MAXIMUM-LIKELIHOOD TRAINING OF THE PLCG-BASED LANGUAGE MODEL

Dong Hoon Van Uytsel, Dirk Van Compernolle and Patrick Wambacq

ESAT/PSI, Katholieke Universiteit Leuven, Belgium

ABSTRACT

In [1] a parsing language model based on a probabilistic left-corner grammar (PLCG) was proposed and encouraging performance on a speech recognition task using the PLCG-based language model was reported. In this paper we show how the PLCG-based language model can be further optimized by iterative parameter reestimation on unannotated training data. The precalculation of forward, inner and outer probabilities of states in the PLCG network provides an elegant crosscut to the computation of transition frequency expectations, which are needed in each iteration of the proposed reestimation procedure. The training algorithm enables model training on very large corpora. In our experiments, test set perplexity is close to saturation after three iterations, 5 to 16% lower than initially. We however observed no significant improvement of recognition accuracy after reestimation.

1. INTRODUCTION

In large-vocabulary continuous speech recognition (LVCSR) systems, the language model is responsible for estimating the probabilities of competing output sentence hypotheses, represented as sequences of word tokens. LVCSR language models are usually implemented as a combination of components such as word-based n-grams, class-based n-grams, trigger models, distance n-grams and sentence-mixture models. None of these, however, explicitly take advantage of the fact that sentences have a syntactic structure. This, and advances in large-scale natural language parsing, explain the recent interest in parsing language models (PLMs) for LVCSR.

For a given sentence, generative stochastic grammars generate a number of derivations and return the probability of each derivation, which is equivalent with an (analysis, sentence)-pair. PLMs compute the probability of a sentence as the sum over these joint probabilities. A fair number of recent papers deal with PLMs based on various grammar formalisms; for an overview, see [2].

For some classes of PLMs, but not for all, automatic procedures have been published that optimize the parameters on the *incomplete* data (plain text) as opposed to optimizing on the *complete* (hand-annotated) training data (e.g. a treebank) [3, 4, 5]. The cited procedures bias the models' behavior towards favoring the hidden syntactic structure that at the same time can be predicted reliably and explains the observed data best — regardless of the "correctness" of that hidden syntactic structure. All of them find a maximum-likelihood (ML) solution with the EM algorithm [6][4, 5] or an approximation thereof [3].

This paper deals with ML training of the PLCG-based LM using the EM algorithm. In [1], we introduced the PLCG-based model and reported encouraging results on speech recognition. The models used in these tests however were directly estimated from a large treebank and not additionally trained. In this paper we evaluate whether and how much improvement can be realized with maximum-likelihood training.

The rest of this paper is structured as follows. In the following section, we review the workings of our left-corner parsing language model. This is necessary to understand the EM training formulas developed in Sec. 3. Sec. 4 reports on small-scale experiments using models trained on the PennTreebank corpus and on larger-scale speech recognition experiments with models trained on a larger corpus. Sec. 5 concludes.

2. PROBABILISTIC LEFT-CORNER PARSING IN A STATE NETWORK

In this section we review concepts and definitions of the probabilistic left-corner parsing framework introduced in [1].

2.1. Simple left-corner parsing

Our probabilistic left-corner parser was inspired by the probabilistic reformulation in [7] of the deterministic left-corner parser [8]. Left-corner parsers work by a recursive procedure of bottom-up and top-down processing. From a goal category (the category of the constituent to be constructed), the first word is shifted, which is treated as any other complete constituent: if its category matches the goal category, then it can be attached, which completes the resolution of the goal category. Otherwise all possible local trees that have the category of the complete constituent as their leftmost daughter are projected. The other categories in each rule's righthand side serve as goal categories and are resolved by recursive application of this procedure.

In [7] a probabilistic formulation of the left-corner parser is given. The rule probabilities applied herein only depend on the category of the left corner (the first category of the righthand side) and its goal corner. The difference with a PCFG lies in the capturing of the dependency between a constituent category and its goal category and in the way local dependencies are parameterized: for a constituent A with goal G and children B and C, a PCFG uses a production probability $P(B, C|A)$, while the left-corner grammar contains a projection probability $P(A, C|B, G)$.

2.2. Context-sensitive, lexicalized left-corner parsing

We have extended Manning's framework with lexicalization and context-sensitivity [1]. A lexicalized grammar characterizes a constituent not only by its syntactical category, but also by a lexical head (a "headword"); all of the currently best large-scale parsers are lexicalized in that they assign one of the words occurring in a constituent as the headword of that constituent. We will follow this

This work was supported by the K.U.Leuven Research Fund. Correspondence e-mail: donghoon.vanuytsel@esat.kuleuven.ac.be

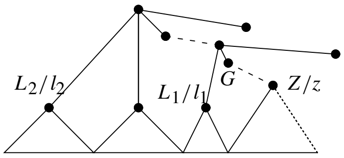

Fig. 1. Context of a state q, defined by Eq. 1. The dashed lines connect constituents with their goal categories.

practice. With context-sensitivity we mean that, besides dependency on the goal category, also dependency on a c-commanding constituent plays a role in the construction of a new constituent.

Since the resulting parser has to deal with a lot more dependencies, an efficient synchronous parsing algorithm that operates in a state space was proposed. The state space is represented as a *network* of states, and the parsing of a sentence is reformulated as optimal path search. The network is a directed acyclic graph containing N nodes, where each node is associated with a parser state q_i $(i = 1 \dots N)$ and each edge corresponds with an elementary parser move t_{ij} from state q_i to state q_j. There is only one edge between two states.

States — A *state* is a kind of enhanced dotted rule. We use the following notation:

$$q = (Z/z \rightarrow_i X/x \star_j \beta; G, L_1/l_1, L_2/l_2) \tag{1}$$

with the following meaning (cfr. Fig. 1): q corresponds with a set of constituents, each of which has the following properties: its category/head label is Z/z and its immediate left daughter constituent has category/head label X/x; it starts in position i and is completed till position j (the "current" position of q); it may have other resolved daughters, represented by a wildcard $\star$ since their identity is not important for the operation of the parser; there may be other unresolved daughters, represented by β; the goal category is G; the first-order left corner (LC) context, L_1/l_1, is the category/head of the left corner of the constituent's goal constituent; the second-order LC context, L_2/l_2 is the category/head pair of the immediate left corner of the goal constituent of Z's goal constituent. In linguistic terms, L_1 c-commands Z and L_2 c-commands Z's goal constituent.

If β is void, q is called a *complete* state, else it is called *incomplete*. If $Z = \mathsf{W}$, denoting that Z corresponds with a word, then q is called a *word* state. In that case Z has no daughters.

Transitions — There are three kinds of state transitions: shift, project and attach. In the constrained network, a **shift** transition is the only possible from an incomplete state q and it leads to a word state q':

$$q = (Z/z \rightarrow_i X/x \star_j Y\beta; G, L_1/l_1, L_2/l_2) \tag{2}$$

$$q' = (\mathsf{W}/w_j \rightarrow_j \star_{j+1}; Y, X/x, L_1/l_1) \tag{3}$$

In other words, in q the parser seeks to resolve a constituent Y starting in position j, so Y becomes the goal category of q'. Also note that Z's immediate left daughter X/x becomes the first-order LC context in q', while the first-order LC context of q becomes the second-order LC context of q'.

A **project** transition starts from a complete state q and leads to q', which is complete if a unary constituent is projected, but incomplete otherwise:

$$q = (Z/z \rightarrow_i X/x \star_j; G, L_1/l_1, L_2/l_2) \tag{4}$$

$$q' = (Z'/z' \rightarrow_i Z/z \star_j \beta'; G, L_1/l_1, L_2/l_2) \tag{5}$$

Table 1. PLCG submodel parameterizations used in this paper. The "from" column contains a reference to the corresponding equation. In the third column, the context elements are ordered by most signicant first. A is the event in which attachment takes place.

transition	from	parameterization
shift	(2)	$P_s(w_j \mid Y, x, l_1)$
tag	(4)	$P_t(Z' \mid z, G, L_1)$
project	(4)	$P_p(Z', \beta' \mid G, Z, X, z)$
attach	(6)	$P_a(\mathsf{A} \mid Y, D, y)$

Note that the constituents corresponding with q become the left daughter of the constituents corresponding with q' and that the start position, current position and the context are inherited. If Z is in head position in the local tree $Z' \rightarrow Z\beta'$, z' is set to z; otherwise it is set to "void".

If the category of a complete state q equals its goal category, then q corresponds with constituents that will complete an incomplete state q'' earlier in the path. The parser decides whether to **attach** (which completes q'' further) or to project further. The attach transition occurs between q and q' and is made possible by q'':

$$q = (Y/y \rightarrow_j D/d \star_k; Y, X/x, L_1/l_1) \tag{6}$$

$$q' = (Z/z \rightarrow_i X/x \star_k \beta; G, L_1/l_1, L_2/l_2) \tag{7}$$

$$q'' = (Z/z'' \rightarrow_i X/x \star_j Y\beta; G, L_1/l_1, L_2/l_2) \tag{8}$$

In plain words, Y is popped from the stack of unresolved daughter categories, and the current position is incremented from j to k. The lexical head z of q' remains void if z'' is void and Y is not in head position in q''. If z'' is not void or Y is not the head daughter of Z, then $z = z''$; otherwise z is void. For convenience, we introduce the notation $A(t(q, q')) = q''$ for the "point of attachment" associated with the attach transition from q to q'.

The probability of a transition is conditioned by and only by the state it starts from. In principle, any state feature may influence the transition probability; however, in practice, there is a trade-off between model detail and the limited reliability with which model parameters can be estimated due to data fragmentation, given a fixed amount of training data. Furthermore, statistical smoothing is necessary because straight maximum likelihood estimation would introduce too much bias. For the shift transition probability, it is necessary for the validity of the inner probability computations (see below) to condition only on features that appear in the arrival state too; i.e. X/x, j, Y, and L_1/l_1 in Eq. 2.

In our experiments and in the rest of this paper we assume parameterizations as given in Table 1. The semantics of the third column follows the equations referenced in the second column. Note that project transitions from a state that reached its goal category are computed as $P_t(\cdot|\cdot)(1 - P_a(\mathsf{A}|\cdot))$. Also note that we mention a "tag" transition; this is in fact a project transition from a word state. It turned out to be favorable to provide a different parameterization in this case. The order in which the context elements appear is important because less significant context elements are stepwise left out from the lower-order smoothing models. The parameterizations were found by some hand tweaking on a development test set held out from the training data, and they were kept fixed for the rest of the experiments afterwards. In a certain sense

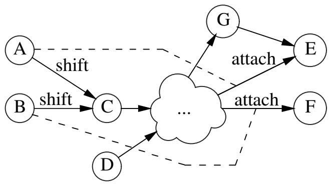

Fig. 2. Non-locality in the constrained network: $A(t)$ must occur in the path taking transition t (connected with $A(t)$ with dashed lines). The sequence B→C→ ... →E is an invalid path, B→C→ ... →F and B→C→ ... →G→E are valid paths.

they are arbitrary and possibly corpus dependent; perhaps the application of automatic discovery techniques could be interesting in this case.

Paths — A *path* $\mathbf{q_k} = (q_{k_1}, q_{k_2}, \ldots, q_{k_m})$ is a sequence of connected states and is characterized by its state index sequence $\mathbf{k}$. The probability of $\mathbf{q_k}$ is given by $P(\mathbf{q_k}) = \prod_{j=1}^{m-1} P(t_{k_j k_{j+1}}|q_{k_j})$ as transitions are assumed mutually independent (actually the dependence of a transition on the preceding transitions is reduced to the source state, which functions as its sole conditioning context). If $\mathbf{q_k}$ starts with the initial state ($k_1 = I$) and ends in the final state $k_m = F$, then it is said to be complete and uniquely corresponds with the derivation (S, T), where T is a parse tree and S is the input sentence. The output of the PLCG-based LM, $P(S)$, is obtained as the S-marginal of $P(S, T)$. Although enumerating all T is, in general, hard or impossible, it is possible to factor the sum over all T using state probabilities as is shown below. Given a sentence S, we will only visit the subset of the network that contains the paths that generate S and no other paths. This subset is called the (S-)*constrained* network.

An essential property of the network is that there are no two states with identical features. This allows a compact representation of the constrained network and the application of dynamic programming such that reduplicating parsing effort is avoided wherever possible. In this sense, the constrained network is a compressed version of all the paths that generate S. Note, however, that in general not all sequences of connected states in the network constitute a valid path. The reason is that an attach transition to a state that completes another state that did not appear earlier in the path, is disallowed. For instance in Fig. 2, the sequence A→C→ ... →F does not constitute a valid path. This non-locality slightly complicates the computation of state probabilities: forward-backward training as in [5] does not fit our purpose — in the next section we show how it can be replaced with a variant of inside-outside training.

3. TRAINING THE PLCG-BASED LM

3.1. State probabilities and expected transition frequency

The definition of state probabilities allows an efficient computation of word prefix probabilities and expected transition frequencies. Word prefix probabilities enable the language model to emit probabilities of the format $P(w_j|w_0 \ldots w_{j-1})$ (in addition to the probability of the whole sentence), which is a most useful property in current speech recognition system architectures. Expected transition frequencies are needed in the E-step of the EM training procedure developed below.

With each state q, defined as (1), we associate three probabilities. The *forward* probability $\mu(q)$ is the sum of probabilities of the paths from the initial state till q. The *inner* probability $\nu(q)$ is the sum of probabilities of the paths that start with a shift of w_i and arrive in q. The *outer* probability $\xi(q)$ is a sum, in which each term is the partial probability of a path from q_I till q_F containing q, but with the probability of the path fragment covered by $\nu(q)$ left out: the fragment that starts with a shift of w_i and arrives in q. A direct consequence is that $P(S) = \mu(q_F) = \nu(q_F) = \xi(q_I)$.

The computation of $\mu(q)$ and $\nu(q)$ starts from the initialization $\mu(q_I) = \nu(q_I) = \xi(q_F) = 1$ and proceeds in a forward direction, synchronously with the parsing algorithm [1]. After the forward step, $\xi(q_F)$ is set to 1 and the other $\xi(q)$ are computed in reverse topological order. The following recursive relations hold:

$$\mu(q_i) = \sum_{t_{ki}} \begin{cases} \nu(q_k)P(t_{ki}|q_k)\mu(A(t_{ki})) & \text{if } t_{ki} \text{ is attach} \\ \mu(q_k)P(t_{ki}|q_k) & \text{otherwise} \end{cases} \tag{9}$$

$$\nu(q_i) = \begin{cases} P(t_{ki}|q_k) \text{ if } q_i \text{ is word state} \\ \sum_{t_{ki}} \begin{cases} \nu(q_k)P(t_{ki}|q_k)\nu(A(t_{ki})) & \text{if } t_{ki} \text{ is attach} \\ \nu(q_k)P(t_{ki}|q_k) & \text{otherwise} \end{cases} \end{cases} \tag{10}$$

$$\xi(q_i) = \begin{cases} \sum_{t_{mn}:A(t_{mn})=q_i} \xi(q_n)\nu(q_m)P(t_{mn}|q_k) \text{ if } q_i \text{ is incomplete} \\ \sum_{t_{ik}} \begin{cases} \xi(q_k)P(t_{ik}|q_i)\nu(q_i) & \text{if } t_{ik} \text{ is attach} \\ \xi(q_k)P(t_{ik}|q_i) & \text{otherwise} \end{cases} \end{cases} \tag{11}$$

Due to space constraints, we do not give a formal proof of these equations here; though they can be understood by reasoning on the network, keeping its specific nested structure in mind (Fig. 2).

Given an input sentence S, the expected frequency of a transition t is the sum of *visit* probabilities of the transitions $t_{ij} = t$ in the S-constrained network: $\mathrm{EF}_S(t) = \sum_{t_{ij}=t} P(t_{ij}|S)$. $P(t_{ij}|S)$ can be expressed as a sum of the probabilities of the complete paths containing t_{ij} given S and efficiently computed using state probabilities:

$$P(t_{ij})|S) = \begin{cases} \nu(q_j)\xi(q_j)/\nu(q_F) & \text{if } t_{ij} \text{ is shift} \\ \nu(q_i)P(t_{ij}|q_i)\xi(q_j)/\nu(q_F) & \text{otherwise} \end{cases} \tag{12}$$

3.2. EM updates

Let Ω be the unannotated training corpus and Ω^* the same data plus the hidden structure generated with a model θ, and suppose we want to update an old model θ^o. The E-step of the EM algorithm involves the construction of an auxiliary function $Q(\theta, \theta^o)$, the θ^o-expectation of the logarithm of the θ-likelihood of Ω^* given Ω. In our case it can be written as

$$\begin{aligned} Q(\theta, \theta^o) &= \sum_{S\in\Omega} \sum_{T} P_{\theta^o}(T|S) \log P_\theta(S, T) \\ &= \sum_{t} \mathrm{EF}_\Omega(t|\theta^o) \log p_\theta(t) \end{aligned} \tag{13}$$

where $\mathrm{EF}_\Omega(t|\theta^o) = \sum_{S\in\Omega} \mathrm{EF}_S(t|\theta^o)$ and t ranges over all possible (transition|context) events. If we choose

$$\hat{\theta} = \arg\max_{\theta} Q(\theta, \theta^o) \tag{14}$$

Table 2. Training times, word error rates and train and test set perplexities before and after reestimation. Training time is expressed in hours of user time of a 650 Mhz PentiumIII processor.

	PTB models		BLLIP models			
	train PPL	test PPL	train time(h)	train PPL	test PPL	WER (%)
3gram		126.0			82.0	7.98
0 it.	83.6	126.0	430	82.5	95.4	7.60
1 it.	66.9	120.4	400	64.2	86.6	7.62
2 it.	64.5	120.2	355	60.2	82.5	7.50
3 it.		119.4			80.4	7.49

then [6] proves that $P_{\hat{\theta}}(\Omega) \geq P_{\theta^o}(\Omega)$, such that the EM algorithm is guaranteed to converge. The conditional probabilities $p_\theta(t)$ are subject to $\sum_{t:H(t)=h} p_\theta(t) = 1$ for each transition context h, where $H(t)$ denotes the conditioning context of t. Under these constraints the maximum is found in

$$p_{\hat{\theta}}(t) = \frac{\mathrm{EF}_\Omega(t|\theta^o)}{\sum\limits_{t_k:H(t_k)=H(t)} \mathrm{EF}_\Omega(t_k|\theta^o)} \qquad (15)$$

4. EXPERIMENTS

We conducted experiments on the PennTreebank (PTB) [9] and the machine-parsed WSJ corpus available from BLLIP through the LDC [10] (BLLIP). BLLIP is about 35 times larger than the PTB, but contains errors. For the PTB experiment set, we initialized models on sections 0–20 and reestimated them on the same data. The models for the BLLIP experiment set were initialized and reestimated on the BLLIP corpus plus sections 0–20 of the PTB corpus; care was taken that the speech recognition test set was excluded from the training data. All the submodels and the word-trigram baseline were pruned by omitting all maximum-order events that had appeared less than twice in the training data, in order to relax memory requirements and counteract overtraining. The submodels were smoothed using Katz back-off [11], which in our experience is very comparable to deleted-interpolation submodels when cooperating in a PLCG-based language model. We preferred the first smoothing method over the latter because it allows a faster evaluation.

Test set perplexities for BLLIP and PTB models were measured on sections 23–24 of the PTB corpus before reestimation and after each iteration. In Table 2, it is shown that test set perplexities are reduced by 5% and 16%, resp. for PTB and BLLIP. Train set perplexity drops by 23% (PTB) and by 27% (BLLIP) after two iterations. It is also worth mentioning that the reestimated models evaluate and retrain faster with fixed pruning settings, as can be read off the “time(h)” column.

In order to evaluate the impact of reestimating the PLCG-based LM on speech recognition performance, we used the BLLIP models for rescoring 100-best lists that were obtained from the first pass of a mainstream HMM-based Viterbi decoder equipped with a word trigram LM. We tested on the evaluation and development test set (20k open vocabulary with verbalized punctuation) of the DARPA WSJ Nov.’92 LVCSR test suite. The 100-best lists were first preprocessed to match the tokenization of the BLLIP models: e.g. “don’t” was replaced with “do n’t” and numbers were replaced with “N”. The results listed in Table 2 are obtained by rescoring the n-best lists with a word-trigram model (as a baseline) and with word-trigram interpolated initial and reestimated versions of the BLLIP models. The WER does not convincingly follow the downward trend of the test set perplexity.

5. CONCLUSION

We proposed an efficient iterative training algorithm for the maximum-likelihood optimization of the PLCG-based language model. It can be used to reestimate the language model on the initialization corpus and to train it on other very large unannotated corpora. We evaluated reestimation of small models (based on the Penn Treebank) and larger models (based on the BLLIP machine-parsed WSJ corpus). Test set perplexities of the reestimated models are reduced by approximately 5% to 16% compared with those obtained with the initial model; reestimated models also tend to evaluate faster with equal pruning parameter settings. The speech recognition accuracy measured with the reestimated models however remains at a status quo. Evaluation of training on additional unannoted corpora is a topic for future research.

6. REFERENCES

[1] D. H. Van Uytsel, F. Van Aelten, and D. Van Compernolle, “A structured language model based on context-sensitive probabilistic left-corner parsing,” in *Proc. NAACL-2001*, (Pittsburgh, PA, USA), pp. 223–230, June 2001.

[2] D. H. Van Uytsel, “Earley-inspired parsing language model: Background and preliminaries,” Internal Report PSI-SPCH-00-1, K.U.Leuven, ESAT, Heverlee, Belgium, May 2000.

[3] C. Chelba, *Exploiting Syntactic Structure for Natural Language Modeling*. PhD thesis, Johns Hopkins University, 2000.

[4] F. Van Aelten and M. Hogenhout, “Inside-outside reestimation of Chelba-Jelinek models,” Technical Report L&H–SR–00–027, Lernout & Hauspie, Wemmel, Belgium, 2000.

[5] R. Bod, “Combining semantic and syntactic structure for language modeling,” in *Proc. ICSLP-2000*, vol. III, (Beijing, China), pp. 106–109, 2000.

[6] A. Dempster, N. Laird, and D. Rubin, “Maximum likelihood from incomplete data via the EM algorithm,” *J. Royal Statistical Society, Series B*, vol. 39, pp. 1–38, 1977.

[7] C. D. Manning and B. Carpenter, “Probabilistic parsing using left corner language models,” in *Proc. IWPT-1997*, pp. 147–158, 1997.

[8] D. J. Rosenkrantz and P. M. Lewis II, “Deterministic left corner parsing,” in *IEEE Conference Record of the 11th Annual Symposium on Switching and Automata*, pp. 139–152, 1970.

[9] M. P. Marcus *et al.*, “Building a large annotated corpus of English: the Penn Treebank,” *Computational Linguistics*, vol. 19, no. 2, pp. 313–330, 1995.

[10] E. Charniak, “A maximum-entropy inspired parser,” in *Proc. NAACL-2000*, pp. 132–139, 2000.

[11] S. M. Katz, “Estimation of probabilities from sparse data for the language model component of a speech recognizer,” *IEEE Trans. ASSP*, vol. 35, pp. 400–401, 1987.