EXACT TRAINING OF A NEURAL SYNTACTIC LANGUAGE MODEL

Ahmad Emami and Frederick Jelinek

Center for Language and Speech Processing
Johns Hopkins University
Baltimore, MD 21218
$\{$*emami, jelinek*$\}$*@jhu.edu*

ABSTRACT

The Structured Language Model aims at making a prediction of the next word in a given word string by making a syntactical analysis of the preceding words. However, it faces the data sparseness problem because of the large dimensionality and diversity of the information available in the syntactic parses. In previous work [1, 2], we proposed using neural network models for the SLM. The neural network model is better suited to tackle the data sparseness problem and its use gave significant improvements in perplexity and word error rate over the baseline SLM.

In this paper we present a new method of training the neural net based SLM. The presented procedure makes use of the partial parses hypothesized by the SLM itself, and is more expensive than the approximate training method used in previous work.

Experiments with the new training method on the UPenn and WSJ corpora show significant reductions in perplexity and word error rate, achieving the lowest published results for the given corpora.

1. INTRODUCTION

The role of a statistical language model is to assign a probability $P(W)$ to any given word string $W = w_1 w_2 \ldots w_n$. This is usually done in a left-to-right manner by factoring the probability:

$$P(W)=P(w_1 w_2 \ldots w_n)=P(w_1)\prod_{i=2}^{n} P(w_i|W_1^{i-1})$$

where the sequence of words $w_1 w_2 \ldots w_j$ is denoted by W_1^j. Ideally, the language model should use the entire history W_1^{i-1} to make its prediction for word w_i. However, because of data sparseness some equivalence classification of histories W_1^{i-1} should be employed. The popular N-gram models classify the word string W_1^{i-1} into W_{i-N+1}^{i-1} and perform surprisingly well given their simple structure. Nevertheless, they lack the ability to use longer histories (locality problem), and still suffer from severe data sparseness even for small values of N.

The *Structured Language Model* (SLM) aims at overcoming the locality problem by constructing syntactical parses of a word string and using the information from these partial parses to predict the next word [3]. In this manner, the SLM also addresses one other problem of the N-gram models: the use of surface (lexical) words only; by using information from the deeper syntactic structures of the word strings.

Distributed representation of variables, combined with a neural network for probability estimation, has enabled the use of longer probabilistic dependencies [4]. The SLM uses N-gram type dependencies for its internal components, and it would be desirable to implement them by neural networks that can use longer and richer dependencies. In fact, the use of neural network models in the SLM has lead to significant reductions in both perplexity and word error rate [1, 2].

In this paper we investigate further the use of a neural net model as the component of the SLM responsible for predicting the next word based on the partial parses of the preceding word string. In previous work, the neural net component was trained on parses extracted from an external treebank (an external source) [1]. We present here an exact training procedure, which optimizes the proper likelihood function computed using the partial parses hypothesized by the SLM itself. The new training method involves using *multiple* partial parses at each position, which results in a different objective function, and hence a different training algorithm for the neural network.

Section 2 serves as an introduction to the SLM. In Section 3 we describe the neural network model and explain how it is used in the SLM, giving the details of the update algorithm for training on multiple partial parses. Experimental results are presented in Section 4.

2. STRUCTURED LANGUAGE MODEL

An extensive presentation of the SLM can be found in [3]. The model assigns a probability $P(W,T)$ to every sentence W and every possible binary parse T of W. The terminals of T are the words of W with POS tags, and the nodes of T are annotated with phrase headwords and non-terminal labels.

Let W be a sentence of n words to which we have prepended the sentence beginning marker `<s>` and appended the sentence end marker `</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 — the words from the beginning of the sentence up to the current position k— and $W_k T_k$ the *word-parse k-prefix*. Figure 1(a) shows a word-parse k-prefix where `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. For a given word-parse k-prefix the headwords are percolated bottom-up in a rule-based non-probabilistic procedure.

2.1. Probabilistic Model

The operation of the SLM is characterized by the finite state machine in Figure 1(b). The joint probability $P(W,T)$ of a word

This work was supported by the National Science Foundation under grant No. IIS-0085940.

sequence W and a complete parse T can be expressed as:

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

where:
• $W_{k-1}T_{k-1}$ is the word-parse $(k-1)$-prefix
• w_k is the word predicted by PREDICTOR
• t_k is the POS tag assigned to w_k by the TAGGER
• $N_k - 1$ is the number of operations the CONSTRUCTOR executes at sentence position k before passing control to the PREDICTOR (the $N_k - th$ operation At position k is the `null` transition);
• p_i^k denotes the $i - th$ CONSTRUCTOR operation carried out at position k in the word string; the operations performed by the CONSTRUCTOR 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. The $p_1^k \dots p_{N_k}^k$ sequence of CONSTRUCTOR operations at position k grows the word-parse $(k-1)$-prefix into a word-parse k-prefix.

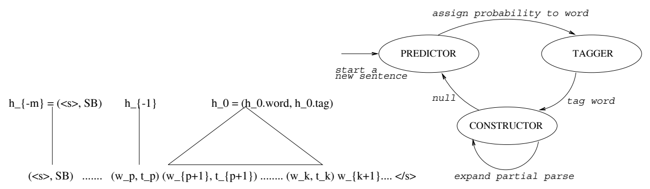
h_{-m} = (<s>, SB)
h_{-1}
h_0 = (h_0.word, h_0.tag)
(<s>, SB) .... (w_p, t_p) (w_{p+1}, t_{p+1}) .... (w_k, t_k) w_{k+1}.... </s>
assign probability to word
PREDICTOR
TAGGER
start a new sentence
null
tag word
CONSTRUCTOR
expand partial parse
(a) word-parse k-prefix (b) FSM

Fig. 1. A partial parse, and FSM representation of SLM

The number of possible parses for a given sentence grows exponentially with sentence length and therefore in practice, the SLM uses a *multi-stack* search strategy with pruning to construct and store only the more likely parses.

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

$$P_{SLM}(w_{k+1}|W_k) = \sum_{T_k \in S_k} P(w_{k+1}|W_kT_k)\cdot\rho(W_k,T_k) \quad (2)$$
$$\rho(W_k,T_k) = P(W_kT_k)/\sum_{T_k \in S_k} P(W_kT_k), \quad (3)$$

which ensures a proper probability normalization over strings W, where S_k is the set of all parses present in our stacks at the current stage k. The score $\rho(W_k, T_k)$ is the weight of the partial parse (W_k, T_k) in the stacks at position k.

As can be seen, the probability $P(w_k|W_{k-1}T_{k-1})$ is used in two different capacities in the operation of the SLM. Once in the construction of and assignment of probabilities to the partial parses (Equation 1), and then in computing the probability of the next word using the already constructed partial parses (Equation 2). We will refer to the second model as the SCORER component henceforth.

2.2. SCORER Training

The SLM training procedure involves using a treebank from which each component model is estimated by training it on the corresponding events extracted from the complete parses. Consequently, the PREDICTOR component is trained maximizing the joint probability given in Equation 1 over the training data. Since both the PREDICTOR and SCORER have the same probabilistic model, it is possible to copy the estimated PREDICTOR directly into the SCORER component. In fact, experiments with a neural net SCORER have shown that this "mismatched" training of the SCORER leads to significant improvements in perplexity and word error rate [1]. The training is mismatched because the model is trained to maximize the probability given by Equation 1 while it is going to be used in estimating probabilities in the form of Equation 3. Therefore, it is desirable to train the SCORER specifically to maximize the likelihood of the training data as given by Equation 3.

In general, we can assume that the type of events encountered by the PREDICTOR and the SCORER are similar to each other, and that the likelihood computed using Equation 1 would be a good approximation of the more proper left-to-right likelihood obtained from Equation 3.

In order to train the SCORER to directly optimize the left-to-right likelihood, we must first estimate the PREDICTOR, the TAGGER, and the CONSTRUCTOR components. Then, using these estimated models we will build (hypothesize) partial parses over the same training data and compute their corresponding scores. The SCORER component is then estimated using these scored partial parses. Notice that in this case we would have multiple partial parses per position (instead of one), therefore, the price paid for this more "exact" training is a considerable increase in the number of training events (which translates to a proportional increase in model size and training time for N-gram and neural net models respectively).

3. NEURAL NETWORK MODEL

In a neural network based language model words are represented by points in a continuous multi-dimensional feature space and the probability of a sequence of words is computed by means of a neural network. The feature vectors of the preceding words make up the input to the neural network, which then will produce a probability distribution over a given vocabulary [4]. The main idea behind this model is to make the estimation task easier by mapping words from the high-dimensional discrete space they exist in, to a low-dimensional continuous one where probability distributions are smooth functions in their variables. The network achieves generalization by assigning to an unseen word sequence a probability close to that of a "similar" word string seen in the training data. The similarity is defined as being close in the multi-dimensional feature space. Since the probability function is a smooth function of the feature vectors, a small change in the features leads to only a small change in the probability.

3.1. Model Details

Suppose the goal is to compute the probability of a certain event $Y = y$ given the values $x_1, x_2, \cdots, x_m$ of m conditioning variables. The conditional probability function $P(y|x_1, x_2, \cdots, x_m)$ is determined in two parts:

1. A mapping that associates a real vector of fixed dimension with each token in the *input vocabulary* V_i: the set of all tokens that can be used for prediction

2. A function which takes as the input the concatenation of the feature vectors of the input items $x_1, x_2, \cdots, x_m$. The function produces a conditional probability distribution (a vector) over the *output vocabulary* V_o: the set of all tokens to be predicted.

Training is achieved by searching for parameters Φ of the neural network and the values of feature vectors that maximize the penalized log-likelihood of the training corpus:

$$L=\frac{1}{T}\sum_t log P(y^t|x_1^t,\dots,x_m^t;\Phi)-R(\Phi) \quad (4)$$

where superscript t denotes the t^{th} event in the training data, T is the training data size and $R(\Phi)$ is a regularization term, which in our case is a factor of L2 norm squared of hidden and output layer weights.

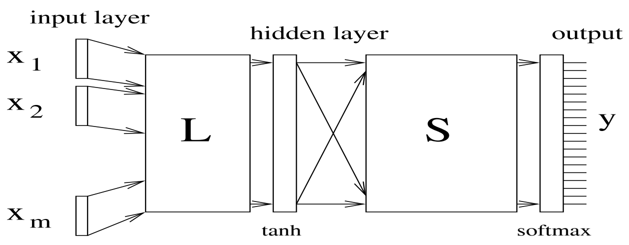

Fig. 2. The neural network architecture

The model architecture is given in Figure 2 [4]. The neural network is fully connected and contains one hidden layer. The operations of the input and hidden layers are given by:

$$\vec{f}=(f_1,\ldots,f_{d(m)})=(\vec{f}(x_1),\vec{f}(x_2),\cdots,\vec{f}(x_m))$$
$$g_k=\tanh\left(\sum_j f_j L_{kj}+B_k^h\right) \qquad k=1,2,\ldots,H$$

where $\vec{f}(x)$ is the d-dimensional feature vector for token x. The weights and biases of the hidden layer are denoted by L_{kj} and B_k^h respectively, and H is the number of hidden units. Indices are used to refer to specific elements of a matrix or vector.

At the output layer of the network we have:

$$z_k=\sum_j g_j S_{kj}+B_k^o \qquad k=1,2,\ldots,|V_o|$$
$$p_k=\frac{e^{z_k}}{\sum_j e^{z_j}} \qquad k=1,2,\ldots,|V_o| \tag{5}$$

with the weights and biases of the output layer denoted by S_{kj} and B_k^o respectively. The softmax layer (Equation 5) ensures that the outputs are valid probabilities and provides a suitable framework for learning a probability distribution.

The k^{th} output of the neural network, corresponding to the k^{th} item y_k of the output vocabulary, is exactly the sought conditional probability: $p_k = P(y^t = y_k|x_1^t, ..., x_m^t)$.

The neural network weights and biases, as well as the input feature vectors, are learned simultaneously using stochastic gradient descent training via back-propagation algorithm, with the objective function being the one given in Equation 4.

3.2. Neural Network Model for SLM

Recent work has shown that enriching the probabilistic dependencies of the SLM component models leads to significant improvements in the parsing accuracy as well as to reductions in both perplexity and word error rate [5]. However, a severe case of data sparseness was observed in those experiments.

Therefore, it is desirable to use neural network – capable of using long and enriched contexts – to model the SLM components [1]. In this work we use a neural network model as the SCORER component and train it on parses built either by an external source or the baseline SLM. All other components are retained from the baseline SLM [3] and are parameterized by N-gram interpolated models.

We chose to "upgrade" only the SCORER component mainly because doing so affects only the language model estimation part of the SLM, keeping the parse construction machinery unchanged from the baseline model, thus economizing on training effort. Furthermore, the SCORER has much higher perplexity than either the TAGGER, or the CONSTRUCTOR, leaving enough potential for a significant improvement in the model's performance.

3.3. Gradient Descent for Multiple Histories

The SLM stacks, at any stage i, contain the set S_i of all partial parses constructed and kept by the model up to that stage. The probability of the next word, computed as a weighted average of predictions by all the retained partial parses, is given by Equation 2. Representing the k^{th} partial parse at stage i as history h_i^k and its weight by ρ_i^k, the probability of the word string W_1^n is given by:

$$P(W_1^n)=\prod_{i=1}^n \sum_{k=1}^{k(i)} \rho_i^k \cdot P(w_i|h_i^k) \tag{6}$$

Where $k(i)$ denotes the number of partial parses (histories) at stage i. Consequently, the log-likelihood of a training data of size T will be in the form:

$$LL=\sum_{i=1}^T log\left(\sum_{k=1}^{k(i)} \rho_i^k \cdot P(w_i|h_i^k)\right) \tag{7}$$

Given this objective function, the gradient for every model parameter θ is computed as follows:

$$\begin{aligned}\frac{\partial}{\partial\theta} log\ P(w_i) &= \frac{\partial}{\partial\theta}\left(log\left(\sum_{k=1}^{k(i)} \rho_i^k \cdot P(w_i|h_i^k)\right)\right)\\ &= \frac{1}{\sum_{k=1}^{k(i)} \rho_i^k \cdot P(w_i|h_i^k)} \frac{\partial}{\partial\theta}\left(\sum_{k=1}^{k(i)} \rho_i^k \cdot P(w_i|h_i^k)\right)\\ &= \frac{1}{P(w_i)} \sum_{k=1}^{k(i)} \rho_i^k \cdot \frac{\partial}{\partial\theta} P(w_i|h_i^k)\end{aligned}$$

This means that for each word, the gradient (and parameter updates) for each partial parse predicting the word is computed, and the actual update will be a weighted average of the obtained updates, where the weights are the fixed scores of the partial parses.For more details on the algorithm the reader is referred to [2].

4. EXPERIMENTS

The baseline SLM components are parameterized as follows: both the PREDICTOR and the SCORER use the two previous heads as the context. The CONSTRUCTOR uses the same context plus the non-terminal tag of the third previous headword, and finally the TAGGER uses the current word plus the tags only of the two previous heads as its context.

All the components except the SCORER were trained on parses obtained from an external source. The estimated model was then used to construct (hypothesize) partial parses on the same training data. The neural net SCORER was then trained on these newly generated partial parses. The inputs to the network are a mixture of words and non-terminal tags, with each item being represented by a 30 dimensional feature vector. All the neural nets in these experiments have 100 hidden units and were trained with a starting learning rate of 10^{-3}.

Before training the neural net SCORER on the partial parses, we trained a separate neural net SCORER on the same events the PREDICTOR was estimated from (externally produced parses) [1]. Following the convention of Section 2.2, we refer to the latter model as the *mismatched* SCORER and to the one trained on SLM hypothesized partial parses as the *exact* SCORER. To speed up convergence, the exact SCORER training was initialized by copying the parameters from the trained mismatched SCORER. This resulted in a significant reduction in the number of iterations required to train the model.

Our experimental setup is as follows: for perplexity results we used the UPenn Treebank portion of the WSJ corpus. The corpus is divided into training, heldout, and test sets containing 930k, 74k, and 82k words respectively. We used an open vocabulary consisting of 10k words. The input vocabulary includes an additional 40 part-of-speech (POS) and 54 non-terminal (NT) tags.

The WER experiments consisted of the re-scoring of the K-best list output by a speech recognizer. We evaluated our models in the WSJ DARPA'93 HUB1 test setup. The test set is a collection of 213 utterances for a total of 3446 words. The 20k words open vocabulary and baseline 3-gram model are standard ones provided by NIST and LDC. The input vocabulary was again augmented with the 94 tags mentioned above; however, in order to save on training complexity, the output vocabulary was limited to the top 5k words, resulting in a proportional reduction (≈ 4 times) in training time [6] (6.2% OOV on training data). For the words outside this limited vocabulary we used the probabilities from a regular back-off 5-gram model. The lattice and K-best lists were generated using the standard 3-gram model trained on 45M words of the WSJ corpus. However, the baseline SLM, as well as all the neural net models, were trained on only a 19M words subset of the WSJ text. The heldout data consisted of 1.86M words.

Table 1 gives the perplexities on the UPenn corpus. The row 'SLM' refers to the baseline model while the rows denoted by '2HW', '3HW', and '(3+1)HW' correspond to contexts consisting of 2 previous heads, 3 previous heads, and 3 previous heads plus the first opposite head. The n^{th} previous opposite head is the child of n^{th} previous head that is not the head itself. The column headings '+slm', '+3gm', and '+5gm' indicate linear interpolation with the baseline SLM, a 3-gram, and a 5-gram model respectively. The 3-gram and 5-gram models are interpolated Kneser-Ney smoothed models built on the same training data as the baseline and neural net based SLMs, and give a perplexity of 148 and 141 on the test set, respectively. For each entry in the table, the perplexity of the neural net based SLM for both mismatched and exact trained SCORER are given, with the latter one in **bold** fonts. The neural net SCORERs were trained for a maximum of 50 and 30 iterations in the mismatch and exact cases respectively. As can be seen in the table, the neural net based SLM improves significantly over the baseline model. Furthermore, the exact model shows consistent improvement over the mismatched case. The lowest figure in the table (107) is the best published perplexity for this corpus.

The K-best re-scoring results are presented in Table 2. Here 'lattice' denotes the scores obtained using the standard 3-gram back-off model on the whole 45M words of the WSJ corpus. The 5-gram model (in '+5gm' column) is again an interpolated Kneser-Ney smoothed model build on the same 19M word text as the baseline and neural net based SLM models. All the interpolation weights were found on the test set using grid search with a step size of 0.05, the exception is the '+all' column were the SLM model is combined with all other three models using a step size of 0.1. The mismatched and exact SCORER networks were trained for 30 and 7 iterations respectively. The results show significant reductions in the WER by using a neural net based SLM. Also, it can be clearly seen that the exact SCORER trained model consistently outperforms the mismatch trained one. However, unlike the perplexity results, the consistent increase of context length does not translate to consistent improvement in WER. This has to do with the fact that the perplexity of a language model in not highly correlated with its performance in reducing the word error rate. Overall, the best result is achieved by the exact model (12.0% WER), which is the lowest published WER for this particular test setup. Note that this performance is attained using only a subset of the WSJ training data.

	no-intpl	+slm	+3gm	+5gm
SLM	161	161	137	132
2HW	166/**141**	135/**125**	125/**119**	121/**115**
3HW	161/**136**	132/**121**	123/**116**	119/**112**
(3+1)HW	155/**131**	129/**117**	121/**113**	117/**110**
All-3	152/**122**	128/**114**	120/**110**	117/**107**

Table 1. UPenn Perplexity

		+slm	+lattice	+5gm	+all
lattice	13.7	12.6	13.7	13.3	12.6
SLM	12.7	12.7	12.6	12.7	12.6
2HW	13.5/**12.8**	12.7/**12.3**	12.7/**12.4**	12.5/**12.3**	12.3/**12.0**
3HW	13.7/**12.9**	12.7/**12.7**	12.9/**12.9**	12.7/**12.6**	12.3/**12.4**
(3+1)HW	13.2/**12.5**	12.4/**12.3**	12.8/**12.4**	12.5/**12.1**	12.4/**12.0**

Table 2. WSJ Word Error Rate

5. CONCLUSIONS

In this paper we introduced an exact training procedure for the neural net based Structured Language Model. The exact training achieves consistently better results than the previously used method of approximate (mismatched) traininig. However, since the training is performed using "multiple" partial parses at each word position, there is a proportional increase in the required training time per iteration (though fewer iterations are required if the model is initialized by copying the trained mismatched model). In our experiments, the average number of partial parses in exact training turned out to be 11.12 and 8.68 for the UPenn and WSJ corpus respectively, increasing training times per iteration accordingly. On the other hand, the advantage of the mismatched training is that it can be used to obtain a quickly trained model that still outperforms the baseline SLM significantly.

For future work we plan to use the presented training method for a SLM model where all the components are modeled by neural networks.

6. REFERENCES

[1] Ahmad Emami, Peng Xu, and Frederick Jelinek, "Using a connectionist model in a syntactical based language model," in *Proc. ICASSP*, 2003.

[2] Ahmad Emami, "Improving a connectionist based syntactical language model," in *Proc. of EUROSPEECH'03.*, Geneva, Switzerland, September 2003.

[3] Ciprian Chelba and Frederick Jelinek, "Structured language modeling," *Computer Speech and Language*, vol. 14, no. 4, pp. 283–332, October 2000.

[4] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin, "A neural probabilistic language model," *Journal of Machine Learning Reseach*, vol. 3, pp. 1137–1155, 2003.

[5] Ciprian Chelba and Peng Xu, "Richer syntactic dependencies for structured language modeling," in *Proceedings of the ASRU Workshop*, December 2001.

[6] Holger Schwenk and Jean-Luc Gauvain, "Connectionist language modeling for large vocabulary continuous speech recognition," in *Proc. ICASSP*, 2002.