Computer Speech and Language (2000) **14**, 283–332
Article No. 10.1006/csla.2000.0147
Available online at http://www.idealibrary.com on IDEAL

Structured language modeling

Ciprian Chelba†§ and Frederick Jelinek‡¶

†Center for Language and Speech Processing, Johns Hopkins University, Baltimore, MD, U.S.A., ‡Center for Language and Speech Processing, Johns Hopkins University, Baltimore, MD, U.S.A.

Abstract

This paper presents an attempt at using the syntactic structure in natural language for improved language models for speech recognition. The structured language model merges techniques in automatic parsing and language modeling using an original probabilistic parameterization of a shift-reduce parser. A maximum likelihood re-estimation procedure belonging to the class of expectation-maximization algorithms is employed for training the model. Experiments on the *Wall Street Journal* and *Switchboard* corpora show improvement in both perplexity and word error rate—word lattice rescoring—over the standard 3-gram language model.

1. Introduction

In the accepted statistical formulation of the speech recognition problem (Jelinek, 1998) the recognizer seeks to find the word string

$$\widehat{W} \doteq \arg\max_{W} P(A|W)\, P(W)$$

where A denotes the observable speech signal, $P(A|W)$ is the probability that when the word string W is spoken, the signal A results, and $P(W)$ is the *a priori* probability that the speaker will utter W.

The language model estimates the values $P(W)$. With $W = w_1, w_2, \ldots, w_n$, we get by Bayes' theorem,

$$P(W) = \prod_{i=1}^{n} P(w_i|w_1, w_2, \ldots, w_{i-1}). \tag{1}$$

Since the parameter space of $P(w_k|w_1, w_2, \ldots, w_{k-1})$ is too large—the words w_j belong to a vocabulary $\mathcal{V}$ whose size is in the tens of thousands—the language model is forced to put the *history* $W_{k-1} = w_1, w_2, \ldots, w_{k-1}$ into an equivalence class determined by a function $\Phi(W_{k-1})$. As a result,

$$P(W) \cong \prod_{k=1}^{n} P(w_k|\Phi(W_{k-1})). \tag{2}$$

§Currently with Microsoft Research, Microsoft Corp., Redmond, WA, U.S.A. E-mail: chelba@microsoft.com
¶E-mail: jelinek@jhu.edu

0885–2308/00/040283 + 50 $35.00/0

Research in language modeling consists of finding appropriate equivalence classifiers Φ and methods to estimate $P(w_k|\Phi(W_{k-1}))$.

The language model of state-of-the-art speech recognizers uses $(n-1)$-gram equivalence classification, that is, defines

$$\Phi(W_{k-1}) \doteq w_{k-n+1}, w_{k-n+2}, \ldots, w_{k-1}.$$

Once the form $\Phi(W_{k-1})$ is specified, only the problem of estimating $P(w_k|\Phi(W_{k-1}))$ from the training data remains.

In most cases, $n = 3$, which leads to a *trigram* language model. The latter has been shown to be surprisingly powerful and, essentially, all attempts to improve on it in the last 20 years have failed. The two interesting enhancements, facilitated by maximum entropy estimation methodology, have been the use of *triggers* (Rosenfeld, 1994) or of *singular value decomposition* (Bellegarda, 1997) (either of which dynamically identify the topic of discourse) in combination with n-gram models.

Measures of language model quality

Word Error Rate. The most desirable measure of success of a language model is the word error rate (WER). In order to calculate it we need to first find the most favorable word alignment between the hypothesis put out by the recognizer $\widehat{W}$ and the true sequence of words uttered by the speaker W—assumed to be known *a priori* for evaluation purposes only—and then count the number of incorrect words in $\widehat{W}$ per total number of words in W.

```
TRANSCRIPTION: UP UPSTATE NEW YORK SOMEWHERE UH     OVER OVER HUGE AREAS
HYPOTHESIS:       UPSTATE NEW YORK SOMEWHERE UH ALL ALL  THE  HUGE AREAS
               1  0       0   0    0         0  1   1    1    0    0
:4 errors per 10 words in transcription; WER = 40%
```

Perplexity. As an alternative to the computationally expensive WER, a statistical language model is evaluated by how well it predicts a string of symbols W_t—commonly referred to as *test data*—generated by the source to be modeled.

Assume we compare two models M_1 and M_2; they assign probabilities $P_{M_1}(W_t)$ and $P_{M_2}(W_t)$, respectively, to the sample test string W_t. The test string has neither been used nor seen at the estimation step of either model, and it was generated by the same source that we are trying to model. "Naturally", we consider M_1 to be a better model than M_2 if $P_{M_1}(W_t) > P_{M_2}(W_t)$.

A commonly used quality measure for a given model M is related to the entropy of the underlying source, and was introduced under the name of perplexity (PPL) (Jelinek, Mercer, Bahl & Baker, 1977):

$$PPL(M) = \exp\left(-1/N \sum_{k=1}^{N} \ln\left[P_M(w_k|W_{k-1})\right]\right). \tag{3}$$

Paper layout

The paper is organized as follows. Section 2 gives a basic description of the structured language model (SLM) and perplexity results on the UPenn Treebank corpus (Marcus, Santorini & Marcinkiewicz, 1995). We point out different research issues that we explore and detail later in the paper. Section 3 describes speech recognition experiments using the structured language model on different corpora: *Wall Street Journal* (WSJ, Doug & Baker, 1992) and

Switchboard (SWB, Godfrey, Holliman & McDaniel, 1992). Section 4 wraps up the loose ends and presents experiments that detail our choices or illustrate some less obvious aspects of the SLM. We conclude with Section 5, outlining the relationship between our approach and others in the literature, pointing out what we believe to be worthwhile future directions of research.

2. Structured language model

As indicated in Equation (2), a language model predicts the next word in a string of words based on an equivalence classification of the word prefix $\Phi(W_{k-1})$. In the case of the structured language model (SLM) this classification is really a mixture of possible equivalence classifications $\Phi^l(W_{k-1}), l = 1 \ldots N$ weighted by their probabilities $P(\Phi^l(W_{k-1})|W_{k-1})$. The component classifications and their probabilities result from the operation of a novel probabilistic shift-reduce push-down automaton (see Abney, McAllester & Pereira, 1999) that has been appropriately parameterized.

The basic ideas leading to the classification of the word prefix are outlined in the next section. One constraint imposed by the incremental operation of the language model associated with standard one-pass speech recognition search algorithms [see Equation (1)] is that it has to proceed from left to right through the word sequence. A two-pass decoding strategy, such as N-best rescoring would not bind us to using a left to right model. However, rescoring has less impact on the final recognizer output than if the language and acoustic models interact directly in the production of first-pass recognition hypotheses. Consequently, we focus on the left-to-right operation.

Section 2.2 presents a model that assigns probability to each possible pair consisting of a word sequence and parse. This is then used to assign probability $P(\Phi^l(W_{k-1})|W_{k-1})$ to each equivalence classification of W_{k-1} considered by the model. A mix of the classifications results in the word level probability $P(w_k|W_{k-1})$, as described in Section 2.4. A few shortcuts are introduced in order to make the computation feasible. Section 2.6 describes two successive stages of model parameter re-estimation. Finally, Section 2.7 presents experiments carried out on the UPenn Treebank corpus, and compares the results of our approach to those obtained from the standard 3-gram model.

2.1. Basic idea and terminology

Consider predicting the word `after` in the following sentence:

```
the contract ended with a loss of 7 cents after trading as low as 9 cents.
```

A 3-gram approach would predict `after` from `(7, cents)`, whereas it is intuitively clear that the strongest predictor would be `(contract, ended)`, which is outside the reach of even 7-grams.

The linguistically correct *partial parse* of the word history when predicting `after` is shown in Figure 1; the word ENDED is called the *headword*[1] of the *constituent*—the word that represents best the entire constituent (ENDED `ended` (WITH `with` `(...)`))—and ENDED is an *exposed headword* when predicting `after`—the topmost headword in the largest constituent that contains it. A model that uses the two most recent exposed headwords would

[1] Figure 1 shows the linguistically accepted headwords for each constituent. As will become clear later, our model allows any word in the constituent to be the headword with a certain probability. The model parameter re-estimation algorithm we are going to use for training the model adjusts these probabilities such that the headwords will be good for our purposes, namely predicting the next word in the sentence.

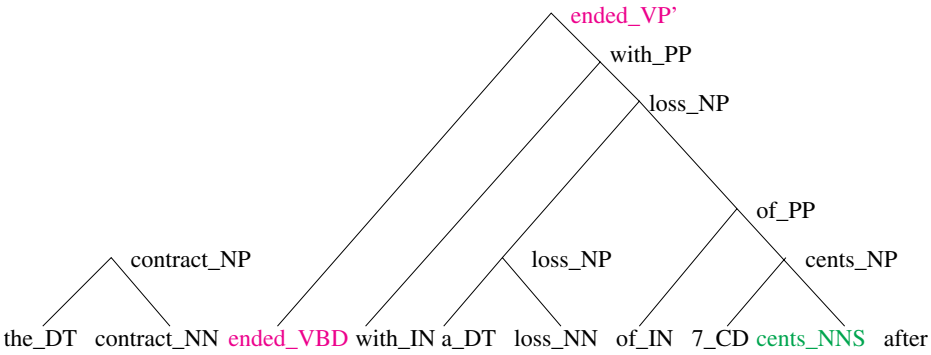

Figure 1. Partial parse: the terminal nodes are words given along with their POS tag, and the non-terminal nodes are annotated with the headword and non-terminal label.

predict `after` from CONTRACT, ENDED, in agreement with our intuition. Another intuitive argument in favor of the headword prediction is the fact that the headword context is invariant to the removal of the optional `(OF of (CENTS 7 cents ))` constituent—yielding a correct sentence—whereas the trigram predictor is not invariant to this change.

Our working hypothesis is that syntactic structure filters out irrelevant words and points to the important ones—exposed headwords—thus providing an equivalence classification of the word prefix and enabling the use of long-distance information when predicting the next word.

The SLM will attempt to build the syntactic structure incrementally while traversing the sentence left-to-right. It will assign a probability to every word sequence W and parse T—every possible POS tag assignment, binary branching parse, non-terminal label, and headword annotation for every constituent of T. The probability assignment is based on a simple encoding of the (W, T) pair described in the next section.

2.1.1. Word sequence and parse encoding

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 $W_k = w_0 \ldots w_k$ be the *word k-prefix* of the sentence and $W_k T_k$ the *word-and-parse k-prefix*. To stress this point, a word-and-parse k-prefix contains only those binary subtrees whose span is completely included in the word k-prefix, excluding w_0 =`<s>`. Single words along with their POS tag can be regarded as root-only trees. Figure 2 shows a word-and-parse k-prefix; `h_0 .. h_{-m}` are the *exposed heads*, each head being a pair (headword, non-terminal label), or (word, POS tag) in the case of a root-only tree. A *complete parse* (Fig. 3) is defined to be any binary parse of the `(<s>, SB)` $(w_1, t_1) \ldots (w_n, t_n)$ `(</s>, SE)` sequence—SB/SE is a distinguished POS tag for `<s>/</s>`, respectively—with the restrictions that:

- `(</s>, TOP)` is the only allowed head;
- $(w_1, t_1) \ldots (w_n, t_n)$ `(</s>, SE)` forms a constituent headed by `(</s>, TOP')`; the model allows parses where `(</s>, TOP')` is the head of any constituent that dominates `</s>` but not `<s>`.

Note that in a complete parse $(w_1, t_1) \ldots (w_n, t_n)$ *need not* form a constituent, but for the parses where it does, there is no restriction on which of its words is the headword or what is the non-terminal label that accompanies the headword. The set of complete parses is a

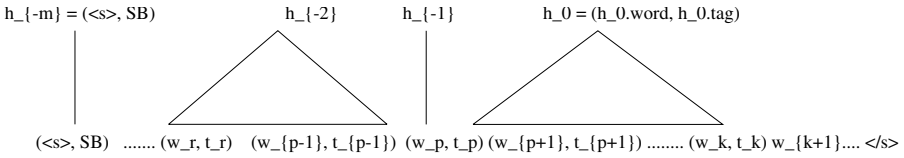

Figure 2. A word-and-parse k-prefix.

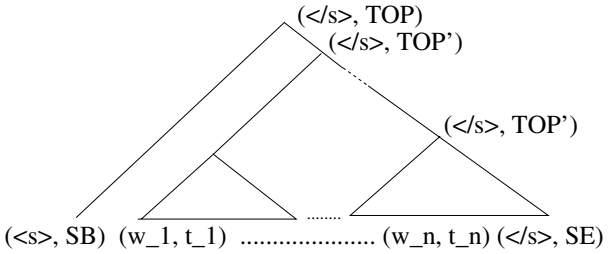

Figure 3. Complete parse.

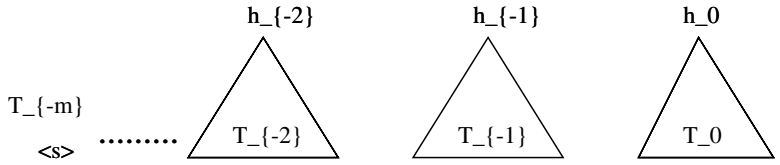

Figure 4. Before an adjoin operation.

superset of the parses in the UPenn Treebank which insist that $(w_1, t_1) \ldots (w_n, t_n)$ form a constituent. Our definition is imposed by the bottom-up, left-to-right operation of the model and the aim of using it as a language model. Once the end-of-sentence is predicted, the parse is wrapped up with probability one.

The SLM operates by means of three modules:

- WORD-PREDICTOR predicts the next word w_{k+1} given the word-and-parse k-prefix, and then passes control to the TAGGER;
- TAGGER predicts the POS tag t_{k+1} of the next word given the word-and-parse k-prefix and the newly predicted word w_{k+1}, and then passes control to the CONSTRUCTOR;
- CONSTRUCTOR grows the already existing binary branching structure by repeatedly generating transitions from the set: `(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. Obviously, the origin of the headword resulting from a unary transition is fully determined.

The operations performed by the CONSTRUCTOR 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.

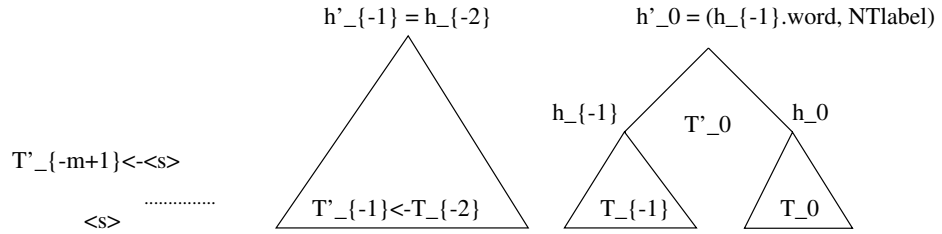

Figure 5. Result of adjoin-left under NTlabel.

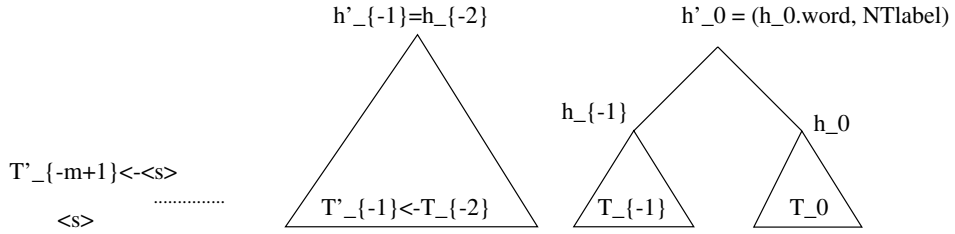

Figure 6. Result of adjoin-right under NTlabel.

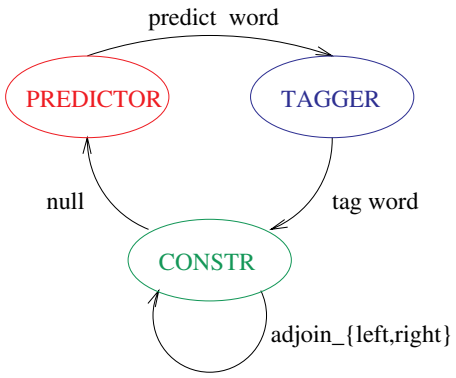

Figure 7. Language model operation as a finite state machine.

The finite state machine in Figure 7 presents a simplified operation of the model—it does not illustrate how the model deals with unary transitions.

The algorithm in Figure 8 formalizes the above description of the sequential generation of a sentence by a complete parse.

A unary transition is allowed only when the most recent exposed head is a leaf of the tree—a regular word along with its POS tag—hence it can be taken at most once at a given position in the input word string. This is an arbitrary modeling decision that avoids dealing with recursive unary transitions, which would be the case if unary transitions were always allowed. The second subtree in Figure 2 provides an example of the structure that results from a unary transition followed by a null transition.

It is easy to see that any given word sequence with a complete parse (see Fig. 3) and head-word annotation is generated by a unique sequence of model actions. This will prove very

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

Figure 8. Algorithm: SLM operation.

useful in initializing our model parameters from a treebank—see Section 2.6.1. Conversely, a generative model running according to the algorithm in Figure 8 can only generate a complete parse.

2.2. Probabilistic model

The language model operation provides an encoding of a given word sequence along with a parse tree (W, T) into a sequence of elementary model actions. In order to obtain a correct probability assignment $P(W, T)$ one has to simply assign proper conditional probabilities to each transition in the finite state machine that describes the model—see Figure 7. We are interested in a probability model able to distinguish between desirable and less desirable parses. The probabilities $P(W, T)$ will also prove useful in assigning probability at the word level.

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

$$P(W, T) = \prod_{k=1}^{n+1} [P(w_k|W_{k-1}T_{k-1}) \cdot P(t_k|W_{k-1}T_{k-1}, w_k) \cdot P(T_{k-1}^k|W_{k-1}T_{k-1}, w_k, t_k)] \quad (4)$$

$$P(T_{k-1}^k | W_{k-1}T_{k-1}, w_k, t_k) = \prod_{i=1}^{N_k} P(p_i^k | W_{k-1}T_{k-1}, w_k, t_k, p_1^k \dots p_{i-1}^k). \quad (5)$$

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;
- T_{k-1}^k is the incremental parse structure that generates $T_k = T_{k-1} \parallel T_{k-1}^k$ when attached to T_{k-1}; it is the parse structure built on top of T_{k-1} and the newly predicted word w_k; the $\parallel$ notation stands for concatenation;
- $N_k - 1$ is the number of operations the CONSTRUCTOR executes at position k of the input string before passing control to the WORD-PREDICTOR (the N_kth operation at position k is the `null` transition); N_k is a function of T;
- p_i^k denotes the ith CONSTRUCTOR action carried out at position k in the word string:
  $p_i^k \in \{$ `(adjoin-left, NTtag)`, `(adjoin-right, NTtag)`, `(unary, NTtag)`$\}$, $1 \le i < N_k$,
  $p_i^k =$ `null`, $i = N_k$.

Note that each $(W_{k-1}T_{k-1}, w_k, t_k, p_1^k \dots p_{i-1}^k)$ *defines a valid word-parse k-prefix* W_kT_k at position k in the sentence, $i = 1 \dots N_k$.

To ensure a proper probabilistic model over the set of *complete parses* for any sentence W, certain CONSTRUCTOR and WORD-PREDICTOR probabilities must be given specific values. The set of constraints on the probability values of different model components is consistent with the algorithm in Figure 8:

- $P($`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>`
  $P($`(adjoin-right, TOP')`$|W_kT_k) = 1$,
  if `h_0 = (</s>, TOP')` and `h_{-1}.word` $\neq$ `<s>`
  both ensure that the parse generated by our model is consistent with the definition of a complete parse;
- $\exists \epsilon > 0$ *s.t.* $\forall W_{k-1}T_{k-1}$, $P(w_k$=`</s>`$|W_{k-1}T_{k-1}) \ge \epsilon$ ensures that the model halts with probability one; once the end of sentence symbol `</s>` is generated, the model wraps up (completes) the parse with probability one.

2.2.1. Model component parameterization

In order to be able to estimate the model components we need to make appropriate equivalence classifications of the conditioning part for each component, respectively. The equivalence classification should identify the strong predictors in the context and allow reliable estimates from a treebank. Our choice relies heavily on exposed heads: the experiments in Chelba (1997) show that exposed heads are good predictors for the WORD-PREDICTOR component of the language model; Collins (1996) shows that they are useful for high accuracy parsing, making them the favorite choice for the CONSTRUCTOR model as well; our experiments showed that they are also useful in the TAGGER component model. Since the word to be tagged is in itself a very strong predictor for the POS tag, we limit the equivalence

classification of the TAGGER model to include only the NT labels of the two most recent exposed heads.

$$P(w_k|W_{k-1}T_{k-1}) = P(w_k|[W_{k-1}T_{k-1}]) = P(w_k|h_0, h_{-1}) \quad (6)$$

$$P(t_k|w_k, W_{k-1}T_{k-1}) = P(t_k|w_k, [W_{k-1}T_{k-1}]) = P(t_k|w_k, h_0 \cdot tag, h_{-1} \cdot tag) \quad (7)$$

$$P(p_i^k|W_kT_k) = P(p_i^k|[W_kT_k]) = P(p_i^k|h_0, h_{-1}). \quad (8)$$

By $[W_{k-1}T_{k-1}]$ we denote an equivalence classification of the word-parse prefix $W_{k-1}T_{k-1}$ suitable for estimating each of the above conditional probabilities and meaningful from a modeling point of view. We use the exposed heads as defined in Section 2.1.1: h_0 and h_{-1} denote the two most recent exposed heads.

Section 4.1 presents experiments justifying our choices in Equations(6)–(8). The above equivalence classifications are limited by the severe data sparseness problem faced by the 3-gram model, and by no means do we believe that they cannot be improved upon, especially those used in the CONSTRUCTOR model (8). Richer equivalence classifications should use a probability estimation method that deals better with sparse data than the one presented in Section 2.2.2.

It is worth noting that the standard 3-gram model belongs to the parameter space of our model: if the binary branching structure developed by the parser were always right-branching—the `null` transition has probability 1 in the CONSTRUCTOR model—and we mapped the POS tag vocabulary to a single type, then our model would become equivalent to a trigram language model.

2.2.2. Modeling tool

All model components (WORD-PREDICTOR, TAGGER, CONSTRUCTOR) are conditional probabilistic models of the type $P(u|z_1, z_2, \ldots, z_n)$ where $u, z_1, z_2, \ldots, z_n$ belong to a mixed set of words, POS tags, NTtags and CONSTRUCTOR actions (u only). Let $\mathcal{U}$ be the vocabulary in which the predicted random variable u takes values.

For simplicity, the probability estimation method we chose was recursive linear interpolation among relative frequency estimates of different orders $f_k(\cdot), k = 0 \ldots n$, using a recursive mixing scheme (see Fig. 9):

$$P_n(u|z_1, \ldots, z_n) = \lambda(z_1, \ldots, z_n) \cdot P_{n-1}(u|z_1, \ldots, z_{n-1}) + (1 - \lambda(z_1, \ldots, z_n)) \cdot f_n(u|z_1, \ldots, z_n), \quad (9)$$

$$P_{-1}(u) = uniform(\mathcal{U}) \quad (10)$$

where:

- $z_1, \ldots, z_n$ is the context of order n when predicting u;
- $f_k(u|z_1, \ldots, z_k)$ is the order-k relative frequency estimate for the conditional probability $P(u|z_1, \ldots, z_k)$:

$$f_k(u|z_1, \ldots, z_k) = C(u, z_1, \ldots, z_k)/C(z_1, \ldots, z_k), \qquad k = 0 \ldots n,$$

$$C(u, z_1, \ldots, z_k) = \sum_{z_{k+1} \in \mathcal{Z}_{k+1}} \ldots \sum_{z_n \in \mathcal{Z}_n} C(u, z_1, \ldots, z_k, z_{k+1} \ldots z_n),$$

$$C(z_1, \ldots, z_k) = \sum_{u \in \mathcal{U}} C(u, z_1, \ldots, z_k),$$

- $\lambda(z_1, \ldots, z_k) \in [0, 1], \ k = 0 \ldots n$ are the interpolation coefficients.

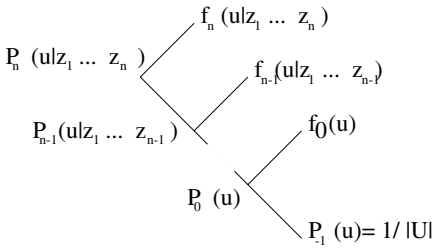

Figure 9. Recursive linear interpolation.

The $\lambda(z_1, \ldots, z_k)$ coefficients are grouped into equivalence classes ("tied") based on the range into which the count $C(z_1, \ldots, z_k)$ falls; the count ranges for each equivalence class—also called "buckets"—are set such that a statistically sufficient number of events $(u|z_1, \ldots, z_k)$ fall within that range. The approach is a standard one (Jelinek & Mercer, 1977). In order to determine the interpolation weights, we apply the deleted interpolation technique:

1. split the training data into two sets—"development" and "cross-validation", respectively;
2. get the relative frequency—maximum likelihood—estimates $f_k(u|z_1, \ldots, z_k),\ k = 0 \ldots n$ from "development" data;
3. employ the expectation-maximization (EM) algorithm (Dempster, Laird & Rubin, 1977) for determining the maximum likelihood estimate from "cross-validation" data of the "tied" interpolation weights $\lambda(C(z_1, \ldots, z_k))$.

The "cross-validation" data cannot be the same as the development data; if this were the case, the maximum likelihood estimate for the interpolation weights would be $\lambda(C(z_1, \ldots, z_k)) = 0$, disallowing the mixing of different order relative frequency estimates and thus performing no smoothing at all.

The deleted interpolation method is not optimal for our problem. Our models require a method able to optimally combine the predictors of different nature in the conditioning part of the model, and this is far from being met by the fixed hierarchical scheme used for context mixing. The best method would probably be maximum entropy (Berger, S. A. Della Pietra & V. J. Della Pietra, 1996), but due to its computational burden we have not used it.

2.3. Pruning strategy

Due to the above smoothing that results in the probability $P(W_k T_k)$, all binary parses with every possible headword annotation are allowed. The number of parses for a given word prefix W_k grows faster than exponential[2] with k. The state space of our model is huge, even for relatively short sentences. We thus have to prune most parses without discarding the most likely ones for a given prefix W_k. Our pruning strategy is a synchronous multi-stack search algorithm.

Each stack contains hypotheses (partial parses) that have been constructed by *the same number of PREDICTOR and the same number of CONSTRUCTOR operations*. The hypotheses in each stack are ranked according to the $\ln(P(W_k, T_k))$ score, with the highest on top. The ordered set of stacks containing partial parses with the same number of PREDICTOR

[2]Thanks to Bob Carpenter, Speechworks, for pointing out this inaccuracy in our Chelba and Jelinek (1998) paper.

operations, but different number of CONSTRUCTOR operations, is referred to as a *stack vector*.

The amount of search is controlled by two parameters:

- the maximum stack depth—the maximum number of hypotheses the stack can contain at any given time;
- 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 10 shows schematically the operations associated with the scanning of a new word w_{k+1}. P_k is the maximum number of adjoin operations for a k-length word prefix; since the tree is binary we have $P_k = k - 1$. First, all hypotheses in a given stack-vector are expanded by the following word. Then, for each possible POS tag the following word can take, we expand the hypotheses further. Due to the finite stack size, some are discarded. We then proceed with the CONSTRUCTOR expansion cycle, which takes place in two steps:

1. First, all hypotheses in a given stack are expanded with all possible CONSTRUCTOR actions excepting the `null` transition. The resulting hypotheses are sent to the immediately lower stack of the same stack-vector—the same number of WORD-PREDICTOR operations and exactly one more CONSTRUCTOR move. Some are discarded due to finite stack size.
2. After completing the previous step, all resulting hypotheses are expanded with the `null` transition and sent into the next stack-vector. Pruning can still occur due to the log-probability threshold on each stack.

A more detailed description—also computationally practical—of the pruning strategy is given in Section 4.3.

2.4. Left-to-right perplexity

To maintain a left-to-right operation of the language model, the probability assignment for the word at position $k + 1$ in the input sentence was made using:

$$P(w_{k+1}|W_k) = \sum_{T_k \in S_k} P(w_{k+1}|W_k T_k) \cdot \rho(W_k, T_k), \tag{11}$$

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

where S_k is the set of all parses present in our stacks at the current stage k. Assuming that the $P(w_{k+1}|W_k T_k)$ probability assignment is properly normalized, the word level probability assignment $P(w_{k+1}|W_k)$ is also properly normalized due to the fact that the interpolation coefficients $\rho(W_k, T_k)$ sum to 1.0. This leads to the following formula for evaluating the perplexity:

$$\text{L2R-PPL} = \exp\left(-1/N \sum_{i=1}^{N} \ln [P(w_i|W_{i-1})]\right). \tag{12}$$

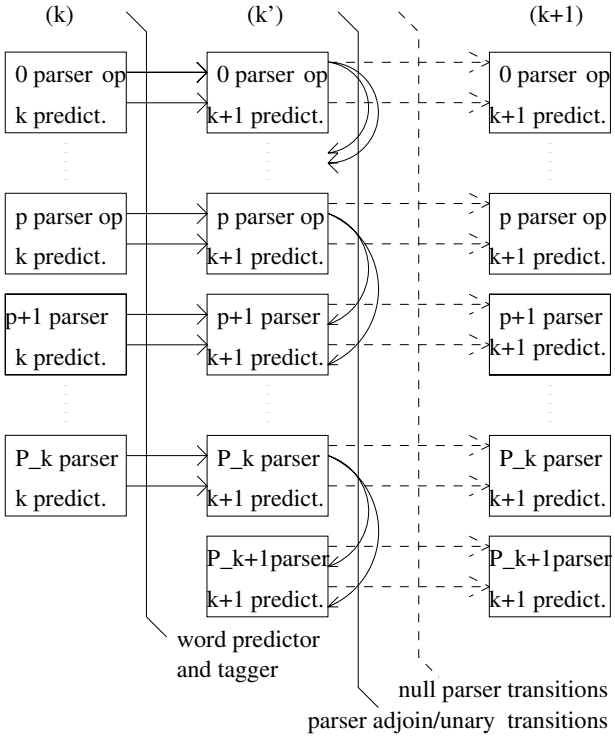

Figure 10. One search extension cycle.

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

$$P(W) = \sum_{k=1}^{N} P(W, T^{(k)}), \tag{13}$$

where $T^{(k)}$ is one of the "N-best"—in the sense defined by our search—parses for an entire sentence W. This is a deficient probability assignment, but it will turn out to be useful for justifying the model parameter re-estimation.

2.5. Separate left-to-right word predictor in the language model

An important observation is that for language modeling purposes, the next-word predictor probability $P(w_{k+1}|W_k T_k)$ in (11) *need not be the same* as the WORD-PREDICTOR probability (6) used to extract the structure T_k. Thus $P(w_{k+1}|W_k T_k)$ can be estimated separately for use in the language model. To be more specific, we can in principle have a WORD-PREDICTOR model component that operates within the parser model, whose role is to strictly extract syntactic structure, and a second L2R-WORD-PREDICTOR model that is used only for the left-to-right word-level probability assignment:

$$P_2(w_{k+1}|W_k) = \sum_{T_k \in S_k} P_{WP}(w_{k+1}|W_k T_k) \cdot \rho(W_k, T_k), \tag{14}$$

$$\rho(W_k, T_k) = P(W_k T_k) / \sum_{T_k \in S_k} P(W_k T_k). \tag{15}$$

This is desirable because many of the parses that are used for assigning probability to the next word are not going to survive the pruning strategy and thus will not be used in the "N-best training" stage of the re-estimation algorithm described in the next section. Estimating a separate L2R-WORD-PREDICTOR model as shown above attempts to remedy this weakness.

In this case, the interpolation coefficient given by (15) uses the regular WORD-PREDICTOR model, whereas the prediction of the next word for the purpose of word-level probability assignment is made using a separate model $P_{WP}(w_{k+1}|W_k T_k)$.

2.6. Model parameter estimation

As outlined in Section 2.4, the word-level probability assigned to a training/test set by our model is calculated using the proper word-level probability assignment in Equation (11). An alternative, which leads to a deficient probability model,[3] is to sum over all the complete parses that survived the pruning strategy, formalized in Equation (13).

The estimation procedure of the SLM parameters takes place in three stages:

1. Initialization of model parameters from a treebank.
2. Employment of the "N-best training" algorithm (see Section 2.6.2) to increase the training data "likelihood" calculated using the deficient probability assignment in Equation (13). The perplexity is still evaluated using the formula in Equation (11).
3. Estimation of a separate L2R-WORD-PREDICTOR model, such that the likelihood of the training data according to the probability assignment in Equation (14) is increased. The initial parameters for the L2R-WORD-PREDICTOR component are obtained by copying the WORD-PREDICTOR estimated at the previous stage.

As a final step in refining the model we have linearly interpolated the structured language model (11) with a trigram model.

Section 2.6.2 presents the basic idea behind the "N-best training" stage, leaving the details of the derivation for Appendix A. Section 2.6.3 presents the training of a separate L2R-WORD-PREDICTOR model—the second re-estimation stage.

2.6.1. First stage: Initial parameters

Each model component (WORD-PREDICTOR, TAGGER, CONSTRUCTOR) is initialized from a set of parsed sentences (a treebank) after the parses undergo headword percolation and binarization, as explained in this section.

Let us define the *derivation* $d(W, T)$ of a given binary parse to be the sequence of steps taken by the SLM when generating that particular parse. Due to the choice of elementary actions in the SLM, the derivation of any given binary parse with headword annotation is unique.

Separately for each mth model component, we then:

- gather joint counts $C^{(m)}(u^{(m)}, \underline{z}^{(m)})$ from the derivations that make up the "development data"—about 90% of the training data;
- estimate the interpolation coefficients on joint counts gathered from "check data"—the remaining 10% of the training data—using the EM algorithm (Dempster *et al.*, 1977). The buckets used for tying the interpolation weights are determined heuristically.

[3]The probabilities sum to less than 1.0.

These are the initial parameters used with the re-estimation procedure described in the previous section.

Headword percolation. In order to obtain training data for our model we need to binarize the UPenn Treebank (Marcus *et al.*, 1995)-style parse trees and percolate headwords. The procedure used was to first percolate headwords using a context-free (CF) rule-based approach and then binarize the parses by again using a rule-based approach. Inherently a heuristic process, we were satisfied with the output of an enhanced version of the procedure described in Collins (1996).

The procedure first decomposes a parse tree from the treebank into its phrase constituents, identified solely by the non-terminal/POS labels. Within each constituent we then identify the headword position and then, in a recursive third step, we fill in the headword position with the actual word percolated up from the leaves of the tree.

The headword percolation procedure is based on rules for identifying the headword position within each constituent. They are presented in Table I.[4]

Let $Z \rightarrow Y_1 \ldots Y_n$ be one of the CF rules that make up a given parse. We identify the headword position as follows:

- identify, in the first column of the table, the entry that corresponds to the Z non-terminal label;
- search $Y_1 \ldots Y_n$ from either left or right, as indicated in the second column of the entry, for the Y_i label that matches the regular expressions listed in the entry; the first matching Y_i is going to be the headword of the $(Z\ (Y_1 \ldots) \ldots (Y_n \ldots))$ constituent; the regular expressions listed in one entry are ranked in order from left to right: first, we try to match the first one, if unsuccessful we try the second one, and so on.

A regular expression of the type `<_CD|~QP>` matches any of the constituents listed between angular parentheses. `< ... >` are used for constituent types that are desired as headwords, whereas `<^ ... >` are used for constituent types that are not acceptable as headwords. For example, the
`<^_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>` regular expression will match any constituent that is *not*—list begins with `<^`—among any of the elements in the list between `<^` and `>`, in this case any constituent that is not a punctuation mark is eligible to be a headword. The terminal labels (POS tags) have `_` prepended to them—as in `_CD`; the non-terminal labels have the `~` prefix—as in `~QP`; `|` is merely a separator in the list.

Binarization. Our model works with binary trees. This is convenient for algorithmic reasons, but, not surprisingly, it is also favored by linguists on learnability grounds (see Haegeman, 1994).

Once the position of the headword within a constituent is identified to be k, we binarize the constituent—equivalent to 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 POS tags (only Y_i can be a POS tag)—as follows: a fixed rule is used to decide which of the two binarization schemes in Figure 11 should be applied, depending only on the value of Z. The intermediate nodes created by the above binarization schemes receive the non-terminal label Z'.

The choice between the two schemes is made according to the list of rules presented in Table II, based on the identity of the label on the left-hand-side of a CF rewrite rule. Notice

[4]The origin of this table of rules is not clear, but they are attributed to Magerman and Black.

TABLE I. Headword percolation rules

TOP	right	_SE _SB
ADJP	right	<~QP|_JJ|_VBN|~ADJP|_$|_JJR>
		<~~PP|~S|~SBAR|_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
ADVP	right	<_RBR|_RB|_TO|~ADVP>
		<~~PP|~S|~SBAR|_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
CONJP	left	_RB <~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
FRAG	left	<~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
INTJ	left	<~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
LST	left	_LS <~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
NAC	right	<_NNP|_NNPS|~NP|_NN|_NNS|~NX|_CD|~QP|_VBG>
		<~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
NP	right	<_NNP|_NNPS|~NP|_NN|_NNS|~NX|_CD|~QP|_PRP|_VBG>
		<~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
NX	right	<_NNP|_NNPS|~NP|_NN|_NNS|~NX|_CD|~QP|_VBG>
		<~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
PP	left	_IN _TO _VBG _VBN ~PP
		<~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
PRN	left	~NP ~PP ~SBAR ~ADVP ~SINV ~S ~VP
		<~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
PRT	left	_RP <~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
QP	left	<_CD|~QP> <_NNP|_NNPS|~NP|_NN|_NNS|~NX> <_DT|_PDT>
		<_JJR|_JJ> <~_CC|_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
RRC	left	~ADJP ~PP ~VP <~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
S	right	~VP <~SBAR|~SBARQ|~S|~SQ|~SINV>
		<~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
SBAR	right	<~S|~SBAR|~SBARQ|~SQ|~SINV>
		<~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
SBARQ	right	~SQ ~S ~SINV ~SBAR <~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
SINV	right	<~VP|_VBD|_VBN|_MD|_VBZ|_VB|_VBG|_VBP> ~S ~SINV
		<~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
SQ	left	<_VBD|_VBN|_MD|_VBZ|_VB|~VP|_VBG|_VBP>
		<~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
UCP	left	<~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
VP	left	<_VBD|_VBN|_MD|_VBZ|_VB|~VP|_VBG|_VBP>
		<~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
WHADJP	right	<~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
WHADVP	right	_WRB <~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
WHNP	right	_WP _WDT _JJ _WP$ ~WHNP
		<~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
WHPP	left	_IN <~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>
X	right	<~_.|_,|_''|_``|_`|_'|_:|_LRB|_RRB>

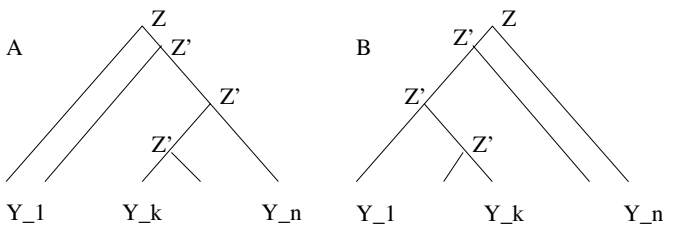

Figure 11. Binarization schemes.

TABLE II. Binarization rules

```
## first column : constituent label
## second column: binarization type : A or B
TOP     A
ADJP    B
ADVP    B
CONJP   A
FRAG    A
INTJ    A
LST     A
NAC     B
NP      B
NX      B
PP      A
PRN     A
PRT     A
QP      A
RRC     A
S       B
SBAR    B
SBARQ   B
SINV    B
SQ      A
UCP     A
VP      A
WHADJP  B
WHADVP  B
WHNP    B
WHPP    A
X       B
```

that whenever $k = 1$ or $k = n$—a case which is very frequent—the two schemes presented above yield the same binary structure.

Another problem when binarizing the parse trees is the presence of unary productions. Our model allows unary productions of the type $Z \rightarrow Y$ only, where Z is a non-terminal label and Y is a POS tag. The unary productions $Z \rightarrow Y$ where both Z and Y are non-terminal labels were deleted from the treebank, with only the Z constituent being retained: therefore, `(Z (Y (.) (.)))` becomes `(Z (.) (.))`.

2.6.2. N-best EM re-estimation

We would like to estimate the model component probabilities (6)–(8) such that the likelihood of the training data is increased. Since our problem is one of maximum likelihood estimation from incomplete data—the parse structure along with POS/NT tags and headword annotation for a given observed sentence is hidden—our approach makes use of the EM algorithm (Dempster *et al.*, 1977). Two specific modifications we make are:

- E-step: instead of scanning all the hidden events allowed—parses T—for a given observed one—sequence of words W—we restrict the algorithm to operate with N-best hidden events; N-best are determined using the search strategy described in Section 2.3. For a presentation of different modifications to EM, the reader is referred to Byrne, Gunawardana and Khudanpur (1998). For a better understanding, this proce-

dure can be thought of as analogous to a compromise between the forward-backward and Viterbi re-estimation for hidden Markov models.

- M-step: assuming that the count ranges and the corresponding interpolation values for each order are kept fixed to their initial values (see Section 2.6.1) the only parameters to be re-estimated using the EM algorithm are the maximal order counts $C^{(m)}(u, z_1, \ldots, z_n)$ for each model component. The interpolation scheme outlined in Section 2.2.2 is then used to obtain a smooth probability estimate for each model component.

Derivation of the re-estimation formulas, and further comments and experiments on the first model re-estimation stage are presented in the Appendices.

2.6.3. Last-stage parameter re-estimation

Once the model is trained according to the procedure described in the previous section, we proceed to a second stage of parameter re-estimation. In order to improve performance, we develop a model to be used strictly for word prediction [see (14)] different from the WORD-PREDICTOR model (6). We will call this new component the L2R-WORD-PREDICTOR.

In order to train this fourth model component, the key step is to recognize in (14) a hidden Markov model (HMM) with fixed transition probabilities—although dependent on the position k in the input sentence—specified by the $\rho(W_k, T_k)$ values.

The E-step of the EM algorithm (Dempster *et al.*, 1977) for gathering joint counts $C^{(m)}(y^{(m)}, \underline{x}^{(m)})$—L2R-WORD-PREDICTOR model—is the standard one, whereas the M-step uses the same count smoothing technique as that described in Section 2.6.2. The second re-estimation step operates directly on the $\mathcal{L}^{L2R}(\mathcal{C}, P_\theta)$ likelihood.

The second re-estimation pass is seeded with the WORD-PREDICTOR model joint counts $C^{(m)}(y^{(m)}, \underline{x}^{(m)})$ resulting from the first parameter re-estimation pass (see Section 2.6.2).

2.7. Preliminary perplexity results

During the original development of the SLM we chose to work on the UPenn Treebank corpus (Marcus *et al.*, 1995)—a subset of the WSJ (Wall Street Journal) corpus. This is a well known corpus, and the existence of a manual treebank makes it ideal for our experiments.

Unless specified otherwise in a specific section, the vocabulary sizes were:

1. word (also WORD-PREDICTOR operation) vocabulary: 10 k, open—all words outside the vocabulary are mapped to the `<unk>` token;
2. POS tag (also TAGGER operation) vocabulary: 40, closed;
3. non-terminal tag vocabulary: 52, closed;
4. CONSTRUCTOR operation vocabulary: 107, closed.

The corpus split into:

1. development set [929 564 wds (sections 00–20)];
2. check set [73 760 wds (sections 21–22)];
3. test data [82 430 wds (sections 23–24)].

The "development" and "check" sets were used strictly for initializing the model parameters as described in Section 2.6.1 and then with the re-estimation techniques described in Sections 2.6.2 and 2.6.3.

The parameters controlling the search (see Section 2.3) were set to:
`maximum-stack-depth` = 10 and `LnP-threshold` = 6.91.

TABLE III. Parameter re-estimation results

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
L2R0 (=E3)	21.26	158.28
L2R5	17.44	153.76

TABLE IV. Interpolation with trigram results

Iteration number	TEST set PPL		
	$\lambda = 0.0$	$\lambda = 0.36$	$\lambda = 1.0$
E0	167.47	152.25	167.14
E3	158.28	148.90	167.14
L2R0 (=E3)	158.28	148.90	167.14
L2R5	153.76	147.70	167.14

As explained in Section 2.6.2, the first stage of model parameter re-estimation re-evaluates the maximal order counts for each model component.

Each iteration involves parsing the entire training data which is a time consuming process—about 60 h of Sun Sparc Ultra-2 CPU-time. Table III shows the results of the re-estimation iterations; `E0-3` denote iterations of the re-estimation procedure described in Section 2.6.2; `L2R0-5` denote iterations of the re-estimation procedure described in Section 2.6.3. A deleted interpolation trigram model derived from the same training data had a perplexity of 167.14 on the same test data.

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 IV. The interpolation weight was estimated on check data to be $\lambda = 0.36$. An overall relative reduction of 11% over the trigram model was achieved.

2.7.1. Maximum depth factorization of the model

The word level probability assignment used by the SLM [Equation (11)] can be thought of as a model factored over different maximum reach depths. Let "depth" $D(T_k)$ be the number of words that need to be skipped starting from the end of the word prefix and going backwards in order to encounter the word that was exposed as the headword $h_{-1}.word$.

Equation (11) can be rewritten as:

$$P(w_{k+1}|W_k) = \sum_{d=0}^{d=k} P(d|W_k) \cdot P(w_{k+1}|W_k, d), \tag{16}$$

where:

$$P(d|W_k) = \sum_{T_k \in S_k} \rho(W_k, T_k) \cdot \delta(D(T_k), d)$$

TABLE V. Maximum depth evolution during training

Iteration number	Expected depth E[D]
E0	3.35
E1	3.46
E2	3.45

$$P(w_{k+1}|W_k, d) = \sum_{T_k \in S_k} P(T_k|W_k, d) \cdot P(w_{k+1}|W_k, T_k)$$
$$P(T_k|W_k, d) = \rho(W_k, T_k) \cdot \delta(D(T_k), d)/P(d|W_k).$$

We can interpret Equation (16) as a linear interpolation of models that reach back to different depths in the word prefix W_k. The expected value of $D(T_k)$ shows how far the SLM reaches in the word prefix:

$$E_{SLM}[D] = 1/N \sum_{k=0}^{k=N} \sum_{d=0}^{d=k} d \cdot P(d|W_k). \tag{17}$$

For the 3-gram model, we have $E_{3\text{-}gram}[D] = 2$. We evaluated the expected depth of the SLM using the formula in Equation (17). The results are presented in Table V.

It can be seen that the memory of the SLM is considerably deeper than that of the 3-gram model, whose depth is 2.

Figure 12 shows the distribution $P(d|W_k)$, averaged over all positions k in the test string:

$$P(d|W) = 1/N \sum_{k=1}^{N} P(d|W_k).$$

The non-zero value of $P(1|W)$ is due to the fact that the prediction of the first word in a sentence is based on the context of length 1—<s>—in both SLM and 3-gram models.

It can be seen that the SLM makes a prediction that reaches further back than the 3-gram model, on an average, in about 40% of cases.

Further experiments in the same set-up (UPenn Treebank, manually annotated data) are reported in Section 4.

3. Lattice rescoring using the structured language model

In a two-pass recognizer, a computationally feasible decoding step is run in the first pass, a set of hypotheses is retained as an intermediate result, and then a more sophisticated recognizer is run over these in a second pass—usually referred to as the rescoring pass. The search space in the second pass is much more restricted compared to the first pass, so we can afford to use better—usually also computationally more intensive—acoustic and/or language models.

The two most popular two-pass strategies differ mainly in the number of intermediate hypotheses saved after the first pass and the form in which they are stored.

In the so-called "N-best rescoring" method (the value of N is typically 100–1000), a list of complete hypotheses along with acoustic/language model scores are retained and then rescored using more complex acoustic/language models.

Due to the limited number of hypotheses in the N-best list, the second pass recognizer might be too constrained by the first pass, so a more comprehensive list of hypotheses is

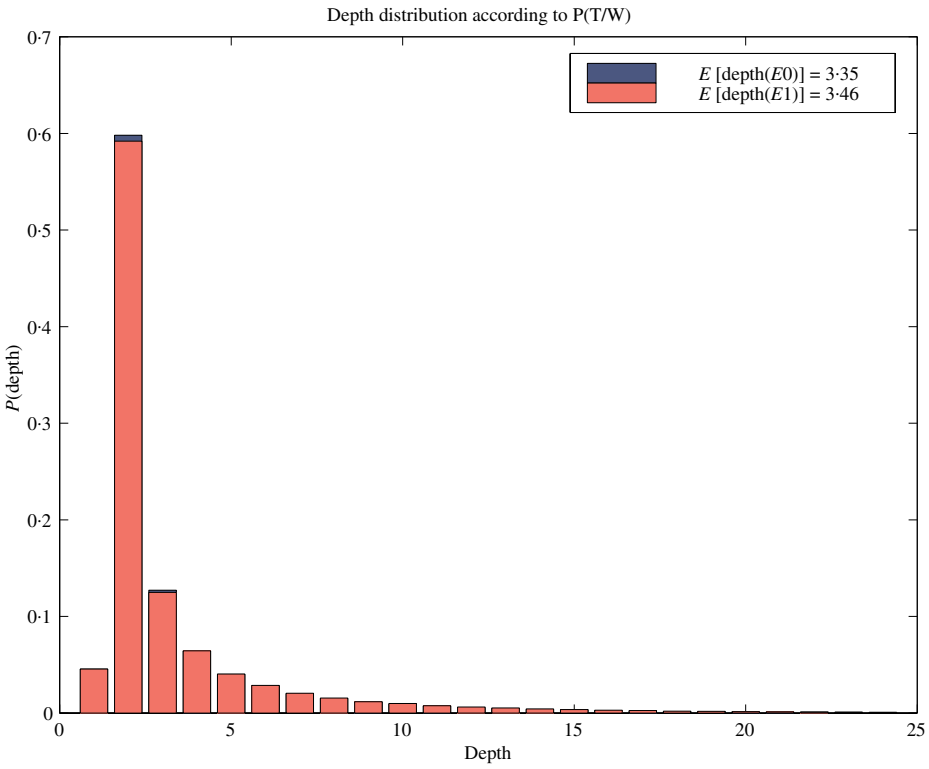

Figure 12. Structured language model maximum depth distribution.

often needed. The alternative preferred to N-best list rescoring is "lattice rescoring" (Ney, Aubert, Dugast & Steinbiss, 1994). The intermediate format in which the hypotheses are stored is now a directed acyclic graph in which the nodes are a subset of the language model states in the composite HMM, and the arcs are labeled with words. Typically, the first-pass acoustic/language model scores associated with each arc (or link) in the lattice are saved, and the nodes contain time alignment information.

For both cases, one can calculate the "oracle" word error rate (WER): the word error rate along the hypothesis with the minimum number of errors. The oracle WER decreases with the number of hypotheses saved.

Of course, a set of N-best hypotheses can be assembled as a lattice, the difference between the two being just in the number of different hypotheses—with different time alignments—stored in the lattice. One reason that makes the N-best rescoring framework attractive is the possibility to use "whole sentence" language models: models that are able to assign a score only to complete sentences due to the fact that they do not operate in a left-to-right fashion. The drawbacks are that the number of hypotheses explored is too small and their quality reflects the models used in the first pass. To clarify the latter assertion, assume that the second-pass language model is dramatically different from the one used in the first pass and that if we extracted the N-best list using the second (better) language model, different kinds of errors, specific to this language model, would be observed. In that case, simple

rescoring of the N-best list generated using the weaker language model may prevent the stronger language model from showing its merits.

It is thus desirable to have as complete a sample as possible of the possible word hypotheses, not biased towards a given model, and of a manageable size. This is what makes lattice rescoring the chosen method in our case.

There are several reasons that make A^* appealing for lattice decoding using the SLM:

- the lattice can be conceptually structured as a prefix tree of hypotheses—the time alignment is taken into consideration when comparing two word prefixes;
- the algorithm operates with whole prefixes x, making it ideal for incorporating language models whose memory is the entire utterance prefix;
- a reasonably good overestimate $h(y|x)$ and an efficient way to calculate $h_L(x)$ are readily available using the n-gram language model, as we will explain later.

We have applied the SLM for rescoring lattices to both WSJ and SWB. The word error rate reduction over a state-of-the-art 3-gram model was:

- 0.8% absolute (13.7 to 12.9%, 6% relative) on WSJ;
- 0.7% absolute (41.3 to 40.5%, 2% relative) on SWB.

We also evaluated the perplexity reduction relative to a standard deleted interpolation 3-gram *trained under the same conditions as the SLM*. We have achieved a relative reduction in PPL of 10 and 5% on WSJ and SWB, respectively.

As a secondary result we found out that lattice rescoring is not justified over simple N-best rescoring if the model used for rescoring is as computationally intensive as the SLM: due to the large amount of computation we had to use an approximation for the A^* search, and the same improvement could have been obtained using N-best rescoring. A faster implementation of the SLM could, however, exploit the potential advantages presented by the lattice rescoring framework, be it only an adjustable depth in the N-best list rescoring. We considered it a worthwhile direction for future research.

The remainder of this section is organized as follows. First, we briefly describe the A^* algorithm, mainly to establish the notation to be used in Section 3.2, which details our implementation of A^* for lattice decoding using the SLM. Section 3.4 then describes our experiments on the WSJ and SWB corpora.

3.1. A^* Algorithm

The A^* algorithm (Nilsson, 1971) is a tree-search strategy that could be compared to depth-first tree-traversal: pursue the most promising path as deeply as possible.

Let a set of hypotheses

$$L = \{h : x_1, \ldots, x_n\},\ x_i \in \mathcal{W}^* \qquad \forall\, i = 1 \ldots n$$

be organized as a prefix tree. We wish to obtain the maximum scoring hypothesis under the scoring function $f : \mathcal{W}^* \to \Re$:

$$h^* = \arg\max_{h \in L} f(h)$$

without scoring all the hypotheses in L, if possible, with a minimal computational effort.

The algorithm operates with prefixes and suffixes of hypotheses in the set L; we will denote prefixes (anchored at the root of the tree) by x and suffixes (anchored at a leaf) by y. A complete hypothesis h can be regarded as the concatenation of an x prefix and a y suffix:

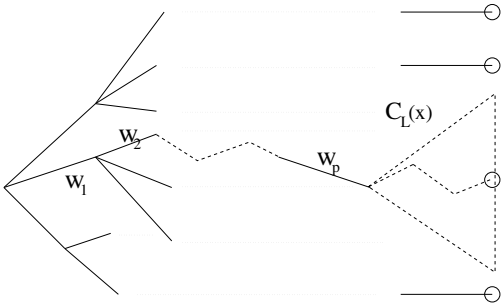

Figure 13. Prefix tree organization of a set of hypotheses L.

$h = x.y$. We assume that the function $f(\cdot)$ can be evaluated at any prefix x, i.e. $f(x)$ is a meaningful quantity.

To be able to pursue the most promising path, the algorithm needs to evaluate all the possible suffixes for a given prefix $x = w_1, \ldots, w_p$ that are allowed in L (see Fig. 13). Let $C_L(x)$ be the set of suffixes allowed by the tree for a prefix x and assume we have an overestimate for the $f(x.y)$ score of any *complete* hypothesis $x.y$, $g(x.y)$:

$$g(x.y) \doteq f(x) + h(y|x) \geq f(x.y).$$

Imposing the condition that $h(y|x) = 0$ for empty y, we have

$$g(x) = f(x), \forall\, complete\ x \in L,$$

that is, the overestimate becomes exact for complete hypotheses $h \in L$. Let the A^* *ranking function* $g_L(x)$ be defined as:

$$g_L(x) \doteq \max_{y \in C_L(x)} g(x.y) = f(x) + h_L(x),\ where \tag{18}$$

$$h_L(x) \doteq \max_{y \in C_L(x)} h(y|x) \tag{19}$$

$g_L(x)$ is an overestimate for the $f(\cdot)$ score of any complete hypothesis that has the prefix x; the overestimate becomes exact for complete hypotheses:

$$g_L(x) \geq f(x.y), \qquad \forall y \in C_L(x) \tag{20}$$

$$g_L(h) = f(h), \qquad \forall\, complete\ h \in L. \tag{21}$$

The A^* algorithm uses a potentially infinite stack—the stack need not be larger than $|L| = n$—in which prefixes x are arranged in decreasing order of the A^* ranking function $g_L(x)$; at each extension step the top-most prefix $x = w_1, \ldots, w_p$ is popped from the stack, expanded with all possible one-symbol continuations of x in L, and then all the resulting expanded prefixes—among which there may be complete hypotheses as well—are inserted back into the stack. The stopping condition is: whenever the popped hypothesis is complete, retain it as the overall best hypothesis h^* (see the algorithm in Fig. 14).

In practice the $h(y|x)$ function is chosen heuristically.

3.2. A^* for lattice decoding

Before explaining our approach to lattice decoding using the A^* algorithm, let us define a few terms.

```
//empty_hypothesis;
//top_most_hypothesis;
//a_hypothesis;
insert empty_hypothesis in stack;
do
{ // one Astar extension step
    top_most_hypothesis = pop top-most hypothesis from stack;

    for all possible one symbol continuations w of top_most_hypothesis
    {
        a_hypothesis = expand top_most_hypothesis with w;
        insert a_hypothesis in stack;
    }
}while(top_most_hypothesis is incomplete)
//top_most_hypothesis is the highest f(.) scoring one
```

Figure 14. Algorithm: A^* search.

The lattices we work with retain the following information after the first pass:

- time-alignment of each node;
- for each link connecting two nodes in the lattice we retain:
  - — word identity $w(link)$;
  - — acoustic model score—log-probability of acoustic segment covered by the link given the word, $\log P_{AM}(A(link)|w, link)$; to make this possible, the ending nodes of the link must contain all contextual information necessary for assigning acoustic model scores; for example, in a crossword triphone system, all the words labeling the links leaving the end node must have the same first phone;
  - — n-gram language model score—log-probability of the word, $\log P_{NG}(w|link)$; again, to make this possible, the start node of the link must contain the context $(n-1)$-gram—it is a state in the finite state machine describing the n-gram language model used to generate the lattice; we thus refer to a lattice as being bigram or trigram depending on the order of the language model used for generating it.

The lattice has a unique starting and ending node, respectively.

A *link* in the lattice is an arc connecting two nodes of the lattice.

A *path* p through the lattice is an ordered set of links $l_0 \dots l_n$ with the constraint that any two consecutive links cover adjacent time intervals:

$$p = \{l_0 \dots l_n : \forall i = 0 \dots n-1, ending\_node(l_i) = starting\_node(l_{i+1})\}. \tag{22}$$

We will refer to the starting node of l_0 as the starting node of path p, and to the ending node of l_n as the ending node of path p.

A *partial path* is a path whose starting node is the same as the starting node of the entire lattice, and a *complete path* is one whose starting/ending nodes are the same as those of the entire lattice, respectively.

With the above definitions, a lattice can be conceptually organized as a prefix tree of paths. When rescoring the lattice using a different language model than the one that was used in the

first pass, we seek to find the complete path $p = l_0 \dots l_n$ maximizing:

$$f(p) = \sum_{i=0}^{n} [\log P_{AM}(l_i) + LMweight \cdot \log P_{LM}(w(l_i)|w(l_0) \dots w(l_{i-1})) - \log P_{IP}], \quad (23)$$

where:

- $\log P_{AM}(l_i)$ is the acoustic model log-likelihood assigned to link l_i;
- $\log P_{LM}(w(l_i)|w(l_0) \dots w(l_{i-1}))$ is the language model log-probability assigned to link l_i given the previous links on the partial path $l_0 \dots l_i$;
- $LMweight > 0$ is a constant weight that multiplies the language model score of a link; its theoretical justification is unclear, but experiments show its usefulness;
- $\log P_{IP} > 0$ is the "insertion penalty"; again, its theoretical justification is unclear, but experiments show its usefulness.

In order to be able to apply the A^* algorithm, we need to find an appropriate stack entry scoring function $g_L(x)$, where x is a partial path and L is the set of complete paths in the lattice. Going back to definition (18) of $g_L(\cdot)$, we need an overestimate $g(x.y) = f(x) + h(y|x) \geq f(x.y)$ for all possible $y = l_k \dots l_n$ complete continuations of x allowed by the lattice. We propose to use the following heuristic:

$$\begin{aligned} h(y|x) = &\sum_{i=k}^{n} [\log P_{AM}(l_i) + LMweight \cdot (\log P_{NG}(l_i) + \log P_{COMP}) - \log P_{IP}] \\ &+ LMweight \cdot \log P_{FINAL} \cdot \delta(k < n). \end{aligned} \quad (24)$$

A simple calculation shows that if $\log P_{LM}(l_i)$ satisfies

$$\log P_{NG}(l_i) + \log P_{COMP} \geq \log P_{LM}(l_i), \qquad \forall l_i,$$

then $g_L(x) = f(x) + \max_{y \in C_L(x)} h(y|x)$ is a an appropriate choice for the A^* stack entry scoring function.

The justification for the $\log P_{COMP}$ term is that it is supposed to compensate for the per word difference in log-probability between the n-gram model, NG and the superior model, LM with which we rescore the lattice—hence $\log P_{COMP} > 0$. Its expected value can be estimated from the difference in perplexity between the two models, LM and NG. Theoretically we should use a higher value than the maximum pointwise difference between the two models:

$$\log P_{COMP} \geq \max_{\forall l_i} [\log P_{LM}(l_i|l_0 \dots l_{i-1}) - \log P_{NG}(l_i)],$$

but in practice we set it by trial and error, starting with the expected value as an initial guess.

The $\log P_{FINAL} > 0$ term is used for practical considerations as explained in Section 3.3. The calculation of $g_L(x)$ (18) is made very efficient using the dynamic programming technique in the Viterbi algorithm (Viterbi, 1967). Indeed, for a given lattice L, the value of $h_L(x)$ is completely determined by the identity of the ending node of x; a Viterbi backward pass over the lattice can store at each node the corresponding value of $h_L(x) = h_L(ending\_node(x))$, such that it is readily available in the A^* search.

3.3. A^* search: Some practical considerations

In practice one cannot maintain a potentially infinite stack during the A^* search over the lattice. We chose to control the stack depth using two thresholds: one on the maximum number

of entries in the stack, called `stack-depth-threshold` and the other on the maximum log-probability difference between the top-most and the bottom-most hypotheses in the stack, called `stack-logP-threshold`.

There is a clear interaction between the quality of the stack entry scoring function (18) and the number of hypotheses explored, which in practice has to be controlled by the maximum stack size. A gross overestimate used in connection with a finite stack may lure the search to a cluster of paths that is suboptimal—the desired cluster of paths may fall out of the stack if the overestimate happens to favor a wrong cluster.

Also, longer prefixes—thus having shorter suffixes—benefit less from the per word $\log P_{COMP}$ compensation, which means that they may fall out of a stack already full with shorter hypotheses that have high scores due to compensation. This is the justification for the $\log P_{FINAL}$-term in the compensation function $h(y|x)$: the variance $\mathrm{var}[\log P_{LM}(l_i|l_0 \ldots l_{i-1}) - \log P_{NG}(l_i)]$ is a finite positive quantity, so the compensation is likely to be closer to the expected value $E[\log P_{LM}(l_i|l_0 \ldots l_{i-1}) - \log P_{NG}(l_i)]$ for longer y continuations than for shorter ones; introducing a constant $\log P_{FINAL}$ term is equivalent to an adaptive $\log P_{COMP}$ depending on the length of the y suffix—smaller equivalent $\log P_{COMP}$ for long y suffixes for which $E[\log P_{LM}(l_i|l_0 \ldots l_{i-1}) - \log P_{NG}(l_i)]$ is a better estimate for $\log P_{COMP}$ than it is for shorter ones.

Because the structured language model is computationally expensive, a strong limitation is being placed on the width of the search—controlled by the `stack-depth-threshold` and the `stack-logP-threshold`. For an acceptable search width (runtime) one seeks to tune the compensation parameters in order to maximize performance measured in terms of WER. However, the correlation between these parameters and the WER is not clear, and this makes the diagnosis of search problems extremely difficult. Our method for choosing the search parameters was to sample a few complete paths $p_1, \ldots, p_N$ from each lattice, rescore those paths according to the $f(\cdot)$ function (23) and then rank the h^* path output by the A^* search among the sampled paths. A correct A^* search should result in average rank 0. In practice this does not happen but one can trace the top-most path p^* in the offending cases—$p^* \neq h^*$ and $f(p^*) > f(h^*)$:

- if a prefix of the p^* hypothesis is still present in the stack when A^* returns, then the search failed strictly because of insufficient compensation;
- if no prefix of p^* is present in the stack, then the incorrect search outcome was caused by an interaction between compensation and insufficient search width.

The method we chose for sampling paths from the lattice was an N-best search using the n-gram language model scores. This is appropriate for pragmatic reasons, since one prefers lattice rescoring to N-best list rescoring exactly because of the possibility of extracting a path that is not among the candidates proposed in the N-best list, as well as practical reasons, since they are among the "better" paths in terms of WER.

3.4. Speech recognition experimental set-up

The set of experiments presented in Section 2.7 showed improvement in perplexity over the 3-gram language model. The experimental set-up is, however, fairly restrictive and artificial when compared to a real-world speech recognition task:

- although the headword percolation and binarization procedure is automatic, the tree-bank used as initial training data was generated by human annotators;

- the amount of training data (approximatively 1 million words) is small, albeit statistically significant, compared to that used for developing language models in real-world speech recognition experiments;
- the word-level tokenization of treebank text is different than that used in the speech recognition community, the former being tuned to facilitate linguistic analysis.

In the remaining part of this section we will briefly describe the set-up used for speech recognition experiments involving the structured language model, followed by the main results and conclusions of the experiments. For clarity of presentation, details on the experimental set-up are delayed until Section 4.5. The experiments were run on two different corpora (SWB and WSJ) sampling two extremes of the speech recognition spectrum: conversational speech over a telephone line at one end and read grammatical text recorded in ideal acoustic conditions at the other.

In order to evaluate our model's potential as a part of a speech recognizer, we had to address the problems outlined below:

- Manual vs. automatic parse trees. There are two corpora for which there exist treebanks (although of limited size): WSJ and SWB. The UPenn Treebank (Marcus *et al.*, 1995) contains manually parsed WSJ text. There also exists a small part of SWB that was manually parsed at UPenn—approx. 20 000 words. This allows training of an automatic parser to be used to generate an *automatic treebank*, possibly with a slightly different word-tokenization than that of the manual treebanks. We evaluated the sensitivity of the structured language model to this aspect and showed that the re-estimation procedure presented in Section 2.6 is powerful enough to overcome this handicap.
- More training data. The availability of an automatic parser to generate parse trees for the SLM training data—used for initializing the SLM—opens the possibility of training the model on much more data than those used in the experiments presented in Section 2.7. The only limitations are of computational nature, imposed by the speed of the parser used to generate the automatic treebank and the efficiency and speed of the re-estimation procedure for the structured language model parameters. As our experiments show, the re-estimation procedure leads to sufficiently better models—under both measures of perplexity and word error rate—that it is worth employing it even though its current implementation is slow. In practice, the speed of the SLM is the limiting factor on the amount of training data. For SWB we have only 2 million words of language modeling training data, so this is not an issue. For WSJ we were able to use 20 million words of training data, less than the 40 million words used by standard language models for this task.
- Different tokenization. We address this problem in Section 4.5.

The refinement of the SLM presented in Section 4.4.1, Equations (14) and (15) was not used at all during the following experiments due to the resulting low improvement in perplexity for the required computational cost.

3.5. Perplexity results

As a first step, we evaluated the perplexity performance of the SLM relative to that of a deleted interpolation 3-gram model *trained under the same conditions*. As outlined in the previous section, we worked on the CSR-Treebank corpora.

TABLE VI. WSJ-CSR-Treebank perplexity results

Language model	L2R perplexity				
	DEV set		TEST set		
λ	0.0	1.0	0.0	0.4	1.0
3-gram + SLM E0	39.1	33	151.9	135.9	147.8
3-gram + SLM E1	34.6	33	144.1	132.8	147.8

3.5.1. Wall Street Journal perplexity results

Due to the low speed of the current implementation of the SLM when used as a parser, which is the computational bottle-neck of the re-estimation procedure, the size of the training data was limited to 20 Mwds; the size of the test data set aside for perplexity measurements was 3.4 kwds—the DARPA'93 HUB1 test set transcriptions.

The parameters controlling the search for the most likely parses in the SLM (see Section 2.3) were set to:
`maximum-stack-depth` = 5 and `LnP-threshold` = 6.91.

We used the DARPA'93 HUB1 standard open vocabulary of size 20 kwds, tokenized such that it matched the UPenn Treebank text. Similar to the experiments reported in Section 2.7, we built a deleted interpolation 3-gram model, which was used as a baseline; we also linearly interpolated the SLM with the 3-gram baseline model, showing a significant reduction in perplexity:

$$P(w_i|W_{i-1}) = \lambda \cdot P_{3\text{-}gram}(w_i|w_{i-1}, w_{i-2}) + (1-\lambda) \cdot P_{SLM}(w_i|W_{i-1}).$$

The interpolation weight was set to $\lambda = 0.4$. The results are presented in Table VI; `E0-1` denote iterations of the re-estimation procedure described in Section 2.6.2.

3.5.2. Switchboard perplexity results

For the SWB experiments, the size of the training data was 2.29 Mwds, the size of the test data set aside for perplexity measurements was 28 kwds (WS97 DevTest, CLSP, 1997), and the closed vocabulary was 22 kwds.

The parameters controlling the search for the most likely parses in the SLM (see Section 2.3) were set to:
`maximum-stack-depth` = 10 and `LnP-threshold` = 6.91.

Again, we also linearly interpolated the SLM with the deleted interpolation 3-gram baseline, showing a modest reduction in perplexity:

$$P(w_i|W_{i-1}) = \lambda \cdot P_{3\text{-}gram}(w_i|w_{i-1}, w_{i-2}) + (1-\lambda) \cdot P_{SLM}(w_i|W_{i-1}).$$

The interpolation weight was determined on a held-out set to be $\lambda = 0.4$. The results are presented in Table VII; `E0-4` denote iterations of the re-estimation procedure described in Section 2.6.2. The perplexity reduction is smaller than that achieved on the WSJ text. We believe this is because the syntactic structure in SWB sentences is very poor—the average sentence length is 7.

3.6. Lattice decoding results

We proceeded to evaluate the WER performance of the SLM using the A^* lattice decoder described in Section 3.2. Before describing the experiments we need to clarify the point that there are two language model scores associated with each link in the lattice:

TABLE VII. SWB-CSR-Treebank perplexity results

Language model	L2R perplexity				
	DEV set		TEST set		
λ	0.0	1.0	0.0	0.4	1.0
3-gram + SLM E0	23.9	22.5	72.1	65.8	68.6
3-gram + SLM E4	22.7	22.5	71.0	65.4	68.6

- the language model score assigned by the model that generated the lattice, referred to as the LAT3-gram; this model operates on text in the CSR tokenization;
- the language model score assigned by rescoring each link in the lattice with the deleted interpolation 3-gram, built on the data in the CSR-Treebank tokenization, referred to as the TRBNK3-gram.

3.6.1. Wall Street Journal lattice decoding results

The lattices for which we ran decoding experiments were obtained using the standard DARPA'93 HUB1 language model (LAT3-gram), built using the standard 20 k *open* vocabulary and trained on additional training data—the standard for WSJ is about 40 Mwds. The deleted interpolation 3-gram model (TRBNK3-gram), built on less training data (20 Mwds) and using the same open vocabulary retokenized, is much weaker than the one used for generating the lattices, as confirmed by our experiments. Consequently, we ran lattice rescoring experiments in two set-ups:

- using the language model that generated the lattice (LAT3-gram) as the baseline model; language model scores are available in the lattice.
- using the TRBNK3-gram language model under the same training conditions as the SLM; we had to assign new language model scores to each link in the lattice.

Comparison between LAT3-gram and TRBNK3-gram. A first batch of experiments evaluated the power of the two 3-gram models at our disposal—baseline results for the WSJ experiments.

The LAT3-gram scores are available in the lattice from the first pass and we can rescore each link in the lattice using the TRBNK3-gram model. The Viterbi algorithm can be used to find the best path through the lattice according to the scoring function—see Equation (23) where $\log P_{LM}(\cdot)$ can be either of the above or a linear combination of the two. Notice that the linear interpolation of link language model scores

$$P(l) = \lambda \cdot P_{LAT3\text{-}gram}(l) + (1-\lambda) \cdot P_{TRBNK3\text{-}gram}(l)$$

does not lead to a proper probabilistic model due to the tokenization mismatch between the two models. To correct this problem we adjusted the operation of the TRBNK3-gram model, such that it takes two steps when it encounters a split link:

$$\begin{aligned} P(don't|x,y) = {} & \lambda \cdot P_{LAT3\text{-}gram}(don't|x,y) + \\ & (1-\lambda) \cdot P_{TRBNK3\text{-}gram}(do|x,y) \cdot P_{TRBNK3\text{-}gram}(n't|y,do). \end{aligned} \tag{25}$$

The lattices were created using the LAT3-gram model which is much stronger than the TRBNK3-gram: it was trained on more training data—a typical value for WSJ is 40 Mwds—and it can be considered a phrase language model.[5]

[5]That is, it treats frequent two-word "phrases" like do n't as being a single token.

TABLE VIII. 3-gram language model: Viterbi decoding results

λ	0.0	0.2	0.4	0.6	0.8	1.0
WER(%)	**14.7**	14.2	13.8	13.7	13.5	**13.7**

TABLE IX. LAT3-gram + structured language model: A^* decoding results

Decoder	A^*						Viterbi
λ	0.0	0.2	0.4	0.6	0.8	1.0	1.0
WER(%) (E0 SLM)	**14.5**	13.1	12.9	13.1	13.3	**13.7**	**13.7**
WER(%) (E3 SLM)	**14.2**	13.3	13.2	13.1	13.1	**13.7**	**13.7**

The parameters in Equation (23) were set to: $LMweight = 16$, $\log P_{IP} = 0$. The results are presented in Table VIII.

LAT3-gram-driven search using the SLM. A second batch of experiments evaluated the performance of the SLM. The perplexity results show that interpolation with the 3-gram model is beneficial for our model. The previous experiments show that the LAT3-gram model is more powerful than the TRBNK3-gram model. The interpolation

$$P(l) = \lambda \cdot P_{LAT3\text{-}gram}(l) + (1 - \lambda) \cdot P_{SLM}(l)$$

between the LAT3-gram model and the SLM is illegitimate for the same reasons as that between the LAT3-gram and the TRBNK3-gram models, and we correct it in the same way as for the TRBNK3-gram [see Equation (25)]. We use the interpolated language model for lattice decoding and obtain an improvement over the baseline result (WER = 13.7%). The results for different interpolation coefficient values are shown in Table IX.

The parameters controlling the search for the most likely parses in the SLM (see Section 2.3) were set to:
`maximum-stack-depth` = 10 and `LnP-threshold` = 6.91.

As explained previously, due to the fact that the SLM's memory extends over the entire prefix, we need to apply the A^* algorithm to find the overall best path in the lattice.

For tuning the search parameters we applied the N-best lattice sampling technique described in Section 3.3. Unfortunately, the average rank of the A^* hypothesis was not very sensitive to different values of the compensation coefficients ($\log P_{COMP}$, $\log P_{FINAL}$) and the search width (`stack-depth-threshold`, `stack-depth-logP-threshold`), such that their choice was dictated by running time considerations.

The $\log P_{COMP}$, $\log P_{FINAL}$ and `stack-depth-threshold`, `stack-depth-logP-threshold` were optimized directly on test data for the best interpolation value found in the perplexity experiments. In all other experiments they were kept fixed to these values. The $LMweight$, $\log P_{IP}$ parameters are the ones typically used with the 3-gram model for the WSJ task.

For the WSJ experiments, the parameters controlling the A^* search were set to: $\log P_{COMP} = 0.5$, $\log P_{FINAL} = 0$, $LMweight = 16$, $\log P_{IP} = 0$, `stack-depth-threshold=30`, `stack-depth-logP-threshold=100` [see Equations (23) and (24)].

The results are presented in Table IX. The structured language model achieved an absolute improvement in WER of 0.8% (6% relative) over the baseline (see Table VIII).

TABLE X. TRBNK3-gram + structured language model: A^* decoding results

Decoder	A^*						Viterbi
λ	0.0	0.2	0.4	0.6	0.8	1.0	1.0
WER(%) (E0 SLM)	14.5	13.6	14.3	14.1	14.4	14.7	14.7
WER(%) (E3 SLM)	13.7	14.0	14.3	14.2	14.3	14.7	14.7

TRBNK3-gram-driven search using the SLM. We rescored each link in the lattice using the TRBNK3-gram language model, and used this as a baseline for further experiments. As shown in Table VIII, the baseline WER is 14.7%. The relevance of the experiments using the TRBNK3-gram rescored lattices is somewhat questionable since the lattice was generated using a much stronger language model—the LAT3-gram. Our point of view is the following: assume that we have a set of hypotheses that were produced in some way; we then rescore them using two language models, M1 and M2; if model M2 is truly superior to M1 (from a speech recognition perspective) then the WER obtained by rescoring the set of hypotheses using model M2 should be lower than that obtained using model M1.

We repeated the experiment in which we linearly interpolated the SLM with the 3-gram language model:

$$P(l) = \lambda \cdot P_{TRBNK3\text{-}gram}(l) + (1 - \lambda) \cdot P_{SLM}(l)$$

for different interpolation coefficients. This now constitutes a valid probability assignment. The A^* search parameters were the same as before. The results are presented in Table X.

The structured language model interpolated with the trigram model achieves 1.1% absolute (7.5% relative) WER reduction over the trigram baseline (see Table VIII); the parameters controlling the A^* search have not been tuned for this set of experiments and they were set to the same values as the ones reported in the previous section.

3.6.2. Switchboard lattice decoding results

On the SWB corpus, the lattices for which we ran decoding experiments were obtained using a language model (LAT3-gram) trained in very similar conditions—roughly the same training data size and vocabulary, closed over test data—to the ones under which the SLM and the baseline-deleted interpolation 3-gram model (TRBNK3-gram) were trained. The only difference is the tokenization (CSR vs. CSR-Treebank, see Section 4.5), which makes the LAT3-gram act as a phrase-based language model when compared to TRBNK3-gram. The experiments confirmed that LAT3-gram is stronger than TRBNK3-gram.

Again, we ran lattice rescoring experiments in two set-ups:

- using the language model that generates the lattice (LAT3-gram) as the baseline model; language model scores are available in the lattice.
- using the TRBNK3-gram language model (same training conditions as the SLM); we had to assign new language model scores to each link in the lattice.

Comparison between LAT3-gram and TRBNK3-gram. Again, we interpolate the two 3-gram models at our disposal (baseline results for the SWB experiments):

$$P(l) = \lambda \cdot P_{LAT3\text{-}gram}(l) + (1 - \lambda) \cdot P_{TRBNK3\text{-}gram}(l)$$

after correcting the tokenization mismatch as described in Section 3.6.1.

TABLE XI. 3-gram language model: Viterbi decoding results

λ	0.0	0.2	0.4	0.6	0.8	1.0
WER(%)	**42.3**	41.8	41.2	41.1	41.1	**41.3**

The results are shown in Table XI. Due to the difference in tokenization, only the extreme values in the table represent legitimate probability models over strings of words. The parameters in Equation (23) were set to: $LMweight = 12$, $\log P_{IP} = 10$.

LAT3-gram-driven search using the SLM. The previous experiments show that the LAT3-gram model is more powerful than the TRBNK3-gram model. We correct the interpolation

$$P(l) = \lambda \cdot P_{LAT3\text{-}gram}(l) + (1 - \lambda) \cdot P_{SLM}(l)$$

between the LAT3-gram model and the SLM as described in Section 3.6.1, and use it for lattice decoding.

The parameters controlling the search for the most likely parses in the SLM (see Section 2.3) were set to:
`maximum-stack-depth` $= 10$ and `LnP-threshold` $= 6.91$.

The A^* search parameters were tuned for the best interpolation value, $\lambda = 0.4$, using the N-best lattice sampling technique described in Section 3.3. The average rank of the hypothesis found by the A^* search among the N-best ones—after rescoring them using the structured language model interpolated with the trigram—was 0.3. There were 376 offending sentences (out of a total of 2427 sentences) in which the A^* search lead to a hypothesis whose score was lower than that of the top hypothesis among the N-best (0-best). In 346 cases, the prefix of the rescored 0-best was still in the stack when A^* returned (inadequate compensation), and in the other 30 cases, the 0-best hypothesis was lost during the search due to the finite stack size.

As a by-product, the WER performance of the structured language model on N-best list rescoring ($N = 25$) was 40.6%, showing that lattice rescoring was not more effective than N-best rescoring.

The parameters controlling the A^* search were set to: $\log P_{COMP} = 0.7$, $\log P_{FINAL} = 0$, $LMweight = 12$, $\log P_{IP} = 10$, `stack-depth-threshold=50`, `stack-depth-logP-threshold=100` [see Equations (23) and (24)]. The $\log P_{COMP}$, $\log P_{FINAL}$ and `stack-depth-threshold`, `stack-depth-logP-threshold` were optimized directly on test data for the best interpolation value found in the perplexity experiments. In all other experiments they were kept fixed to these values. The $LMweight$, $\log P_{IP}$ parameters are the ones typically used with the 3-gram model for the SWB task.

For the best interpolation value we have also experimented with the effect of SLM parameter re-estimation. The results for different interpolation coefficient values are shown in Table XII.

The structured language model achieved an absolute improvement of 0.7% WER over the baseline (see Table XI); the improvement is statistically significant at the 0.008 level according to a sign test at the sentence level. The 0.1% difference between the A^* and the Viterbi search results for a pure 3-gram model score ($\lambda = 1$) is due to the heuristic nature the A^* search: inadequate compensation $h(y|x)$ and finite stack. Note that this difference was 0 in the WSJ experiments.

TABLE XII. LAT3-gram + structured language model: A^* decoding results

Decoder	A^*						Viterbi
λ	0.0	0.2	0.4	0.6	0.8	1.0	1.0
WER(%) (E0 SLM)	42.0	40.8	40.7	40.6	40.7	41.2	**41.3**
WER(%) (E3 SLM)	41.8	40.7	40.7	40.7	40.7	41.2	**41.3**

TABLE XIII. TRBNK3-gram + structured language model: A^* decoding results

Decoder	A^*						Viterbi
λ	0.0	0.2	0.4	0.6	0.8	1.0	1.0
WER(%) (E0 SLM)	42.1	41.9	41.8	41.9	42.0	42.5	42.3
WER(%) (E3 SLM)	42.0	41.8	41.9	41.5	42.1	42.5	42.3

TRBNK3-gram-driven search using the SLM. We rescored each link in the lattice using the TRBNK3-gram language model, and used this as a baseline for further experiments. As shown in Table XI, the baseline WER is 42.3%.

We then repeated the experiment in which we linearly interpolate the SLM with the 3-gram language model:

$$P(l) = \lambda \cdot P_{TRBNK3\text{-}gram}(l) + (1 - \lambda) \cdot P_{SLM}(l)$$

for different interpolation coefficients. The A^* search parameters were the same as before. The results are presented in Table XIII.

The structured language model interpolated with the trigram model achieves 0.5% absolute reduction over the trigram baseline (see Table XI). The parameters controlling the A^* search have not been tuned for this set of experiments.

4. Practical considerations

This section describes experiments and details procedures that were considered to be too technical and to hinder the flow and clarity of explanation, but are nevertheless important for a comprehensive presentation of the model.

Subsections 4.1 through 4.4.1 detail experiments and choices we made regarding the structured language model; Subsection 4.5 gives a few more details on the experimental set-up used for lattice rescoring.

4.1. Choosing the model components parameterization

The experiments presented in Chelba (1997) show the usefulness of the two most recent exposed heads for word prediction. The same criterion—conditional perplexity—can be used as a guide in selecting the parameterization of each model component: WORD-PREDICTOR, TAGGER, CONSTRUCTOR. For each model component we gather the counts from the UPenn Treebank as explained in Section 2.6.1. The relative frequencies are determined from the "development" data, the interpolation weights estimated on "check" data (as described in Section 2.6.1). We then test each model component on counts gathered from the "test" data. Note that the smoothing scheme described in Section 2.2.2 discards elements of the context $\underline{z}$ from right to left.

TABLE XIV. WORD-PREDICTOR conditional perplexities

	Equivalence classification	Conditional PPL	Vocabulary size
HH	$\underline{z} = h_0.tag, h_0.word, h_{-1}.tag, h_{-1}.word$	115	10 000
WW	$\underline{z} = w_{-1}.tag, w_{-1}.word, w_{-2}.tag, w_{-2}.word$	156	10 000
hh	$\underline{z} = h_0.word, h_{-1}.word$	154	10 000
ww	$\underline{z} = w_{-1}.word, w_{-2}.word$	167	10 000

TABLE XV. TAGGER conditional perplexities

	Equivalence classification	Conditional PPL	Vocabulary size
HHw	$\underline{z} = w_k, h_0.tag, h_0.word, h_{-1}.tag, h_{-1}.word$	1.23	40
WWw	$\underline{z} = w_k, w_{-1}.tag, w_{-1}.word, w_{-2}.tag, w_{-2}.word$	1.24	40
ttw	$\underline{z} = w_k, h_0.tag, h_{-1}.tag$	1.24	40

4.1.1. Selecting the WORD-PREDICTOR equivalence classification

The experiments in Chelba (1997) were repeated using deleted interpolation as a modeling tool and the training/testing set-up described above. The results for different equivalence classifications of the word-parse k-prefix (W_k, T_k) are presented in Table XIV.

The different equivalence classifications of the word-parse k-prefix retain the following predictors:

1. ww: the two previous words (regular 3-gram model);
2. hh: the two most recent exposed headwords (no POS/NT label information);
3. WW: the two previous exposed words along with their POS tags;
4. HH: the two most recent exposed heads (headwords along with their NT/POS labels).

It can be seen that the most informative predictors for the next word are the exposed heads—the HH model. Except for the ww model (the regular 3-gram model) none of the others is a valid word-level perplexity, since it conditions the prediction on hidden information (namely the tags present in the treebank parses); the entropy of guessing the hidden information would need to be factored in.

4.1.2. Selecting the TAGGER equivalence classification

The results for different equivalence classifications of the word-parse k-prefix (W_k, T_k) for the TAGGER model are presented in Table XV.

The different equivalence classifications of the word-parse k-prefix retain the following predictors:

1. WWw: the two previous exposed words along with their POS tags and the word to be tagged;
2. HHw: the two most recent exposed heads (headwords along with their NT/POS labels) and the word to be tagged;
3. ttw: the NT/POS labels of the two most recent exposed heads and the word to be tagged.

It can be seen that among the equivalence classifications considered, none performs significantly better than the others, and the prediction of the POS tag for a given word is a relatively easy task—the conditional perplexities are very close to one. Because of its simplicity, we chose to work with the ttw equivalence classification.

TABLE XVI. CONSTRUCTOR conditional perplexities

	Equivalence classification	Conditional PPL	Vocabulary size
HH	$\underline{z} = h_0.tag, h_0.word, h_{-1}.tag, h_{-1}.word$	1.68	107
hhtt	$\underline{z} = h_0.tag, h_{-1}.tag, h_0.word, h_{-1}.word$	1.54	107
tt	$\underline{z} = h_0.tag, h_{-1}.tag$	1.71	107

4.1.3. Selecting the CONSTRUCTOR equivalence classification

The results for different equivalence classifications of the word-parse k-prefix (W_k, T_k) for the CONSTRUCTOR model are presented in Table XVI.

The different equivalence classifications of the word-parse k-prefix retain the following predictors:

1. HH: the two most recent exposed heads (headwords along with their NT/POS labels) and the word to be tagged;
2. hhtt: same as HH, except that the backing-off order is changed;
3. ttw: the NT/POS labels of the two most recent exposed heads.

It can be seen that the presence of headwords improves the accuracy of the CONSTRUCTOR component; also, the backing-off order of the predictors is important (hhtt vs. HH). We chose to work with the hhtt equivalence classification.

4.2. Fudged TAGGER and CONSTRUCTOR scores

The probability values for the three model components fall into different ranges. As pointed out at the beginning of Section 2.7, the WORD-PREDICTOR vocabulary is of the order of thousands, whereas the TAGGER and CONSTRUCTOR have vocabulary sizes of the order of tens. This leads to the undesirable effect that the contribution of the TAGGER and CONSTRUCTOR to the overall probability of a given partial parse $P(W, T)$ is very small compared to that of the WORD-PREDICTOR. We explored the idea of bringing the probability values into the same range by *fudging* the TAGGER and CONSTRUCTOR probability values, namely:

$$P(W, T) = \prod_{k=1}^{n+1} [P(w_k|W_{k-1}T_{k-1}) \cdot \{P(t_k|W_{k-1}T_{k-1}, w_k) \cdot P(T_{k-1}^k|W_{k-1}T_{k-1}, w_k, t_k)\}^\gamma]$$

$$P(T_{k-1}^k|W_{k-1}T_{k-1}) = \prod_{i=1}^{N_k} P(p_i^k|W_{k-1}T_{k-1}, w_k, t_k, p_1^k \ldots p_{i-1}^k), \tag{26}$$

where γ is the fudge factor. For $\gamma \neq 1.0$ we do not have a valid probability assignment anymore, since the sum over all possible word-sequences and parses is different than 1.0. However, the L2R-PPL calculated using Equation (11), Section 2.4 is still a valid word-level probability assignment due to the renormalization of the interpolation coefficients. Table XVII shows the PPL values calculated using Equation (11) where $P(W, T)$ is calculated using Equation (26). As can be seen, the optimal fudge factor turns out to be 1.0, corresponding to the *correct* calculation of the probability $P(W, T)$.

TABLE XVII. Perplexity values: Fudged TAGGER and CONSTRUCTOR

fudge	0.01	0.02	0.05	0.1	0.2	0.5	1.0	2.0	5.0	10.0	20.0	50.0	100.0
PPL	341	328	296	257	210	168	167	189	241	284	337	384	408

4.3. Pruning strategy

The pseudo-code for parsing a given input sentence is given in the algorithms in Figures 15–17.

4.3.1. Second pruning step

The pruning strategy described so far proved to be insufficient. Assuming that all stacks contain the maximum number of entries—equal to the stack-depth—the search effort grows squared with the sentence length. In order to approximately linearize the search effort with respect to sentence length, we chose to discard also the hypotheses whose score was more than a fixed log-probability relative threshold below the score of the topmost hypothesis in the current stack vector. This additional pruning step is performed after all hypotheses in stage k' have been extended with the `null` parser transition.

4.3.2. Cached TAGGER and CONSTRUCTOR lists

Another opportunity for speeding up the search is to have a cached list of possible POS tags/parser-operations in a given TAGGER/CONSTRUCTOR context. A good cacheing scheme should use an equivalence classification of the context that is specific enough to

```
current_stack_vector // set of stacks at current input position
future_stack_vector  // set of stacks at future  input position
hypothesis           // intial hypothesis
stack                // initial empty stack

// initialize algorithm
insert hypothesis in stack;
push stack at end of current_stack_vector;

// traverse input sentence
for each position in input sentence{
  PREDICTOR and TAGGER extension cycle;
  current_stack_vector = future_stack_vector;
  erase future_stack_vector;

  CONSTRUCTOR extension cycle;
  current_stack_vector = future_stack_vector;
  erase future_stack_vector;
}
// output the hypothesis with the highest score;
output max scoring hypothesis in current_stack_vector;
```

Figure 15. Pruning algorithm.

```
current_stack_vector // set of stacks at current input position
future_stack_vector  // set of stacks at future  input position
word                 // word at current input position
for each stack in current_stack_vector{
  // based on number of predictor and parser operations
  identify corresponding future_stack in future_stack_vector;

  for each hypothesis in stack{
    for all possible POS tag assignments for word{ //CACHE-ING
      expand hypothesis with word, POS tag;
      insert hypothesis in future_stack;
    }
  }
}
```

Figure 16. PREDICTOR and TAGGER extension algorithm.

```
current_stack_vector // set of stacks at current input position
future_stack_vector  // set of stacks at future  input position

// all possible parser transitions but the null-transition
for each stack in current_stack_vector, from bottom up{
  // based on number of parser operations
  identify corresponding future_stack in current_stack_vector;
  for each hypothesis in current_stack{                    // HARD
PRUNING
    for each parser_transition except the null-transition{ //CACHE-ING
      expand hypothesis with parser_transition;
      insert hypothesis in future_stack;
    }
  }
}
// null-transition moves us to the next position in the input
for each stack in current_stack_vector{
  // based on number of predictor and parser operations
  identify corresponding future_stack in future_stack_vector;
  for each hypothesis in current_stack{
    expand hypothesis with null-transition;
    insert hypothesis in future_stack;
  }
}
prune future_stack_vector                              //SECOND PRUNING
STEP
```

Figure 17. Parser extension algorithm.

actually reduce the list of possible options and general enough to apply in almost all the situations. For the TAGGER model we cache the list of POS tags for a given word seen in the training data and scan only those in the TAGGER extension cycle (see the algorithm in Fig. 16). For the CONSTRUCTOR model, we cache the list of parser operations seen in a given $(h_0.tag, h_{-1}.tag)$ context in the training data; parses that expose heads whose pair of NTtags has not been seen in the training data are discarded (see the algorithm in Fig. 17).

4.4. Word-level conditional probability

The correct formula for calculating the word-level conditional probability would be

$$P(w_{k+1}|W_k) = \sum_{T_k} P(w_{k+1}|W_k T_k) \cdot P(T_k|W_k). \tag{27}$$

This involves scanning all the partial parses T_k allowed by our model for a given word prefix W_k. Since the number of parses grows faster than exponential with the prefix length k, this is computationally intractable. A chart algorithm that takes advantage of the equivalence classifications (6)–(8) is described in Jelinek and Chelba (1999). Computational shortcuts are still needed since the complexity of the algorithm is proportional to k^6.

Our approach was to devise a search strategy that samples from among the most likely parses in the summation in Equation (27). The advantage of this procedure over the chart algorithm is that it is amenable to changing the equivalence classification in any of the model components without any impact on the complexity of the algorithm. On the other hand, for a given set of equivalence classifications in the model components, a chart algorithm is more efficient and thus able to take into account more parses for a given prefix, allowing for a more accurate computation in Equation (27).

4.4.1. Computationally tractable word-level probability

Left-to-right perplexity. As explained previously, the probability assignment for the word at position $k + 1$ in the input sentence was made using:

$$P(w_{k+1}|W_k) = \sum_{T_k \in S_k} P(w_{k+1}|W_k T_k) \cdot \rho(W_k, T_k), \tag{28}$$

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

where S_k is the set of all parses present in our stacks at the current stage k. This leads to the following formula for evaluating the perplexity:

$$\text{L2R-PPL} = \exp\left(-1/N \sum_{i=1}^{N} \ln [P(w_i|W_{i-1})]\right). \tag{29}$$

Note that if we set $\rho(W_k, T_k) = \delta(T_k, T_k^*|W_k)$—0-entropy guess for the prefix of the parse T_k to equal that of the final best parse T_k^*—the two probability assignments (30) and (28) would be the same, giving us an indication of the perplexity achievable by our model when using a given pruning strategy.

Non-causal "perplexity". Attempting to calculate the conditional perplexity by assigning to a whole sentence the probability

$$P(W|T^*) = \prod_{k=0}^{n} P(w_{k+1}|W_k T_k^*), \tag{30}$$

where $T^* = \text{argmax}_T P(W, T)$—the search for T^* being carried out according to our pruning strategy—is not valid because it is not causal: when predicting w_{k+1} we would be using T^* which was determined by looking at the entire sentence. In order to have a valid perplexity calculation we would need to factor in the uncertainty of guessing the prefix of the final best parse T_k^* *before predicting* w_{k+1}, based solely on the word prefix W_k.

However, the perplexity value calculated using (30) is an indication of the lower bound for the achievable perplexity of our model; for the above search parameters and E0 model statistics this bound was 98, corresponding to a relative reduction of 40% over the perplexity of the 3-gram model. For comparison, the value when conditioning on the manual parses in the UPenn Treebank (which was used to get the E0 statistics) was 115, as shown in Table XIV. This shows that the parses found by our model are better predictors (under the exposed heads parameterization) than those in the treebank.

This suggests that a better parameterization in the SLM—one that reduces the entropy $H(\rho(T_k|W_k))$ of guessing the "good" parse given the word prefix—would lead to a better model. Indeed, as we have already pointed out, the trigram model is a particular case of our model for which the parse is always right-branching, and we have no POS/NT tag information, leading to $H(\rho(T_k|W_k)) = 0$ and a standard 3-gram WORD-PREDICTOR. The 3-gram model is thus an extreme case of the structured language model: one for which the "hidden" structure is a *function of the word prefix*. Our result shows that better models can be obtained by allowing richer "hidden" structures (parses), and that a promising direction of research is to find the best compromise between the predictive power of the WORD-PREDICTOR—measured by $H(w_{k+1}|T_k, W_k)$—and the ease of guessing the most desirable hidden structure $T_k|W_k$—measured by $H(\rho(T_k|W_k))$—on which the WORD-PREDICTOR operation is based. The re-estimation procedure we have presented is one such method.

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

$$P(W) = \sum_{k=1}^{N} P(W, T^{(k)}), \tag{31}$$

where $T^{(k)}$ is one of the "N-best" (in the sense defined by our search) parses for an entire sentence W. This is a deficient probability assignment. However, it will turn out to be useful for justifying the model parameter re-estimation.

The two estimates (28) and (31) are both consistent in the sense that if the sums are carried out over all possible parses we get the correct value for the word-level perplexity of our model (27). This is important because the model parameter re-estimation procedure decreases the "sum-perplexity" (SUM-PPL)—calculated based on the probability assignment in Equation (31)—whereas we are interested in decreasing the "left-to-right-perplexity" (L2R-PPL)—calculated based on the probability assignment in Equation (28). As explained in Appendix 5.2, we rely on the consistency property of the two estimates to correlate the decrease in SUM-PPL with the desired decrease in L2R-PPL.

4.5. Experimental set-up for lattice rescoring

In order to train the SLM we use parse trees from which to initialize the parameters of the model. Fortunately a part of the SWB/WSJ data has been manually parsed at UPenn (Marcus *et al.*, 1995); let us refer to either of these corpora as the Treebank. The training data used for speech recognition (CSR) is different from the Treebank in two aspects:

TABLE XVIII. Treebank: CSR tokenization mismatch

Treebank	CSR
do n't	don't
it 's	it's
jones '	jones'
i 'm	i'm
i 'll	i'll
i 'd	i'd
we 've	we've
you 're	you're

- the Treebank data is a subset of the CSR data;
- the Treebank tokenization is different from that of the CSR corpus; among other spurious small differences, the most frequent ones are of the type presented in Table XVIII.

Our goal is to train the SLM on the CSR corpus.

Training set-up. The training of the SLM model proceeds as follows:

- Process the CSR training data to bring it closer to the Treebank format. We applied the transformations suggested by Table XVIII. The resulting corpus will be called CSR-Treebank, although at this stage we only have words and no parse trees for it.
- Transfer the syntactic knowledge from the Treebank onto the CSR-Treebank training corpus. As a result of this stage, CSR-Treebank is truly a "treebank" containing binarized and headword annotated trees:
  - — For the SWB experiments we parsed the SWB-CSR-Treebank corpus using the SLM trained on the SWB-Treebank, thus using the SLM as a parser. The vocabulary for this step was the union between the SWB-Treebank and the SWB-CSR-Treebank closed vocabularies. The resulting trees are already binary and have headword annotation.
  - — For the WSJ experiments we parsed the WSJ-CSR-Treebank corpus using the Ratnaparkhi maximum entropy parser (Ratnaparkhi, 1997), trained on the UPenn Treebank data. The resulting trees were binarized and annotated with headwords using the procedure described in Section 2.6.1. Note that the parser is mismatched, the most important difference being the fact that in the training data of the parser numbers are written as "$123", whereas in the data to be parsed they are expanded to "one hundred twenty three dollars"; we rely on the SLM parameter re-estimation procedure to smooth out this mismatch.
- Apply the SLM parameter re-estimation procedure to the CSR-Treebank training corpus using the parse trees obtained at the previous step for gathering initial statistics.

Notice that we have avoided "transferring" the syntactic knowledge from the Treebank tokenization directly onto the CSR tokenization. The reason is that CSR word tokens, like "he's" or "you're", cross boundaries of syntactic constituents in the Treebank corpus, and the transfer of parse trees from the Treebank to the CSR corpus is far from obvious and likely to violate syntactic knowledge present in the Treebank.

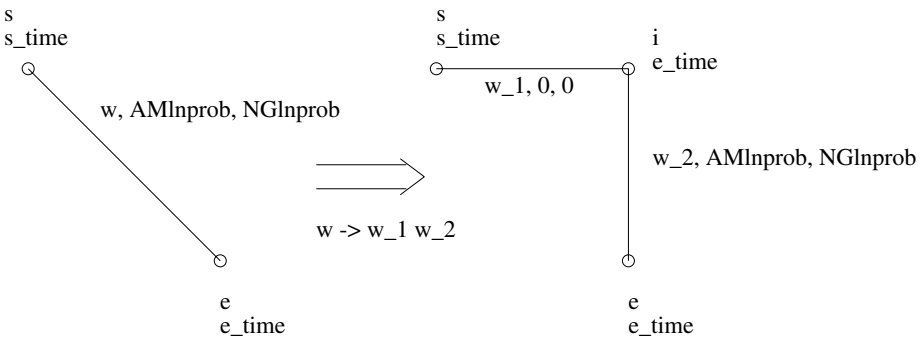

Figure 18. Lattice CSR to CSR-Treebank processing.

Lattice decoding set-up. To be able to run lattice decoding experiments we need to bring the lattices (in CSR tokenization) to the CSR-Treebank format. The only operation involved in this transformation is splitting certain words into two parts, as suggested by Table XVIII. Each link, whose word needs to be split, is cut into two parts and an intermediate node is inserted into the lattice as shown in Figure 18. The acoustic and language model scores of the initial link are copied onto the second new link.

For all the decoding experiments we have carried out, the WER is measured after undoing the transformations highlighted above. The reference transcriptions for the test data were not touched and the NIST SCLITE[6] package was used for measuring the WER.

5. Conclusions and future work

5.1. Comments on using the SLM as a parser

The structured language model could be used as a parser, namely by selecting the most likely parse according to our pruning strategy: $T^* = \text{argmax}_T P(W, T)$. Due to the fact that the SLM allows parses in which the words in a sentence are not joined under a single root node (see the definition of a complete parse and Fig. 3), a direct evaluation of the parse quality against the UPenn Treebank parses is unfair. However, a simple modification will constrain the parses generated by the SLM to join all words in a sentence under a single root node.

Imposing the additional constraint that:

- $P(w_k$=</s>$|W_{k-1}T_{k-1}) = 0 \; if \; h_{-1}.tag \neq SB$ ensures that the end of sentence symbol </s> is generated only from a parse in which all the words have been joined in a single constituent.

One important observation is that in this case one has to eliminate the second pruning step in the model and the hard pruning in the cache-ing of the CONSTRUCTOR model actions. It is sufficient if this is done only when operating on the last stack vector before predicting the end of sentence </s>. Otherwise, the parses that have all the words joined under a single root node may not be present in stacks before the prediction of the </s> symbol, resulting in a failure to parse a given sentence.

[6]SCLITE is a standard program supplied by NIST for scoring speech recognizers.

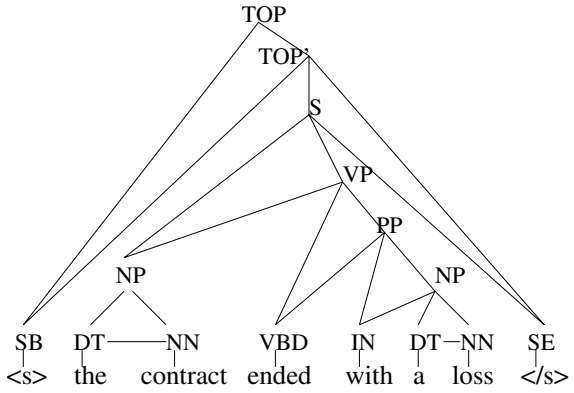

Figure 19. CFG dependencies.

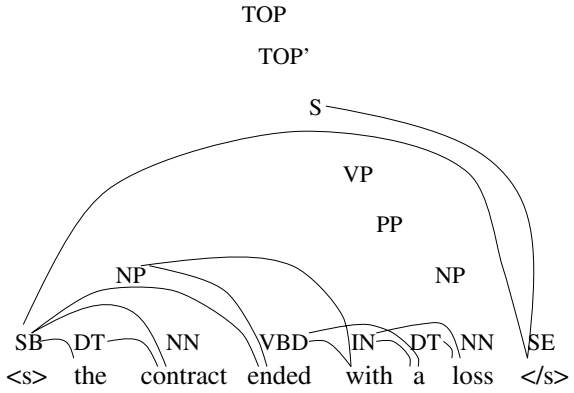

Figure 20. Tag-reduced WORD-PREDICTOR dependencies.

5.2. Comparison with other approaches

Underlying $P(W, T)$ probability model. The actions taken by the model are very similar to a LR parser. However, the encoding of the word sequence along with a parse tree (W, T) is different, proceeding bottom-up and interleaving the word predictions. This leads to a different probability assignment than that in a PCFG grammar, which is based on a different encoding of (W, T). Regarding (W, T) as a graph, Figure 19 shows the dependencies in a regular CFG. In contrast, Figures 20–22 show the probabilistic dependencies for each model component in the SLM; a complete dependency structure is obtained by super-imposing the three figures. To make the SLM directly comparable with a CFG, we discard the lexical information at intermediate nodes in the tree (headword annotation), thus assuming the following equivalence classifications in the model components [see Equations (6)–(8)]:

$$P(w_k|W_{k-1}T_{k-1}) = P(w_k|[W_{k-1}T_{k-1}]) = P(w_k|h_0.tag, h_{-1}.tag) \quad (32)$$

$$P(t_k|w_k, W_{k-1}T_{k-1}) = P(t_k|w_k, [W_{k-1}T_{k-1}]) = P(t_k|w_k, h_0.tag, h_{-1}.tag) \quad (33)$$

$$P(p_i^k|W_kT_k) = P(p_i^k|[W_kT_k]) = P(p_i^k|h_0.tag, h_{-1}.tag). \quad (34)$$

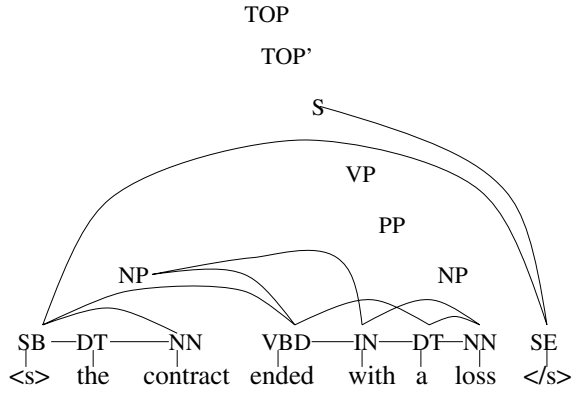

Figure 21. TAGGER dependencies.

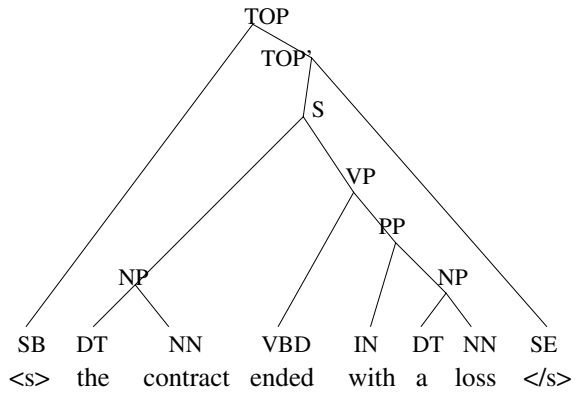

Figure 22. Tag-reduced CONSTRUCTOR dependencies.

It can be seen that the probabilistic dependency structure is more complex than that in a CFG even in this simplified SLM.

Along the same lines, the approach in Grenadier, Mark and Miller (1996) regards the word sequence W with the parse structure T as a Markov graph (W, T) modeled using the CFG dependencies superimposed on the regular word-level 2-gram dependencies, showing improvement in perplexity over both 2- and 3-gram modeling techniques. We believe our model is more constrained than the former as well.

As a formal probabilistic grammar, the SLM pertains to the class of shift-reduce probabilistic push-down automata. A thorough comparison between the two related classes of probabilistic languages (PCFGs and shift-reduce probabilistic push-down automata) has been presented in Abney *et al.* (1999).

Related work in statistical parsing of natural language includes the early work in Jelinek, Lafferty, Magerman, Mercer, Ratnaparkhi and Roukos (1994) and that of Ratnaparkhi (1997) with which we share many similarities. Generative models of word sequences along with parses are presented in Collins (1996) and Charniak (1997). However, none of these approaches were evaluated as a language model, with perplexity and word error rate reduction.

Language model. A structured approach to language modeling has been taken in S. Della Pietra, V. Della Pietra, Gillet Lafferty, Printz and Ures (1994). The underlying probability model $P(W, T)$ is a simple lexical link grammar, which is *automatically* induced and re-estimated using EM from a training corpus containing word sequences (sentences). The model does not make use of POS/NT labels, which we found extremely useful for word prediction and parsing. The impact of using POS/NT labels in the SLM may in part be due to better smoothing. Another constraint is placed on the context used by the WORD PREDICTOR: the two words in the context used for word prediction are always adjacent; our model's hierarchical scheme allows the exposed headwords to originate at any two different positions in the word prefix. Both approaches share the desirable property that the 3-gram model belongs to the parameter space of the model.

The language model we present is closely related to the one investigated in Chelba *et al.* (1997), but is, however, different in a few important aspects:

- our model operates in a left-to-right manner, thus allowing its use directly in the hypothesis search for $\hat{W}$ in (1);
- 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 that model;
- our model assigns probability at the word level, being a proper language model.

The SLM shares many features with both class-based language models (Brown, Della Pietra, deSouza, Lai & Mercer, 1997) and skip n-gram language models (Rosenfeld, 1994); an interesting approach combining class-based language models and different order skip-bigram models is presented in Saul and Pereira (1997). It seems worthwhile to make two comments relating the SLM to these approaches:

- The smoothing involving NT/POS tags in the WORD-PREDICTOR is similar to a class-based language model using NT/POS labels for classes. We depart, however, from the usual approach by *not making the conditional independence assumption* $P(w_{k+1}|w_k, \text{class}(w_k)) = P(w_{k+1}|\text{class}(w_k))$. Also, in our model the "class" assignment—through the heads exposed by a given parse T_k for the word prefix W_k and its "weight" $\rho(W_k, T_k)$, see Equation (11)—is highly context-sensitive, as it depends on the entire word-prefix W_k, and is syntactically motivated through the operations of the CONSTRUCTOR. A comparison between the hh and HH equivalence classifications in the WORD-PREDICTOR (see Table XIV) shows the usefulness of POS/NT labels for word prediction.
- recalling the depth factorization of the model in Equation (16), our model can be viewed as a skip n-gram, where the probability of a skip $P(d_0, d_1|W_k)$—d_0, d_1 are the depths at which the two most recent exposed headwords h_0, h_1 can be found, similar to $P(d|W_k)$—is highly context sensitive. Notice that the hierarchical scheme for organizing the word prefix allows for contexts that do not necessarily consist of adjacent words, as in regular skip n-gram models.

Future directions. We have presented an original approach to language modeling that makes use of syntactic structure. The experiments we have carried out show improvement in both perplexity and word error rate over current state-of-the-art techniques. Preliminary experiments reported in Wu and Khudanpur (1999) show complementarity between the SLM

and a topic language model yielding almost additive results—word error rate improvement—on the SWB task. Among the directions that we consider worth exploring in the future, are:

- automatic induction of the SLM initial parameter values;
- better integration of the 3-gram model and the SLM;
- better parameterization of the model components;
- to study the interaction between SLM and other language modeling techniques, such as cache and trigger or topic language models.

The authors would like to thank to Sanjeev Khudanpur, Harry Printz, Eric Ristad, Andreas Stolcke, Dekai Wu, and all the other members of the dependency modeling group at the summer'96 DoD Workshop for useful comments on the model, programming support, and an extremely creative environment. Also thanks to our STIMULATE colleagues Eric Brill, Sanjeev Khudanpur, David Yarowsky, Radu Florian, Lidia Mangu, and Jun Wu for useful input. Thanks to Shankar Kumar for running the experiments in Section 2.7.1. Also thanks to Bill Byrne for making available the Switchboard lattices, Vaibhava Goel for making available the N-best decoder, and Vaibhava Goel, Harriet Nock, and Murat Saraclar for useful discussions about lattice rescoring.

The work reported here was supported in part by NSF under NSF grant IRI-9618874 (STIMULATE). The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the funding agencies.

References

Abney, S., McAllester, D. & Pereira, F. (1999). Relating probabilistic grammars and automata. In *Proceedings of ACL*, College Park, MD, U.S.A., volume 1, pp. 541–549.

Bellegarda, J. R. (1997). A latent semantic analysis framework for large-span language modeling. In *Proceedings of Eurospeech 97*, Rhodes, Greece, pp. 1451–1454.

Berger, A. L., Della Pietra, S. A. & Della Pietra, V. J. (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, **22**, 39–72.

Brown, P., Della Pietra, V., deSouza, P., Lai, J. & Mercer, R. (1997). Class-based n-gram models of natural language. *Computational Linguistics*, **18**, 467–479.

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

Charniak, E. (1997). Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pp. 598–603. AAAI Press/MIT Press, Menlo Park, CA.

Chelba, C. (1997). A structured language model. In *Proceedings of ACL-EACL*, Madrid, Spain, Student section, pp. 498–500.

Chelba *et al.*, C. (1997). Structure and performance of a dependency language model. In *Proceedings of Eurospeech*, Rhodes, Greece, volume 5, pp. 2775–2778.

Chelba, C. & Jelinek, F. (1998). Exploiting syntactic structure for language modeling. In *Proceedings of COLING-ACL*, Montreal, Canada, volume 1, pp. 225–231.

CLSP, (1997). WS97. In *Proceedings of the 1997 CLSP/JHU Workshop on Innovative Techniques for Large Vocabulary Continuous Speech Recognition*, Baltimore, MD, July–August 1997.

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

Della Pietra, S., Della Pietra, V., Gillet, J., Lafferty, J., Printz, H. & Ures, L. (1994). Inference and estimation of a long-range trigram model. Technical Report CMU-CS-94-188, School of Computer Science, Carnegie Mellon University, Pittsburg.

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*, **39B**, 1–38.

Doug, B. Paul & Janet, M. Baker (1992). The design for the Wall Street Journal-based CSR corpus. In *Proceedings of the DARPA SLS Workshop*.

Godfrey, J. J., Holliman, E. C. & McDaniel, J. (1992). SWITCHBOARD telephone speech corpus for research and development. In *Proceedings of IEEE Conference on Acoustics, Speech and Signal Processing*, San Francisco, volume 1, pp. 517–520.

Grenadier, U., Mark, K. E. & Miller, M. I. (1996). Constrained stochastic language models. In *Image Models (and their Speech Model Cousins)*, (Levinson, S. E. and Shepp, L., eds), pp. 131–137. Springer, New York.

Haegeman, L. (1994). *Introduction to Government and Binding Theory*, pp. 138–141. Blackwell, Oxford, UK and Cambridge, MA.

Jelinek, F. (1998). *Statistical Methods for Speech Recognition*, MIT Press, Cambridge, Massachusetts and London, England.

Jelinek, F. & Chelba, C. (1999). Putting language into language modeling. In *Proceedings of Eurospeech '99*, Budapest, Hungary, pp. 2179–2182.

Jelinek, F. & Mercer, R. (1980). Interpolated estimation of Markov source parameters from sparse data, In *Pattern Recognition in Practice*. (Gelsema, E. and Kanal, L., eds), pp. 381–397. North Holland, Amsterdam.

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

Jelinek, F., Mercer, R. L., Bahl, L. R. & Baker, J. K. (1977). Perplexity—a measure of difficulty of speech recognition tasks. *Journal of the Acoustic Society of America*, **62,** S63, Supplement 1.

Jelinek, F. & Mercer, R. L. (1980). *Proceedings of the Workshop on Pattern Recognition in Practice*, 381–397. North Holland, Amsterdam,, deleted interpolation for n-gram models.

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

Ney, H., Aubert, X., Dugast, C. & Steinbiss, V. (1994). Large vocabulary continuous speech recognition of Wall Street Journal data. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing'94*, Adelaide, Australia, volume 2, pp. 129–132.

Nilsson, N. (1971). *Problem Solving Methods in Artificial Intelligence*, pp. 266–278. McGraw-Hill, New York.

Ratnaparkhi, A. (1997). A linear observed time statistical parser based on maximum entropy models. In *Second Conference on Empirical Methods in Natural Language Processing*, Providence, RI, U.S.A., pp. 1–10.

Rosenfeld, R. (1994). *Adaptive statistical language modeling: a maximum entropy approach.* PhD Thesis. School of Computer Science, Carnegie Mellon University, Pittsburgh.

Saul, L. & Pereira, F. (1997). Aggregate and mixed-order markov models for statistical language processing. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, San Francisco, CA, U.S.A., pp. 81–89.

Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymmetrically optimum decoding algorithm. *IEEE Transactions on Information Theory*, **IT-13**, 260–267.

Wu, J. & Khudanpur, S. (1999). Combining nonlocal, syntactic and *n*-gram dependencies in language modeling. In *Proceedings of Eurospeech'99, Budapest, Hungary*, pp. 1567–1570.

(Received 13 December 1999 and accepted for publication 26 April 2000)

Appendix A. Derivation of N-best re-estimation formulas

Let (W, T) denote the joint sequence of W with parse structure T—headword and POS/NT tag annotation included. As described in Section 2.1, (W, T) was produced by a unique sequence of model actions: WORD-PREDICTOR, TAGGER, and CONSTRUCTOR moves. The ordered collection of these moves will be called a *derivation*:

$$d(W, T) \doteq (e_1, \ldots, e_l),$$

where each *elementary event*

$$e_i \doteq (u^{(m)}|\underline{z}^{(m)})$$

identifies a model component action.

- m denotes the model component that took the action, $m \in$ {WORD-PREDICTOR, TAGGER, CONSTRUCTOR };

- u is the action taken:
  - — u is a word for $m =$ WORD-PREDICTOR;
  - — u is a POS tag for $m =$ TAGGER;
  - — $u \in$ {(adjoin-left, NTtag), (adjoin-right, NTtag), null} for $m =$ CONSTRUCTOR;
- $\underline{z}$ is the context in which the action is taken [see Equations (6)–(8)]:
  - — $\underline{z} = h_0.tag, h_0.word, h_{-1}.tag, h_{-1}.word$ for $m =$ WORD-PREDICTOR;
  - — $\underline{z} = w, h_0.tag, h_{-1}.tag$ for $m =$ TAGGER;
  - — $\underline{z} = h_0.tag, h_{-1}.tag, h_0.word, h_{-1}.word$ for $m =$ CONSTRUCTOR.

For each given (W, T) that satisfies the requirements in Section 2.1 there is a unique derivation $d(W, T)$. The converse is not true, namely that not every derivation corresponds to a correct (W, T). However, the constraints in Section 2.2 ensure that the illegal derivations receive 0 probability.

The probability of a (W, T) sequence is obtained by chaining the probabilities of the elementary events in its derivation, as described in Section 2.2:

$$P(W, T) = P(d(W, T)) = \prod_{i=1}^{length(d(W,T))} p(e_i). \tag{A1}$$

The probability of an elementary event is calculated using the smoothing technique presented in Section 2.2.2 and repeated here for clarity of explanation:

$$P_n(u|z_1, \ldots, z_n) = \lambda(z_1, \ldots, z_n) \cdot P_{n-1}(u|z_1, \ldots, z_{n-1}) + (1 - \lambda(z_1, \ldots, z_n)) \cdot f_n(u|z_1, \ldots, z_n), \tag{A2}$$

$$P_{-1}(u) = uniform(\mathcal{U}) \tag{A3}$$

- $z_1, \ldots, z_n$ is the context of order n when predicting u; $\mathcal{U}$ is the vocabulary in which u takes values;
- $f_k(u|z_1, \ldots, z_k)$ is the order-k relative frequency estimate for the conditional probability $P(u|z_1, \ldots, z_k)$:

$$f_k(u|z_1, \ldots, z_k) = C(u, z_1, \ldots, z_k)/C(z_1, \ldots, z_k), k = 0 \ldots n,$$

$$C(u, z_1, \ldots, z_k) = \sum_{z_{k+1} \in \mathcal{Z}} \ldots \sum_{z_n \in \mathcal{Z}} C(u, z_1, \ldots, z_k, z_{k+1} \ldots z_n),$$

$$C(z_1, \ldots, z_k) = \sum_{u \in \mathcal{U}} C(u, z_1, \ldots, z_k);$$

- $\lambda(z_1, \ldots, z_k)$ are the interpolation coefficients satisfying $0 < \lambda(z_1, \ldots, z_k) < 1, k = 0 \ldots n$.

The $\lambda(z_1, \ldots, z_k)$ coefficients are grouped into equivalence classes ("tied") based on the range into which the count $C(z_1, \ldots, z_k)$ falls, and the count ranges for each equivalence class are set such that a statistically sufficient number of events $(u|z_1, \ldots, z_k)$ fall within that range.

The parameters of a given model component m are:

- the maximal order counts $C^{(m)}(u, z_1, \ldots, z_n)$;

- the count ranges for grouping the interpolation values into equivalence classes—"tying";
- the interpolation value for each equivalence class.

The parameters θ being re-estimated are the maximal order joint counts $C^{(m)}(u^{(m)}, \underline{z}^{(m)})$ for each model component $m \in \{\text{WORD-PREDICTOR, TAGGER, CONSTRUCTOR}\}$; the other parameters are fixed to their initial values.

The E-step is carried by sampling the space of hidden events for a given seen sequence W according to the pruning strategy outlined in Section 2.3:

$$P_\theta^s(T|W) \doteq \frac{P_\theta(T, W)}{\sum_{T \in s_\theta(W)} P_\theta(W, T)} \cdot 1_{s_\theta(W)}(T),$$

where $1_{s_\theta(W)}(T)$ is the indicator function of the set of parses T for a given sentence W that survived the pruning strategy s_θ.

The logarithm of the probability of a given derivation can be calculated as follows:

$$\begin{aligned}
\log P_\theta(W, T) &= \sum_{i=1}^{length(d(W,T))} \log P_\theta(e_i) \\
&= \sum_m \sum_{(u^{(m)}, \underline{z}^{(m)})} \sum_{i=1}^{length(d(W,T))} \log P_\theta(u^{(m)}, \underline{z}^{(m)}) \cdot \delta(e_i, (u^{(m)}, \underline{z}^{(m)})) \\
&= \sum_m \sum_{(u^{(m)}, \underline{z}^{(m)})} \left[\sum_{i=1}^{length(d(W,T))} \delta(e_i, (u^{(m)}, \underline{z}^{(m)})) \right] \cdot \log P_\theta(u^{(m)}, \underline{z}^{(m)}) \\
&= \sum_m \sum_{(u^{(m)}, \underline{z}^{(m)})} \#\left[(u^{(m)}, \underline{z}^{(m)}) \in d(W, T)\right] \cdot \log P_\theta(u^{(m)}, \underline{z}^{(m)}),
\end{aligned}$$

where the random variable

$$\#[(u^{(m)}, \underline{z}^{(m)}) \in d(W, T)]$$

denotes the number of occurrences of the $(u^{(m)}, \underline{z}^{(m)})$ event in the derivation of W, T.

Let

$$\begin{aligned}
E_{P_{\theta_i}^s(T|W)}[\#[(u^{(m)}, \underline{z}^{(m)}) \in d(W, T)]] &\doteq a_{\theta_i}((u^{(m)}, \underline{z}^{(m)}), W) \\
\sum_{W \in \mathcal{T}} f(W) \cdot a_{\theta_i}((u^{(m)}, \underline{z}^{(m)}), W) &\doteq a_{\theta_i}(u^{(m)}, \underline{z}^{(m)}).
\end{aligned}$$

We then have:

$$\begin{aligned}
&E_{P_{\theta_i}^s(T|W)}[\log P_\theta(W, T)] \\
&= \sum_m \sum_{(u^{(m)}, \underline{z}^{(m)})} a_{\theta_i}((u^{(m)}, \underline{z}^{(m)}), W) \cdot \log P_\theta(u^{(m)}, \underline{z}^{(m)}),
\end{aligned}$$

and the EM auxiliary function becomes

$$\sum_{W \in \mathcal{T}} f(W) \cdot E_{P_{\theta_i}^s(T|W)}[\log P_\theta(W, T)] \tag{A4}$$

$$= \sum_m \sum_{(u^{(m)}, \underline{z}^{(m)})} a_{\theta_i}(u^{(m)}, \underline{z}^{(m)}) \cdot \log P_\theta(u^{(m)}, \underline{z}^{(m)}). \tag{A5}$$

The E-step thus consists of the calculation of the expected values $a_{\theta_i}((u^{(m)}, \underline{z}^{(m)}))$, for every model component and every event $(u^{(m)}, \underline{z}^{(m)})$ in the derivations that survived the pruning process.

In the M-step we need to find a new parameter value θ_{i+1} such that we maximize the EM auxiliary function (A4):

$$\theta_{i+1} = \arg\max_{\theta\in\theta} \sum_{W\in\mathcal{T}} f(W) \cdot E_{P^s_{\theta_i}(T|W)}[\log P_\theta(W, T)] \tag{A6}$$

$$= \arg\max_{\theta\in\theta} \sum_{m} \sum_{(u^{(m)}, \underline{z}^{(m)})} a_{\theta_i}(u^{(m)}, \underline{z}^{(m)}) \cdot \log P_\theta(u^{(m)}, \underline{z}^{(m)}). \tag{A7}$$

One can easily notice that the M-step is in fact a problem of maximum likelihood estimation for each model component m from joint counts $a_{\theta_i}(u^{(m)}, \underline{z}^{(m)})$. Taking into account the parameterization of $P_\theta(u^{(m)}, \underline{z}^{(m)})$ (see Section 2.2.2) the problem can be seen as an HMM re-estimation problem. The EM algorithm can be employed to solve it. Convergence takes place in exactly one EM iteration to

$$C^{(m)}_{i+1}(u^{(m)}, \underline{z}^{(m)}) = a_{\theta_i}(u^{(m)}, \underline{z}^{(m)}).$$

We wish to emphasize that due to both the smoothing involved in the M-step (imposed by the smooth parameterization of the $P(W, T)$ model) and the fact that the set of sampled "N-best" hidden events (parses) are re-evaluated at each iteration, we allow new maximal order events to appear in each model component while discarding others.

Appendix B. Comments and experiments on model parameters re-estimation

As outlined in Section 2.4, the word-level probability assigned to a training/test set by our model is calculated using the proper word-level probability assignment in Equation (11). An alternative that leads to a deficient probability model is to sum over all the complete parses that survived the pruning strategy, formalized in Equation (13). Let the likelihood assigned to a corpus $\mathcal{C}$ by our model P_θ be denoted by:

- $\mathcal{L}^{L2R}(\mathcal{C}, P_\theta)$, where P_θ is calculated using (11), repeated here for clarity:

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

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

  Note that this is a proper probability model.
- $\mathcal{L}^{N}(\mathcal{C}, P_\theta)$, where P_θ is calculated using (13):

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

  This is a deficient probability model.

One seeks to devise an algorithm that finds the model parameter values that maximize the likelihood of a test corpus. This is an unsolved problem; the standard approach is to resort to maximum likelihood estimation techniques on the training corpus and make provisions that will ensure that the increase in likelihood on training data carries over to unseen test data.

As outlined previously, the estimation procedure of the SLM parameters takes place in two stages:

TABLE XIX. Evolution of different "perplexity" values during training

"Perplexity"	Iteration		Relative change
	E0	E3	
TOP-PPL	97.5	99.3	+1.85%
BOT-PPL	107.9	106.2	−1.58%
SUM-PPL	195.1	175.5	−10.05%
L2R-PPL	167.5	158.3	−5.49%

1. The "N-best training" algorithm (see Section 2.6.2) is employed to increase the training data "likelihood" $\mathcal{L}^N(\mathcal{C}, P_\theta)$. The initial parameters for this first estimation stage are gathered from a treebank. The perplexity is still evaluated using the formula in Equation (11).
2. Estimate a separate L2R-WORD-PREDICTOR model such that $\mathcal{L}^{L2R}(\mathcal{C}, P_\theta)$ is increased [see Equation (14)]. The initial parameters for the L2R-WORD-PREDICTOR component are obtained by copying the WORD-PREDICTOR estimated at stage one.

The "N-best training" algorithm is employed to increase the training data "likelihood" $\mathcal{L}^N(\mathcal{C}, P_\theta)$. We rely on the consistency of the probability estimates underlying the calculation of the two different likelihoods to correlate the increase in $\mathcal{L}^N(\mathcal{C}, P_\theta)$ with the desired increase of $\mathcal{L}^{L2R}(\mathcal{C}, P_\theta)$.

To be more specific, $\mathcal{L}^N(\mathcal{C}, P_\theta)$ and $\mathcal{L}^{L2R}(\mathcal{C}, P_\theta)$ are calculated using the probability assignments in Equation (31) (deficient) and Equation (28), respectively. Both probability estimates are consistent in the sense that if we summed over all the parses T for a given word sequence W they would yield the correct probability $P(W)$ according to our model. Although there is no formal proof, there are reasons to believe that the N-best re-estimation procedure should not decrease the $\mathcal{L}^N(\mathcal{C}, P_\theta)$ likelihood,[7] but no claim can be made about the increase in the $\mathcal{L}^{L2R}(\mathcal{C}, P_\theta)$ likelihood, which is the one we are interested in. Our experiments show that the increase in $\mathcal{L}^N(\mathcal{C}, P_\theta)$ is correlated with an increase in $\mathcal{L}^{L2R}(\mathcal{C}, P_\theta)$, a key factor in this being a good heuristic search strategy (see Sections 2.3 and 4.3). Table XIX shows the evolution of different "perplexity" values during N-best re-estimation. L2R-PPL is calculated using the proper probability assignment in Equation (28). TOP-PPL and BOT-PPL are calculated using the probability assignment in Equation (30), where $T^* = \text{argmax}_T P(W, T)$ and $T^* = \text{argmin}_T P(W, T)$, respectively—the search for T^* being carried out according to our pruning strategy (we condition the word predictions on the top-most and bottom-most parses present in the stacks after parsing the entire sentence). SUM-PPL is calculated using the deficient probability assignment in Equation (31). It can be noticed that TOP-PPL and BOT-PPL stay almost constant during the re-estimation process; the value of TOP-PPL is slightly increasing and that of BOT-PPL is slightly decreasing. As expected, the value of the SUM-PPL decreases, and its decrease is correlated with that of the L2R-PPL.

It is very important to note that due to both the smoothing involved in the M-step (imposed by the smooth parameterization of the model) and the fact that the set of sampled "N-best" hidden events (parses) are re-evaluated at each iteration, we allow new maximal order events to appear in each model component while discarding others. Unlike standard parameterizations, we do not re-estimate the relative frequencies from which each component probabilistic model is derived; that would lead to a shrinking or, at best, fixed set of events. *Not only are we estimating the counts of maximal order n-gram events in each model component* (WORD-

[7] It is very similar to a rigorous EM approach.

TABLE XX. Dynamics of WORD-PREDICTOR distribution on types during re-estimation

Iteration	No. tokens	No. types for order				
		0	1	2	3	4
E0	929 564	9976	77 225	286 329	418 843	591 505
E1	929 564	9976	77 115	305 266	479 107	708 135
E2	929 564	9976	76 911	305 305	482 503	717 033
E3	929 564	9976	76 797	307 100	490 687	731 527
L2R0 (=E3)	929 564	9976	76 797	307 100	490 687	731 527
L2R1-5	929 564	9976	257 137	2 075 103	3 772 058	5 577 709

PREDICTOR, TAGGER, CONSTRUCTOR), *but we also allow the distribution on types to change from one iteration to the other*. This is because the set of hidden events allowed for a given observed word sequence is not invariant. For example, the count set that describes the WORD-PREDICTOR component of the model to be used at the next iteration may have a different *n*-gram composition than that used at the current iteration.

We evaluated the change in the distribution on types[8] of the maximal order events ($y^{(m)}$, $\underline{x}^{(m)}$) from one iteration to the next. Table XX shows the dynamics of the set of types of the different order events during the re-estimation process for the WORD-PREDICTOR model component. Similar dynamics were observed for the other two components of the model. The equivalence classifications corresponding to each order are:

- $\underline{z} = h_0.tag, h_0.word, h_{-1}.tag, h_{-1}.word$ for order 4;
- $\underline{z} = h_0.tag, h_0.word, h_{-1}.tag$ for order 3;
- $\underline{z} = h_0.tag, h_0.word$ for order 2;
- $\underline{z} = h_0.tag$ for order 1.

An event of order 0 consists of the predicted word only.

The higher order events (closer to the root of the linear interpolation scheme in Fig. 9) become more and more diverse during the first estimation stage, as opposed to the lower order events. This shows that the "N-best" parses for a given sentence change from one iteration to the next. Although the E0 counts were collected from "1-best" parses (binarized treebank parses), the increase in number of maximal order types from E0 to E1 (collected from "N-best", N = 10) is far from dramatic, yet higher than that from E1 to E2, both collected from "N-best" parses.

The big increase in number of types from E3 (=L2R0) to L2R1 is due to the fact that at each position in the input sentence, WORD-PREDICTOR counts are now collected for all the parses in the stacks, many of which do not belong to the set of N-best parses for the *complete* sentence used for gathering counts during E0-3.

Although the perplexity on test data still decreases during the second re-estimation stage—we are not over-training—this decrease is very small and not worth the computational effort if the model is linearly interpolated with a 3-gram model, as shown in Table IV. Better integration of the 3-gram and the head predictors is desirable.

[8] A type is a particular value, regarded as one entry in the alphabet spanned by a given random variable.