STRUCTURED LANGUAGE MODELING FOR SPEECH RECOGNITION†

Ciprian Chelba and Frederick Jelinek

Abstract

A new language model for speech recognition is presented. The model develops hidden hierarchical syntactic-like structure incrementally and uses it to extract meaningful information from the word history, thus complementing the locality of currently used trigram models. The structured language model (SLM) and its performance in a two-pass speech recognizer — lattice decoding — are presented. Experiments on the WSJ corpus show an improvement in both perplexity (PPL) and word error rate (WER) over conventional trigram models.

1 Structured Language Model

An extensive presentation of the SLM can be found in [1]. The model assigns a probability $P(W,T)$ to every sentence W and its every possible binary parse T. The terminals of T are the words of W with POStags, and the nodes of T are annotated with phrase headwords and non-terminal labels.

Let W be a sentence of length n words to which we have prepended `<s>` and appended `</s>` so that w_0 =`<s>` and w_{n+1} =`</s>`. Let W_k be the word k-prefix $w_0 \dots w_k$ of the sentence and $W_k T_k$ the *word-parse k-prefix*. Figure 1 shows a word-parse k-prefix; `h_0 .. h_{-m}` are the *exposed heads*, each head being a pair(headword, non-terminal label), or (word, POStag) in the case of a root-only tree.

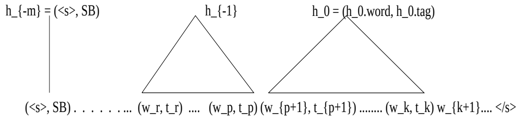

Figure 1: **A word-parse k-prefix**

1.1 Probabilistic Model

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

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

where:

- $W_{k-1}T_{k-1}$ is the word-parse $(k-1)$-prefix

†This work was funded by the NSF IRI-19618874 grant STIMULATE

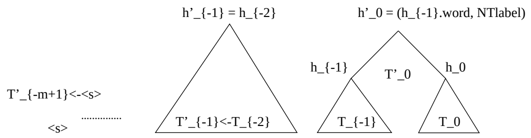

Figure 2: **Result of adjoin-left under NTlabel**

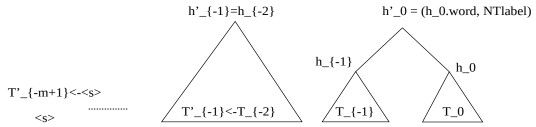

Figure 3: **Result of adjoin-right under NTlabel**

- w_k is the word predicted by WORD-PREDICTOR
- t_k is the tag assigned to w_k by the TAGGER
- $N_k - 1$ is the number of operations the PARSER executes at sentence position k before passing control to the WORD-PREDICTOR (the N_k-th operation at position k is the `null` transition); N_k is a function of T
- p_i^k denotes the i-th PARSER operation carried out at position k in the word string; the operations performed by the PARSER are illustrated in Figures 2-3 and they ensure that all possible binary branching parses with all possible headword and non-terminal label assignments for the $w_1 \dots w_k$ word sequence can be generated. Our model is based on three probabilities, estimated using deleted interpolation (see [3]), parameterized as follows:

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

$$P(t_k/w_k, W_{k-1}T_{k-1}) = P(t_k/w_k, h_0.tag, h_{-1}.tag) \quad (2)$$

$$P(p_i^k/W_kT_k) = P(p_i^k/h_0, h_{-1}) \quad (3)$$

It is worth noting that if the binary branching structure developed by the parser were always right-branching and we mapped the POStag and non-terminal label vocabularies to a single type then our model would be equivalent to a trigram language model.
Since the number of parses for a given word prefix W_k grows exponentially with k, $|\{T_k\}| \sim O(2^k)$, the state space of our model is huge even for relatively short sentences so we had to use a search strategy that prunes it. Our choice was a synchronous multi-stack search algorithm which is very similar to a beam search.
The probability assignment for the word at position $k+1$ in the input sentence is made using:

$$P(w_{k+1}/W_k) = \sum_{T_k \in S_k} P(w_{k+1}/W_kT_k) \cdot [\ P(W_kT_k) / \sum_{T_k \in S_k} P(W_kT_k)\ ] \quad (4)$$

which ensures a proper probability over strings W^*, where S_k is the set of all parses present in our stacks at the current stage k. An N-best EM variant is employed to reestimate the model parameters such that the PPL on training data is decreased — the likelihood of the training data under our model is increased. The reduction in PPL is shown experimentally to carry over to the test data.

2 A^* Decoder for Lattices

The *speech recognition lattice* is an intermediate format in which the hypotheses produced by the first pass recognizer are stored. For each utterance we save a *directed acyclic graph* in which the *nodes* are a subset of the language model states in the composite hidden Markov model and the arcs — *links* — are labeled with words. Typically, the first pass acoustic/language model scores associated with each link in the lattice are saved and the nodes contain time alignment information.

There are a couple of reasons that make A^* [4] appealing for lattice decoding using the SLM:
• the algorithm operates with whole prefixes, making it ideal for incorporating language models whose memory is the entire sentence prefix;
• a reasonably good lookahead function and an efficient way to calculate it using dynamic programming techniques are both readily available using the n-gram language model.

2.1 A^* Algorithm

Let a set of hypotheses $L = \{h : x_1, \ldots, x_n\}, \; x_i \in \mathcal{W}^* \; \forall \; i$ be organized as a prefix tree. We wish to obtain the maximum scoring hypothesis under the scoring function $f : \mathcal{W}^* \rightarrow \Re$: $h^* = \arg\max_{h \in L} f(h)$ without scoring all the hypotheses in L, if possible with a minimal computational effort. The A^* algorithm operates with prefixes and suffixes of hypotheses — paths — in the set L; we will denote prefixes — anchored at the root of the tree — with x and suffixes — anchored at a leaf — with y. A complete hypothesis h can be regarded as the concatenation of a x prefix and a y suffix: $h = x.y$.

To be able to pursue the most promising path, the algorithm needs to evaluate all the possible suffixes that are allowed in L for a given prefix $x = w_1, \ldots, w_p$ — see Figure 4. 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 $x.y$: $g(x.y) \doteq f(x) + h(y|x) \geq f(x.y)$. Imposing 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:

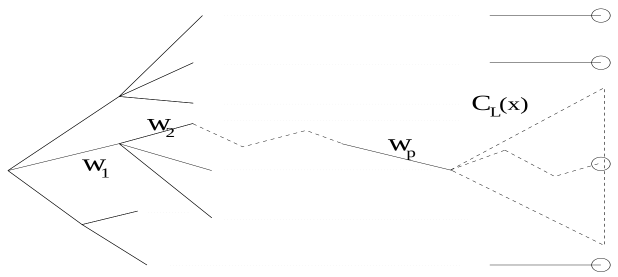

Figure 4: **Prefix Tree Organization of a Set of Hypotheses L**

$$g_L(x) \doteq \max_{y \in C_L(x)} g(x.y) = f(x) + h_L(x), \; where \quad (5)$$
$$h_L(x) \doteq \max_{y \in C_L(x)} h(y|x) \quad (6)$$

$g_L(x)$ is an overestimate for the $f(\cdot)$ score of any complete hypothesis that has the prefix x; the overestimate becomes exact for complete hypotheses. The A^* algorithm uses a potentially infinite stack in which prefixes x are ordered 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 a complete one, retain it as the overall best hypothesis h^*.

2.2 A^* Lattice Rescoring

A speech recognition 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 $p = l_0 \ldots l_n$ maximizing:

$$f(p) = \sum_{i=0}^{n} [logP_{AM}(l_i) + LMweight \cdot logP_{LM}(w(l_i)|w(l_0) \ldots w(l_{i-1})) - logP_{IP}] \quad (7)$$

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

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 the definition (5) of $g_L(\cdot)$ we need an overestimate $g(x.y) = f(x) + h(y|x) \geq f(x.y)$ for all possible $y = l_k \ldots l_n$ complete continuations of x allowed by the lattice. We propose to use the heuristic:

$$h(y|x) = \sum_{i=k}^{n} [logP_{AM}(l_i) + LMweight \cdot (logP_{NG}(l_i) + logP_{COMP}) - logP_{IP}] + LMweight \cdot logP_{FINAL} \cdot \delta(k < n) \quad (8)$$

A simple calculation shows that if $logP_{LM}(l_i)$ satisfies: $logP_{NG}(l_i) + logP_{COMP} \geq logP_{LM}(l_i), \forall l_i$ 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. In practice one cannot maintain a potentially infinite stack. The $logP_{COMP}$ and

$logP_{FINAL}$ parameters controlling the quality of the overstimate in (8) are adjusted empirically. A more detailed description of this procedure is precluded by the length limit on the article.

3 Experiments

As a first step we evaluated the perplexity performance of the SLM relative to that of a baseline deleted interpolation 3-gram model trained under the same conditions: training data size 5Mwds (section 89 of WSJ0), vocabulary size 65kwds, closed over test set. We have linearly interpolated the SLM with the 3-gram model: $P(\cdot) = \lambda \cdot P_{3gram}(\cdot) + (1 - \lambda) \cdot P_{SLM}(\cdot)$ showing a 16% relative reduction in perplexity; the interpolation weight was determined on a held-out set to be $\lambda = 0.4$. A second batch of experiments evaluated the performance of the SLM for

Trigram + SLM			
λ	0.0	0.4	1.0
PPL	116	109	130
Lattice Trigram + SLM			
WER	11.5	9.6	10.6

Table 1: **Test Set Perplexity and Word Error Rate Results**

trigram lattice decoding[1]. The results are presented in Table 1. The SLM achieved an absolute improvement in WER of 1% (10% relative) over the lattice 3-gram baseline; the improvement is statistically significant at the 0.0008 level according to a sign test. As a by-product, the WER performance of the structured language model on 10-best list rescoring was 9.9%.

4 Experiments: ERRATA

We repeated the WSJ lattice rescoring experiments reported in [2] in a standard setup. We chose to work on the DARPA'93 evaluation HUB1 test set — 213 utterances, 3446 words. The 20kwds open vocabulary and baseline 3-gram model are the standard ones provided by NIST.

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: training data size 20Mwds (a subset of the training data used for the baseline 3-gram model), standard HUB1 open vocabulary of size 20kwds; both the training data and the vocabulary were re-tokenized such that they conform to the Upenn Treebank tokenization. We have linearly interpolated the SLM with the above 3-gram model:

$$P(\cdot) = \lambda \cdot P_{3gram}(\cdot) + (1 - \lambda) \cdot P_{SLM}(\cdot)$$

showing a 10% relative reduction over the perplexity of the 3-gram model. The results are

[1] The lattices were generated using a language model trained on 45Mwds and using a 5kwds vocabulary closed over the test data.

presented in Table 2. The SLM parameter reestimation procedure[2] reduces the PPL by 5% (2% after interpolation with the 3-gram model). The main reduction in PPL comes however from the interpolation with the 3-gram model showing that although overlapping, the two models successfully complement each other. The interpolation weight was determined on a held-out set to be $\lambda = 0.4$. Both language models operate in the UPenn Treebank text tokenization.

Trigram(20Mwds) + SLM			
λ	0.0	0.4	1.0
PPL, initial SLM, iteration 0	152	136	148
PPL, reestimated SLM, iteration 1	144	133	148

Table 2: **Test Set Perplexity Results**

A second batch of experiments evaluated the performance of the SLM for 3-gram[3] lattice decoding. The lattices were generated using the standard baseline 3-gram language model trained on 40Mwds and using the standard 20kwds open vocabulary. The best achievable WER on these lattices was measured to be 3.3%, leaving a large margin for improvement over the 13.7% baseline WER.

For the lattice rescoring experiments we have adjusted the operation of the SLM such that it assigns probability to word sequences in the CSR tokenization and thus the interpolation between the SLM and the baseline 3-gram model becomes valid. The results are presented in Table 3. The SLM achieved an absolute improvement in WER of 0.7% (5% relative) over the baseline despite the fact that it used half the amount of training data used by the baseline 3-gram model. Training the SLM does not yield an improvement in WER when interpolating with the 3-gram model, although it improves the performance of the SLM by itself.

Lattice Trigram(40Mwds) + SLM			
λ	0.0	0.4	1.0
WER, initial SLM, iteration 0	14.4	13.0	13.7
WER, reestimated SLM, iteration 1	14.3	13.2	13.7

Table 3: **Test Set Word Error Rate Results**

5 Acknowledgements

The authors would like to thank to Sanjeev Khudanpur for his insightful suggestions. Also thanks to Bill Byrne for making available the WSJ lattices, Vaibhava Goel for making available

[2]Due to the fact that the parameter reestimation procedure for the SLM is computationally expensive we ran only a single iteration

[3]In the previous experiments reported on WSJ we have accidentally used bigram lattices

the N-best decoder, Adwait Ratnaparkhi for making available his maximum entropy parser, and Vaibhava Goel, Harriet Nock and Murat Saraclar for useful discussions about lattice rescoring. Special thanks to Michael Riley and Murat Saraclar for help in generating the WSJ lattices used in the revised experiments.

References

[1] C. CHELBA and F. JELINEK. Exploiting syntactic structure for language modeling. In *Proceedings of COLING-ACL*, volume 1, pages 225–231. Montreal, Canada, 1998.

[2] C. CHELBA and F. JELINEK. Structured language modeling for speech recognition. In *Proceedings of NLDB99*. Klagenfurt, Austria, 1999.

[3] F. JELINEK and R. MERCER. Interpolated estimation of markov source parameters from sparse data. In E. Gelsema and L. Kanal, editors, *Pattern Recognition in Practice*, pages 381–397. 1980.

[4] N. NILSSON. *Problem Solving Methods in Artificial Intelligence*, pages 266–278. McGraw-Hill, New York, 1971.