (5')$$

$$q' = (G, L_1, L_2;\, T/t \to_i Z/z \star_j \alpha;\, \mu', \nu') \quad (6')$$

$$p = p_{\rm p}(T, \alpha|q) \cdot (1 - p_{\rm a}(q)) \quad (7')$$

If Z is in head position, $t = z$; otherwise t is left unspecified.

The ATTACH operation $q \Rightarrow_{\rm a} q'$ is a transition from q to q' given q'' with a probability p where

$$q'' = (G, L_1, L_2;\, Z/z \to_h X/x \star_i Y\beta;\, \mu'', \nu'') \quad (8')$$

$$q = (Y, X/x, L_1;\, Y/y \to_i T/t \star_j;\, \mu, \nu)$$

$$q' = (G, L_1, L_2;\, Z/z' \to_h X/x \star_j \beta;\, \mu', \nu') \quad (9')$$

$$p = p_{\rm a}(q) \cdot \nu'' \quad (10')$$

If Y is in head position, $z' = y$; otherwise, $z' = z$.

3.3 PLCG-based language model

A language model (LM) is a word sequence predictor (or an estimator of word sequence probabilities). Following common practice in language modeling for speech recognition, we predict words in a sentence from left to right[4] with probabilities of the form $p(w_j|w_0^{j-1})$. Suppose the parser has worked its way through w_0^{j-1} and is about to make w_j-SHIFT transitions. Then we can write

$$p(w_j|w_0^{j-1}) = \sum_{q \in \mathcal{G}_j} p(w_j|q)\,p(q|w_0^{j-1}). \quad (14)$$

[4]Since this allows the language model to be applied in early stages of the search.

where $\mathcal{G}_j$ is the set of goal states in position j. The factor $p(w_j|q)$ is given by the transition probability associated with the SHIFT operation.[5]

On the other hand, note that

$$\sum_{q \in \mathcal{G}_j} \mu(q) = \sum_{q \in \mathcal{W}_j} \mu(q) = p(w_0^{j-1}) \quad (15)$$

where $\mathcal{W}_j$ is the set of states in position j that resulted from SHIFT operations. The first equation holds because there are only PROJECT and ATTACH transitions between the elements of $\mathcal{W}_j$ and $\mathcal{G}_j$, since the sum of outgoing transitions from each state in that region equals 1 and therefore the total probability mass is preserved. By inserting (15) into (14) we obtain

$$p(w_j|w_0^{j-1}) = \frac{\sum_{q \in \mathcal{G}_j} p(w_j|q)\mu(q)}{\sum_{q \in \mathcal{G}_j} \mu(q)}. \quad (16)$$

3.4 Model reestimation

The $p_{\rm p}$, $p_{\rm s}$ and $p_{\rm a}$ submodels can be reestimated with iterative expectation-maximization, which needs the computation of frequency expectations. For this purpose we define the *outer probability* of a state q, written as $\xi(q)$, as the sum of probabilities of precisely that part of the paths that is not included in the inner probability of q. The outer probability of a complete state is analogous to Baker's (1979) definition of an outside probability.

The outer probabilities are computed in the reverse direction starting from $q_{\rm F}$, provided that a list of backward references were stored with each state ($\xi(q') \equiv \xi'$, $\xi(q'') \equiv \xi''$):[6]

- $\xi(q_{\rm F}) = 1$.
- Reverse ATTACH (cfr. (8′, 9′, 10′)): $\xi \mathrel{+}= \xi' p$ and $\xi'' \mathrel{+}= \xi' \nu p/\nu''$. These formulas are made clear in Fig. 1.
- Reverse PROJECT (cfr. (5′, 6′, 7′)): $\xi \mathrel{+}= \xi' p$.
- A reverse SHIFT is not necessary, but could be used as a computational check.

[5]Consequently the computation of LM probabilities requires almost no extra work. A model $p(w_j|q)$ used in (14) different from $p_{\rm s}(w_j|q)$ used by the parser may be chosen however.

[6]Care has to be taken that an outer probability is complete before it propagates to other items. A topological sort could serve this purpose.

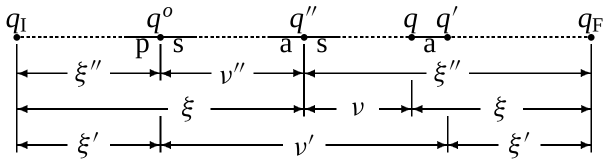

Figure 1: Relations between inner and outer probabilities along a single path at attachment of q to q'' resulting into q'.

Now the expected frequency of a transition $o \in \{\mathrm{s}, \mathrm{p}, \mathrm{a}\}$ from q to q' in a full parse of S is

$$E[\mathrm{Freq}(q \Rightarrow_o q' | S)] = \sum_{\text{all paths}} \Pr(\mathrm{path}|S) \mathrm{Freq}(q \Rightarrow_o q' | \mathrm{path}). \tag{17}$$

Since all full parses terminate in q_{F}, the final state, $\nu(q_{\mathrm{F}}) = \mu(q_{\mathrm{F}}) = \Pr(S)$. Therefore (17) is computable as

$$E[\mathrm{Freq}(q \Rightarrow_o q' | S)] = \begin{cases} \frac{1}{\nu(q_{\mathrm{F}})} \nu(q') \xi(q') & \text{if } o = \mathrm{s}, \\ \frac{1}{\nu(q_{\mathrm{F}})} \nu(q) p(q \Rightarrow_o q') \xi(q') & \text{else}. \end{cases} \tag{18}$$

The expected frequencies required for the reestimation of the conditional distributions are then obtained by summing (18) over the state attributes from which the required distribution is independent.

4 Empirical evaluation

4.1 Modeling

We have trained two sets of models. The first set was trained on sections 0–20 of the Penn Treebank (PTB) (Marcus et al., 1995) using sections 21–22 for development decisions and tested on sections 23–24. The second set was trained on the BLLIP WSJ Corpus (BWC), which is a machine-parsed (Charniak, 2000) version of (a selection of) the ACL/DCI corpus, very similar to the selection made for the WSJ0/1 CSR corpus. As the training set, we used the BWC minus the WSJ0/1 "dfiles" and "efiles" intended for CSR development and evaluation testing.

The PTB devset was used for fixing submodel parameterizations and software debugging, while perplexities are measured on the PTB testset. The BWC trainset was used in rescoring N-best lists in order to assess the models' potential in speech recognition. Both the PTB and BWC underwent the following preprocessing steps: (a) A vocabulary was fixed as the 10k (PTB) resp. 30k (BWC) most frequent words; out-of-vocabulary words were replaced by ⟨unk⟩. Numbers in Arabic digits were replaced by one token 'N'. (b) Punctuation was removed. (c) All characters were converted to lowercase. (d) All parse trees were binarized in much the same way as detailed in (Chelba, 2000, pp. 12–17); non-terminal unary productions were eliminated by collapsing two nodes connected by a unary branch to one node annotated with a combined label. This step allowed a simple implementation and comparison of results with related publications. We distinguished 1891 different projections, 143 different non-terminal categories and 41 different parts-of-speech. (e) All constituents were annotated with a lexical head using deterministic rules by Magerman (1994).

The training then proceded by decomposing all parse trees into sequences of SHIFT, PROJECT and ATTACH transitions. The submodels were finally estimated from smoothed relative counts of transitions using standard language modeling techniques: Good-Turing back-off (Katz, 1987) and deleted interpolation (Jelinek, 1997).

Shift submodel

The SHIFT submodel implements (4′). Finding a good parameterization entails fixing the features that should explicitly appear in the context and in which order, so that all information-bearing elements are incorporated, with limited data fragmentation. This is not a straightforward task. We went through an iterative process of intuitively guessing which feature should be added or removed from the context or changing the order, building a corresponding model and evaluating its *conditional* perplexity (CPPL) against the devset. The CPPL of a SHIFT submodel is its perplexity measured on a test set consisting of (context, word to be predicted) pairs (i.e. the SHIFT transitions according to a certain parameterization) extracted from the correct parse trees of a parsed test corpus. In other words, the CPPL is an underbound of the PPL in that it would be the PPL from an ideal parser. We finally concluded that the parameterization (notation being consistent with (2′))

$$p_{\mathrm{s}}(w | Y, x, L_1.\mathrm{WORD}), \tag{19}$$

where the conditioning sequence is ordered from most to least significant, is optimal for our purposes in the given experimental conditions. The CPPL of

Table 1: Word trigram (baseline) and PTB model perplexities.

	model	GT	DI
(a)	word trigram	190	193
(b)	PLCG-based LM	185	187
(c)	linear interpolation: .6(a) + .4(b)	159	166

this model on the PTB devset is 48, which displays the great potential of a correct syntactic partial parse to predict the next word.

Project/attach submodel

The ATTACH submodel can be incorporated into the PROJECT submodel by treating the attachment as a special kind of projection. This approach was systematically applied since it sped up parsing. Having the possibility to choose different parameterizations in separate PROJECT and ATTACH submodels did not lower perplexity and increased execution time. Therefore, we always used combined PROJECT/ATTACH submodels in further experiments.

The PROJECT/ATTACH submodel implements (7′) and (10′). The process of finding an appropriate parameterization used to build the SHIFT submodel was also applied here. Finally we concluded that the parameterization (notation being consistent with (5′))

$$p_{\mathrm{p}}(T, \alpha | Z, G, z) \qquad (20)$$

is optimal for our purposes in the given experimental conditions.

4.2 Evaluation of PTB models

Table 1 lists test set perplexities (excluding OOVs and unparsed parts of sentences) of Good-Turing smoothed back-off models (GT) and deleted-interpolation smoothed (DI) models trained on the PTB trainset and tested on the PTB testset. We observed similar results with both smoothing methods. As a baseline, word trigram (a) was trained and tested on the same material. The PPL obtained with the PLCG-based LM (b), using parametrizations (19) and (20), is not much lower than the baseline PPL.[7] Interpolation (c) with the baseline however yields a relative PPL reduction of 14 to 16% with respect to the baseline.

[7]Using parametrizations $p_{\mathrm{p}}(T, \alpha | z, G, L_1.\mathrm{CAT})$ for projection from W-items and $p_{\mathrm{p}}(T, \alpha | G, Z, X, z)$ for other projections, we recently obtained a PPL of 178 (and 155 when interpolated). This result is left out from the discussion in order to keep it clear and complete.

Table 2: WER results (%) after 100-best list rescoring on the DARPA WSJ Nov '92 evaluation test set, non-verbalized punctuation. The models are smoothed with Good-Turing back-off (WER results in column GT) or deleted interpolation (DI).

	rescoring model	GT	DI
(a)	DARPA word trigram	10.44	
(b)	BWC word trigram	11.31	11.08
(c)	BWC Chelba-Jelinek SLM		10.86
(d)	(a) and (c) combined		9.82
(e)	(b) and (c) combined		10.60
(f)	BWC PLCG-based SLM	11.45	11.48
(g)	(a) and (e) combined	9.85	9.87
(h)	(b) and (e) combined	10.38	10.58
(i)	Best possible	4.46	4.46

Parse accuracy is around 79% for both labeled precision and recall on section 23 of PTB (excluding unparsed sentences, about 4% of all sentences). In comparison, with our own implementation of Chelba-Jelinek, we measured a labeled precision and recall of 57% and 75% on the same input. These results seem fairly low compared to other recent work on large-scale parsing, but may be partly due to the left-to-right restriction of our language models,[8] which for instance prohibits word-lookahead. Moreover, while we measured accuracy against a binarized version of PTB, the original parses are rather flat, which may allow higher accuracies.

4.3 Evaluation of BWC-models

The main target application of our research into LM is speech recognition. We performed N-best list rescoring experiments on the DARPA WSJ Nov '92 evaluation test set, non-verbalized punctuation. The N-best lists were obtained from the L&H Voice Xpress v4 speech recognizer using the standard trigram model included in the test suite (20k open vocabulary, no punctuation).

In Table 2 we report word-recognition error rates (WER) after rescoring using Chelba-Jelinek and PLCG-based models. Both DI and GT smoothing methods yielded very comparable results. Due to technical limitations, all the models except the baseline trigram were trimmed by ignoring highest-order events that occurred only once.

The best PLCG-based SLM trained on the BWC train set (f) performs worse than the official word trigram (a). However, since the BWC does not completely cover the complete WSJ0 LM train material

[8]Not to be confused with left-to-right parsing.

and slightly differs in tokenization, it is more fair to compare with the performance of a word trigram trained on the BWC train set (b). Results (g) and (h) show that the PLCG-based SLM lowers WER with 4% relative when used in combination with the baseline models. A comparable result was obtained with the Chelba-Jelinek SLM (results (d) and (e)).

5 Conclusion and future work

The PLCG-based SLM exposes a slight loss of robustness in the reduced recognition rate when it is used as a stand-alone rescoring LM. Combined with a word trigram LM however, perplexity and WER reductions with respect to a word 3-gram baseline seem similar to those obtained with the Chelba-Jelinek SLM and those previously reported by Chelba (2000). On the other hand, the PLCG-based SLM is significantly faster and obtains a higher parsing accuracy.

In the future we plan to evaluate full EM reestimation of the models on the trainset using the formulas given in this paper.

Acknowledgements

The authors wish to thank Paul Vozila for discussing intermediate results and for providing the authors with the 100-best lists used for sentence rescoring. The authors are also indebted to Saskia Janssens and Kristin Daneels for their help with some of the experiments.

This research is supported by the Institute for the promotion of Innovation by Science and Technology in Flanders (IWT-Flanders), contract no. 000286.

References

James K. Baker. 1979. Trainable grammars for speech recognition. In Jared J. Wolf and Dennis H. Klatt, editors, *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pages 547–550. The MIT Press, Cambridge, MA.

Eugene Charniak. 2000. A maximum-entropy inspired parser. In *Proc. of the NAACL*, pages 132–139.

Ciprian Chelba. 2000. *Exploiting Syntactic Structure for Natural Language Modeling*. Ph.D. thesis, Johns Hopkins University.

Michael J. Collins. 1996. A new statistical parser based on bigram lexical dependencies. In *Proc. of the 34th Annual Meeting of the ACL*, pages 184–191.

Frederick Jelinek and Ciprian Chelba. 1999. Putting language into language modeling. In *Proc. of Eurospeech '99*, volume I, pages KN–1–6.

Frederik Jelinek and John Lafferty. 1991. Computation of the probability of initial substring generation by stochastic context-free grammars. *Computational Linguistics*, 17(3):315–323.

Frederick Jelinek. 1997. *Statistical Methods for Speech Recognition*. The MIT Press, Cambridge, MA.

Slava M. Katz. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Trans. on Acoustics, Speech and Signal Processing*, 35:400–401.

David M. Magerman. 1994. *Natural Language Parsing as Statistical Pattern Recognition*. Ph.D. thesis, Stanford University.

Christopher D. Manning and Bob Carpenter. 1997. Probabilistic parsing using left corner language models. In *Proc. of the Fifth International Workshop on Parsing Technologies*, pages 147–158.

Christopher D. Manning and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, MA.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1995. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Brian Roark and Mark Johnson. 2000. Efficient probabilistic top-down and left-corner parsing. In *Proc. of the 37th Annual Meeting of the ACL*, pages 421–428.

Andreas Stolcke. 1995. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201.

Filip Van Aelten and Marc Hogenhout. 2000. Inside-outside reestimation of Chelba-Jelinek models. Internal Report L&H–SR–00–027, Lernout & Hauspie, Wemmel, Belgium.

Dong Hoon Van Uytsel. 2000. Earley-inspired parsing language model: Background and preliminaries. Internal Report PSI-SPCH-00-1, K.U.Leuven, ESAT, Heverlee, Belgium.