cs.CL/9811022 12 Nov 1998

Exploiting Syntactic Structure for Language Modeling

Ciprian Chelba and **Frederick Jelinek**
Center for Language and Speech Processing
The Johns Hopkins University, Barton Hall 320
3400 N. Charles St., Baltimore, MD-21218, USA
{chelba,jelinek}@jhu.edu

Abstract

The paper presents a language model that develops syntactic structure and uses it to extract meaningful information from the word history, thus enabling the use of long distance dependencies. The model assigns probability to every joint sequence of words–binary-parse-structure with headword annotation and operates in a left-to-right manner — therefore usable for automatic speech recognition. The model, its probabilistic parameterization, and a set of experiments meant to evaluate its predictive power are presented; an improvement over standard trigram modeling is achieved.

1 Introduction

The main goal of the present work is to develop a language model that uses syntactic structure to model long-distance dependencies. During the summer96 DoD Workshop a similar attempt was made by the dependency modeling group. The model we present is closely related to the one investigated in (Chelba et al., 1997), however different in a few important aspects:

• our model operates in a left-to-right manner, allowing the decoding of word lattices, as opposed to the one referred to previously, where only whole sentences could be processed, thus reducing its applicability to n-best list re-scoring; the syntactic structure is developed as a model component;

• our model is a factored version of the one in (Chelba et al., 1997), thus enabling the calculation of the joint probability of words and parse structure; this was not possible in the previous case due to the huge computational complexity of the model.

Our model develops syntactic structure incrementally while traversing the sentence from left to right. This is the main difference between our approach and other approaches to statistical natural language parsing. Our parsing strategy is similar to the incremental syntax ones proposed relatively recently in the linguistic community (Philips, 1996). The probabilistic model, its parameterization and a few experiments that are meant to evaluate its potential for speech recognition are presented.

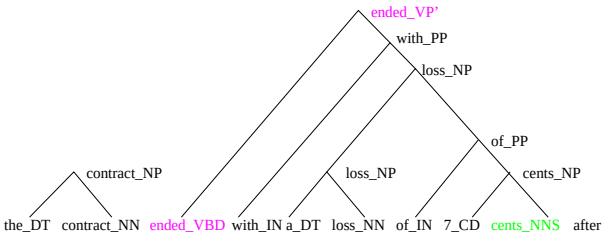

Figure 1: Partial parse

2 The Basic Idea and Terminology

Consider predicting the word `after` in the sentence: `the contract ended with a loss of 7 cents after trading as low as 89 cents`.
A 3-gram approach would predict `after` from `(7, cents)` whereas it is intuitively clear that the strongest predictor would be `ended` which is outside the reach of even 7-grams. Our assumption is that what enables humans to make a good prediction of `after` is the syntactic structure in the past. The linguistically correct *partial parse* of the word history when predicting `after` is shown in Figure 1. The word `ended` is called the *headword* of the *constituent* `(ended (with (...)))` and `ended` is an *exposed headword* when predicting `after` — topmost headword in the largest constituent that contains it. The syntactic structure in the past filters out irrelevant words and points to the important ones, thus enabling the use of long distance information when predicting the next word.

Our model will attempt to build the syntactic structure incrementally while traversing the sentence left-to-right. The model will assign a probability $P(W, T)$ to every sentence W with every possible POStag assignment, binary branching parse, nonterminal label and headword annotation for every constituent of T.

Let W be a sentence of length n words to which we have prepended `<s>` and appended `</s>` so that w_0 =`<s>` and w_{n+1} =`</s>`. Let W_k be the word k-prefix $w_0 \ldots w_k$ of the sentence and $W_k T_k$

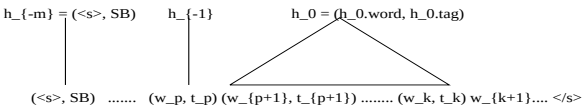

Figure 2: A word-parse k-prefix

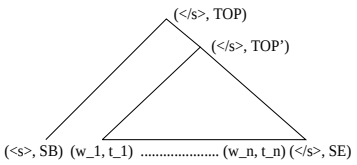

Figure 3: Complete parse

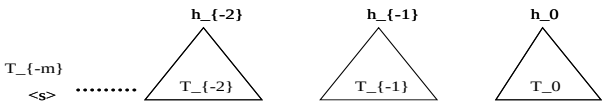

Figure 4: Before an adjoin operation

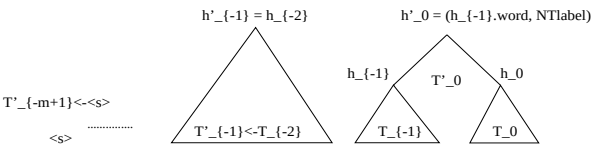

Figure 5: Result of adjoin-left under NTlabel

the *word-parse k-prefix*. To stress this point, a word-parse k-prefix contains — for a given parse — only those binary subtrees whose span is completely included in the word k-prefix, excluding w_0 =<s>. Single words along with their POStag can be regarded as root-only trees. Figure 2 shows a word-parse k-prefix; `h_0 .. h_{-m}` are the *exposed heads*, each head being a pair(headword, non-terminal label), or (word, POStag) in the case of a root-only tree. A *complete parse* — Figure 3 — is any binary parse of the
$(w_1, t_1) \ldots (w_n, t_n)$ `(</s>, SE)` sequence with the restriction that `(</s>, TOP')` is the only allowed head. Note that $((w_1, t_1) \ldots (w_n, t_n))$ *needn't* be a constituent, but for the parses where it is, there is no restriction on which of its words is the headword or what is the non-terminal label that accompanies the headword.

The model will operate by means of three modules:

- WORD-PREDICTOR predicts the next word w_{k+1} given the word-parse k-prefix and then passes control to the TAGGER;
- TAGGER predicts the POStag of the next word t_{k+1} given the word-parse k-prefix and the newly predicted word and then passes control to the PARSER;
- PARSER grows the already existing binary branching structure by repeatedly generating the transitions:

`(unary, NTlabel)`, `(adjoin-left, NTlabel)` or `(adjoin-right, NTlabel)` until it passes control to the PREDICTOR by taking a `null` transition. `NTlabel` is the non-terminal label assigned to the newly built constituent and `{left,right}` specifies where the new headword is inherited from.

The operations performed by the PARSER are illustrated in Figures 4-6 and they ensure that all possible binary branching parses with all possible headword and non-terminal label assignments for the $w_1 \ldots w_k$ word sequence can be generated. The following algorithm formalizes the above description of the sequential generation of a sentence with a complete parse.

```
Transition t;          // a PARSER transition
predict (<s>, SB);
do{
  //WORD-PREDICTOR and TAGGER
  predict (next_word, POStag);
  //PARSER
  do{
    if(h_{-1}.word != <s>){
      if(h_0.word == </s>)
        t = (adjoin-right, TOP');
      else{
        if(h_0.tag == NTlabel)
          t = [(adjoin-{left,right}, NTlabel),
               null];
        else
          t = [(unary, NTlabel),
               (adjoin-{left,right}, NTlabel),
               null];
      }
    }
    else{
      if(h_0.tag == NTlabel)
        t = null;
      else
        t = [(unary, NTlabel), null];
    }
  }while(t != null) //done PARSER
}while(!(h_0.word==</s> && h_{-1}.word==<s>))
t = (adjoin-right, TOP); //adjoin <s>_SB; DONE;
```

The unary transition is allowed only when the most recent exposed head is a leaf of the tree — a regular word along with its POStag — hence it can be taken at most once at a given position in the

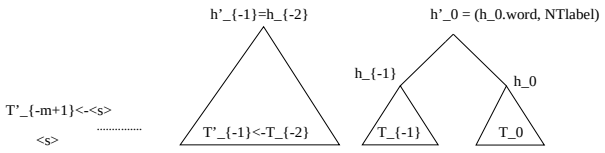

Figure 6: Result of adjoin-right under NTlabel

input word string. The second subtree in Figure 2 provides an example of a unary transition followed by a null transition.

It is easy to see that any given word sequence with a possible parse and headword annotation is generated by a unique sequence of model actions. This will prove very useful in initializing our model parameters from a treebank — see section 3.5.

3 Probabilistic Model

The probability $P(W,T)$ of a word sequence W and a complete parse T can be broken into:

$$P(W,T) = \prod_{k=1}^{n+1}[\ P(w_k/W_{k-1}T_{k-1}) \cdot P(t_k/W_{k-1}T_{k-1}, w_k) \cdot \prod_{i=1}^{N_k} P(p_i^k/W_{k-1}T_{k-1}, w_k, t_k, p_1^k \ldots p_{i-1}^k)] \quad (1)$$

where:
• $W_{k-1}T_{k-1}$ is the word-parse $(k-1)$-prefix
• w_k is the word predicted by WORD-PREDICTOR
• t_k is the tag assigned to w_k by the TAGGER
• $N_k - 1$ is the number of operations the PARSER executes before passing control to the WORD-PREDICTOR (the N_k-th operation at position k is the `null` transition); N_k is a function of T
• p_i^k denotes the i-th PARSER operation carried out at position k in the word string;
$p_1^k \in \{$`(unary, NTlabel)`, `(adjoin-left, NTlabel)`, `(adjoin-right, NTlabel), null`$\}$,
$p_i^k \in \{$ `(adjoin-left, NTlabel)`, `(adjoin-right, NTlabel)`$\}, 1 < i < N_k$,
$p_i^k =$`null`$, i = N_k$

Our model is based on three probabilities:

$$P(w_k/W_{k-1}T_{k-1}) \quad (2)$$

$$P(t_k/w_k, W_{k-1}T_{k-1}) \quad (3)$$

$$P(p_i^k/w_k, t_k, W_{k-1}T_{k-1}, p_1^k \ldots p_{i-1}^k) \quad (4)$$

As can be seen, $(w_k, t_k, W_{k-1}T_{k-1}, p_1^k \ldots p_{i-1}^k)$ is one of the N_k word-parse k-prefixes W_kT_k at position k in the sentence, $i = \overline{1, N_k}$.

To ensure a proper probabilistic model (1) we have to make sure that (2), (3) and (4) are well defined conditional probabilities and that the model halts with probability one. Consequently, certain PARSER and WORD-PREDICTOR probabilities must be given specific values:
• $P($`null`$/W_kT_k) = 1$, if `h_{-1}.word = <s>` and `h_{0}` $\neq$ `(</s>, TOP')` — that is, before predicting `</s>` — ensures that `(<s>, SB)` is adjoined in the last step of the parsing process;
• $P($`(adjoin-right, TOP)`$/W_kT_k) = 1$, if `h_0 = (</s>, TOP')` and `h_{-1}.word = <s>` and
$P($`(adjoin-right, TOP')`$/W_kT_k) = 1$, if `h_0 = (</s>, TOP')` and `h_{-1}.word` $\neq$ `<s>`
ensure that the parse generated by our model is consistent with the definition of a complete parse;
• $P($`(unary, NTlabel)`$/W_kT_k) = 0$, if `h_0.tag` $\neq$ `POStag` ensures correct treatment of unary productions;
• $\exists \epsilon > 0, \forall W_{k-1}T_{k-1}, P(w_k$=`</s>`$/W_{k-1}T_{k-1}) \geq \epsilon$ ensures that the model halts with probability one.

The word-predictor model (2) predicts the next word based on the preceding 2 *exposed heads*, thus making the following equivalence classification:

$$P(w_k/W_{k-1}T_{k-1}) = P(w_k/h_0, h_{-1})$$

After experimenting with several equivalence classifications of the word-parse prefix for the tagger model, the conditioning part of model (3) was reduced to using the word to be tagged and the tags of the two most recent exposed heads:

$$P(t_k/w_k, W_{k-1}T_{k-1}) = P(t_k/w_k, h_0.tag, h_{-1}.tag)$$

Model (4) assigns probability to different parses of the word k-prefix by chaining the elementary operations described above. The workings of the parser module are similar to those of Spatter (Jelinek et al., 1994). The equivalence classification of the W_kT_k word-parse we used for the parser model (4) was the same as the one used in (Collins, 1996):

$$P(p_i^k/W_kT_k) = P(p_i^k/h_0, h_{-1})$$

It is worth noting that if the binary branching structure developed by the parser were always right-branching and we mapped the POStag and non-terminal label vocabularies to a single type then our model would be equivalent to a trigram language model.

3.1 Modeling Tools

All model components — WORD-PREDICTOR, TAGGER, PARSER — are conditional probabilistic models of the type $P(y/x_1, x_2, \ldots, x_n)$ where $y, x_1, x_2, \ldots, x_n$ belong to a mixed bag of words, POStags, non-terminal labels and parser operations (y only). For simplicity, the modeling method we chose was deleted interpolation among relative frequency estimates of different orders $f_n(\cdot)$ using a

recursive mixing scheme:

$$P(y/x_1,\ldots,x_n) = \lambda(x_1,\ldots,x_n)\cdot P(y/x_1,\ldots,x_{n-1}) + (1-\lambda(x_1,\ldots,x_n))\cdot f_n(y/x_1,\ldots,x_n), \quad (5)$$

$$f_{-1}(y) = uniform(vocabulary(y)) \quad (6)$$

As can be seen, the context mixing scheme discards items in the context in right-to-left order. The λ coefficients are tied based on the range of the count $C(x_1,\ldots,x_n)$. The approach is a standard one which doesn't require an extensive description given the literature available on it (Jelinek and Mercer, 1980).

3.2 Search Strategy

Since the number of parses for a given word prefix W_k grows exponentially with k, $|\{T_k\}| \sim O(2^k)$, the state space of our model is huge even for relatively short sentences so we had to use a search strategy that prunes it. Our choice was a synchronous multi-stack search algorithm which is very similar to a beam search.

Each stack contains hypotheses — partial parses — that have been constructed by *the same number of predictor and the same number of parser operations*. The hypotheses in each stack are ranked according to the $\ln(P(W,T))$ score, highest on top. The width of the search is controlled by two parameters:

• the maximum stack depth — the maximum number of hypotheses the stack can contain at any given state;

• log-probability threshold — the difference between the log-probability score of the top-most hypothesis and the bottom-most hypothesis at any given state of the stack cannot be larger than a given threshold.

Figure 7 shows schematically the operations associated with the scanning of a new word w_{k+1}. The above pruning strategy proved to be insufficient so we chose to also discard all hypotheses whose score is more than the log-probability threshold below the score of the topmost hypothesis. This additional pruning step is performed after all hypotheses in stage k' have been extended with the `null` parser transition and thus prepared for scanning a new word.

3.3 Word Level Perplexity

The conditional perplexity calculated by assigning to a whole sentence the probability:

$$P(W/T^*) = \prod_{k=0}^{n} P(w_{k+1}/W_k T_k^*), \quad (7)$$

where $T^* = argmax_T P(W,T)$, is not valid because it is not causal: when predicting w_{k+1} we use T^* which was determined by looking at the entire sentence. To be able to compare the perplexity of our

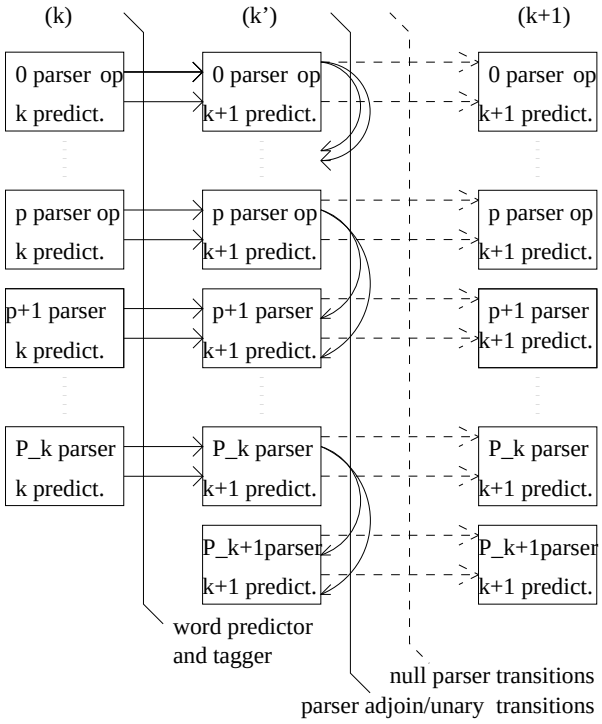

Figure 7: One search extension cycle

model with that resulting from the standard trigram approach, we need to factor in the entropy of guessing the correct parse T_k^* *before predicting* w_{k+1}, based solely on the word prefix W_k.

The probability assignment for the word at position $k+1$ in the input sentence is made using:

$$P(w_{k+1}/W_k) = \sum_{T_k\in S_k} P(w_{k+1}/W_k T_k)\cdot\rho(W_k,T_k), \quad (8)$$

$$\rho(W_k,T_k) = P(W_k T_k)/\sum_{T_k\in S_k} P(W_k T_k) \quad (9)$$

which ensures a proper probability over strings W^*, where S_k is the set of all parses present in our stacks at the current stage k.

Another possibility for evaluating the word level perplexity of our model is to approximate the probability of a whole sentence:

$$P(W) = \sum_{k=1}^{N} P(W,T^{(k)}) \quad (10)$$

where $T^{(k)}$ is one of the "N-best" — in the sense defined by our search — parses for W. This is a deficient probability assignment, however useful for justifying the model parameter re-estimation.

The two estimates (8) and (10) are both consistent in the sense that if the sums are carried over all

possible parses we get the correct value for the word level perplexity of our model.

3.4 Parameter Re-estimation

The major problem we face when trying to reestimate the model parameters is the huge state space of the model and the fact that dynamic programming techniques similar to those used in HMM parameter re-estimation cannot be used with our model. Our solution is inspired by an HMM re-estimation technique that works on pruned — N-best — trellises(Byrne et al., 1998).

Let $(W, T^{(k)}), k = 1 \ldots N$ be the set of hypotheses that survived our pruning strategy until the end of the parsing process for sentence W. Each of them was produced by a sequence of model actions, chained together as described in section 2; let us call the sequence of model actions that produced a given (W, T) the $derivation(W, T)$.

Let *an elementary event* in the $derivation(W, T)$ be $(y_l^{(m_l)}, \underline{x}_l^{(m_l)})$ where:

- l is the index of the current model action;
- m_l is the model component — WORD-PREDICTOR, TAGGER, PARSER — that takes action number l in the $derivation(W, T)$;
- $y_l^{(m_l)}$ is the action taken at position l in the derivation:

if m_l = WORD-PREDICTOR, then $y_l^{(m_l)}$ is a word;
if m_l = TAGGER, then $y_l^{(m_l)}$ is a POStag;
if m_l = PARSER, then $y_l^{(m_l)}$ is a parser-action;

- $\underline{x}_l^{(m_l)}$ is the context in which the above action was taken:

if m_l = WORD-PREDICTOR or PARSER, then $\underline{x}_l^{(m_l)} = (h_0.tag, h_0.word, h_{-1}.tag, h_{-1}.word)$;
if m_l = TAGGER, then
$\underline{x}_l^{(m_l)}$ = (word-to-tag, $h_0.tag, h_{-1}.tag$).

The probability associated with each model action is determined as described in section 3.1, based on counts $C^{(m)}(y^{(m)}, \underline{x}^{(m)})$, one set for each model component.

Assuming that the deleted interpolation coefficients and the count ranges used for tying them stay fixed, these counts are the only parameters to be re-estimated in an eventual re-estimation procedure; indeed, once a set of counts $C^{(m)}(y^{(m)}, \underline{x}^{(m)})$ is specified for a given model m, we can easily calculate:

- the relative frequency estimates $f_n^{(m)}(y^{(m)}/\underline{x}_n^{(m)})$ for all context orders $n = 0 \ldots$maximum-order$(model(m))$;
- the count $C^{(m)}(\underline{x}_n^{(m)})$ used for determining the $\lambda(\underline{x}_n^{(m)})$ value to be used with the order-n context $\underline{x}_n^{(m)}$.

This is all we need for calculating the probability of an *elementary event* and then the probability of an entire derivation.

One training iteration of the re-estimation procedure we propose is described by the following algorithm:

```
N-best parse development data; // counts.Ei
// prepare counts.E(i+1)
for each model component c{
  gather_counts development model_c;
}
```

In the parsing stage we retain for each "N-best" hypothesis $(W, T^{(k)}), k = 1 \ldots N$, only the quantity $\phi(W, T^{(k)}) = P(W, T^{(k)}) / \sum_{k=1}^{N} P(W, T^{(k)})$ and its $derivation(W, T^{(k)})$. We then scan all the derivations in the "development set" and, for each occurrence of the elementary event $(y^{(m)}, \underline{x}^{(m)})$ in $derivation(W, T^{(k)})$ we accumulate the value $\phi(W, T^{(k)})$ in the $C^{(m)}(y^{(m)}, \underline{x}^{(m)})$ counter to be used in the next iteration.

The intuition behind this procedure is that $\phi(W, T^{(k)})$ is an approximation to the $P(T^{(k)}/W)$ probability which places all its mass on the parses that survived the parsing process; the above procedure simply accumulates the expected values of the counts $C^{(m)}(y^{(m)}, \underline{x}^{(m)})$ under the $\phi(W, T^{(k)})$ conditional distribution. As explained previously, the $C^{(m)}(y^{(m)}, \underline{x}^{(m)})$ counts are the parameters defining our model, making our procedure similar to a rigorous EM approach (Dempster et al., 1977).

A particular — and very interesting — case is that of events which had count zero but get a non-zero count in the next iteration, caused by the "N-best" nature of the re-estimation process. Consider a given sentence in our "development" set. The "N-best" derivations for this sentence are trajectories through the state space of our model. They will change from one iteration to the other due to the smoothing involved in the probability estimation and the change of the parameters — event counts — defining our model, thus allowing new events to appear and discarding others through purging low probability events from the stacks. The higher the number of trajectories per sentence, the more dynamic this change is expected to be.

The results we obtained are presented in the experiments section. All the perplexity evaluations were done using the left-to-right formula (8) (L2R-PPL) for which the perplexity on the "development set" is not guaranteed to decrease from one iteration to another. However, we believe that our re-estimation method should not increase the approximation to perplexity based on (10) (SUM-PPL) — again, on the "development set"; we rely on the consistency property outlined at the end of section 3.3 to correlate the desired decrease in L2R-PPL with that in SUM-PPL. No claim can be made about the change in either L2R-PPL or SUM-PPL on test data.

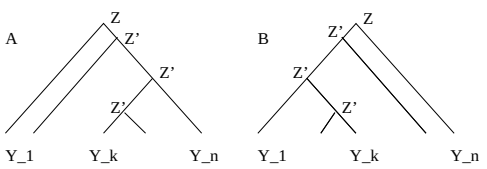

Figure 8: Binarization schemes

3.5 Initial Parameters

Each model component — WORD-PREDICTOR, TAGGER, PARSER — is trained initially from a set of parsed sentences, after each parse tree (W, T) undergoes:
• headword percolation and binarization — see section 4;
• decomposition into its $derivation(W, T)$.
Then, separately for each m model component, we:
• gather joint counts $C^{(m)}(y^{(m)}, \underline{x}^{(m)})$ from the derivations that make up the "development data" using $\phi(W, T) = 1$;
• estimate the deleted interpolation coefficients on joint counts gathered from "check data" using the EM algorithm.
These are the initial parameters used with the re-estimation procedure described in the previous section.

4 Headword Percolation and Binarization

In order to get initial statistics for our model components we needed to binarize the UPenn Treebank (Marcus et al., 1995) parse trees and percolate headwords. The procedure we used was to first percolate headwords using a context-free (CF) rule-based approach and then binarize the parses by using a rule-based approach again.

The headword of a phrase is the word that best represents the phrase, all the other words in the phrase being modifiers of the headword. Statistically speaking, we were satisfied with the output of an enhanced version of the procedure described in (Collins, 1996) — also known under the name "Magerman & Black Headword Percolation Rules".

Once the position of the headword within a constituent — equivalent with a CF production of the type $Z \rightarrow Y_1 \ldots Y_n$, where $Z, Y_1, \ldots Y_n$ are non-terminal labels or POStags (only for Y_i) — is identified to be k, we binarize the constituent as follows: depending on the Z identity, a fixed rule is used to decide which of the two binarization schemes in Figure 8 to apply. The intermediate nodes created by the above binarization schemes receive the non-terminal label Z'.

5 Experiments

Due to the low speed of the parser — 200 wds/min for stack depth 10 and log-probability threshold 6.91 nats (1/1000) — we could carry out the re-estimation technique described in section 3.4 on only 1 Mwds of training data. For convenience we chose to work on the UPenn Treebank corpus. The vocabulary sizes were:
• word vocabulary: 10k, open — all words outside the vocabulary are mapped to the `<unk>` token;
• POS tag vocabulary: 40, closed;
• non-terminal tag vocabulary: 52, closed;
• parser operation vocabulary: 107, closed;
The training data was split into "development" set — 929,564wds (sections 00-20) — and "check set" — 73,760wds (sections 21-22); the test set size was 82,430wds (sections 23-24). The "check" set has been used for estimating the interpolation weights and tuning the search parameters; the "development" set has been used for gathering/estimating counts; the test set has been used strictly for evaluating model performance.

Table 1 shows the results of the re-estimation technique presented in section 3.4. We achieved a reduction in test-data perplexity bringing an improvement over a deleted interpolation trigram model whose perplexity was 167.14 on the same training-test data; the reduction is statistically significant according to a sign test.

iteration number	DEV set L2R-PPL	TEST set L2R-PPL
E0	24.70	167.47
E1	22.34	160.76
E2	21.69	158.97
E3	21.26	158.28
3-gram	21.20	167.14

Table 1: Parameter re-estimation results

Simple linear interpolation between our model and the trigram model:

$$Q(w_{k+1}/W_k) = \lambda \cdot P(w_{k+1}/w_{k-1}, w_k) + (1 - \lambda) \cdot P(w_{k+1}/W_k)$$

yielded a further improvement in PPL, as shown in Table 2. The interpolation weight was estimated on check data to be $\lambda = 0.36$.

An overall relative reduction of 11% over the trigram model has been achieved.

6 Conclusions and Future Directions

The large difference between the perplexity of our model calculated on the "development" set — used

iteration number	TEST set L2R-PPL	TEST set 3-gram interpolated PPL
E0	167.47	152.25
E3	158.28	148.90
3-gram	167.14	167.14

Table 2: Interpolation with trigram results

for model parameter estimation — and "test" set — unseen data — shows that the initial point we choose for the parameter values has already captured a lot of information from the training data. The same problem is encountered in standard n-gram language modeling; however, our approach has more flexibility in dealing with it due to the possibility of reestimating the model parameters.

We believe that the above experiments show the potential of our approach for improved language models. Our future plans include:

• experiment with other parameterizations than the two most recent exposed heads in the word predictor model and parser;

• estimate a separate word predictor for left-to-right language modeling. Note that the corresponding model predictor was obtained via re-estimation aimed at increasing the probability of the "N-best" parses of the entire sentence;

• reduce vocabulary of parser operations; extreme case: no non-terminal labels/POS tags, word only model; this will increase the speed of the parser thus rendering it usable on larger amounts of training data and allowing the use of deeper stacks — resulting in more "N-best" derivations per sentence during re-estimation;

• relax — flatten — the initial statistics in the re-estimation of model parameters; this would allow the model parameters to converge to a different point that might yield a lower word-level perplexity;

• evaluate model performance on n-best sentences output by an automatic speech recognizer.

7 Acknowledgments

This research has been funded by the NSF IRI-19618874 grant (STIMULATE).

The authors would like to thank to Sanjeev Khudanpur for his insightful suggestions. Also to Harry Printz, Eric Ristad, Andreas Stolcke, Dekai Wu and all the other members of the dependency modeling group at the summer96 DoD Workshop for useful comments on the model, programming support and an extremely creative environment. Also thanks to Eric Brill, Sanjeev Khudanpur, David Yarowsky, Radu Florian, Lidia Mangu and Jun Wu for useful input during the meetings of the people working on our STIMULATE grant.

References

W. Byrne, A. Gunawardana, and S. Khudanpur. 1998. Information geometry and EM variants. Technical Report CLSP Research Note 17, Department of Electical and Computer Engineering, The Johns Hopkins University, Baltimore, MD.

C. Chelba, D. Engle, F. Jelinek, V. Jimenez, S. Khudanpur, L. Mangu, H. Printz, E. S. Ristad, R. Rosenfeld, A. Stolcke, and D. Wu. 1997. Structure and performance of a dependency language model. In *Proceedings of Eurospeech*, volume 5, pages 2775–2778. Rhodes, Greece.

Michael John Collins. 1996. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 184–191. Santa Cruz, CA.

A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. In *Journal of the Royal Statistical Society*, volume 39 of *B*, pages 1–38.

Frederick Jelinek and Robert Mercer. 1980. Interpolated estimation of markov source parameters from sparse data. In E. Gelsema and L. Kanal, editors, *Pattern Recognition in Practice*, pages 381–397.

F. Jelinek, J. Lafferty, D. M. Magerman, R. Mercer, A. Ratnaparkhi, and S. Roukos. 1994. Decision tree parsing using a hidden derivational model. In ARPA, editor, *Proceedings of the Human Language Technology Workshop*, pages 272–277.

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

Colin Philips. 1996. *Order and Structure*. Ph.D. thesis, MIT. Distributed by MITWPL.