

# USING A CONNECTIONIST MODEL IN A SYNTACTICAL BASED LANGUAGE MODEL

*Ahmad Emami, Peng Xu, and Frederick Jelinek*

Center for Language and Speech Processing  
Johns Hopkins University  
Baltimore, MD 21218  
{*ahmad, xp, jelinek*}@*cls.jhu.edu*

## ABSTRACT

We investigate the performance of the Structured Language Model when one of its components is modeled by a connectionist model. Using a connectionist model and a distributed representation of the items in the history makes the component able to use much longer contexts than possible with currently used interpolated or back-off models, both because of the inherent capability of the connectionist model to fight the data sparseness problem, and because of the only sub-linear growth in the model size when increasing the context length. Experiments show significant improvement in perplexity and moderate reduction in word error rate over the baseline SLM results on the UPENN treebank and Wall Street Journal (WSJ) corpora respectively. The results also show that the probability distribution obtained by our model is much less correlated to regular N-grams than the baseline SLM model.

## 1. INTRODUCTION

A language model is a main component of many systems dealing with speech or natural language such as Speech Recognition or Machine Translation systems. N-gram language models are the models used in all of the current practical state-of-the-art systems. In these models only the surface (words only) information, and that only limited to a short span (last N-1 words), is used to predict the next word and the prediction is based on how often a given N-gram was seen in the training data. These models have the inherent disadvantage of having a short context available for the prediction. Increasing the context length is not trivial because the model size increases exponentially with context length, making it impossible to estimate the model parameters reliably even with very plentiful training data.

There have been attempts to use longer contexts by means of inducing some features from the whole past and using them (fewer parameters than if the corresponding past was used itself) for the prediction. In one such method, the Structured Language Model (SLM) [1], partial syntactical parses are built on the past sequence of words and a subset of information (features) gained from the parses is used to predict the next word. By using only a small number of features obtained from a parse of a long past, the Structured Language Model avoids the data sparseness problem without limiting itself to a short context. It also addresses the other problem with the N-gram models, the use of surface (lexical) words only, by using information and features from the deeper syntactic structure of the prefix of the sentence. The Structured Language Model shows improvement over N-gram models in perplexity as well as in reducing a speech recognizer's word error rate.

The choice of the features to be used for the purpose of prediction in SLM has been based mostly on intuition. In the original SLM work the two previous exposed headwords and their non-terminal tags were chosen from among all the information available in the partial parse. This decision was based on the belief that the exposed headwords have the highest predicting power among all the information gained from a partial parse. Ideally one would like to use as much information from the partial parses as possible. In fact, recent works [2, 3] have shown that using more information in the SLM leads to significant reductions in both perplexity and word error rate over a regular SLM baseline model. However, the problem is that the SLM internal models grow exponentially in size with the number of features used since they are structurally similar to a word N-gram model, and in fact, a severe data sparseness problem was observed when the number of conditioning features was increased.

The goal of this paper is to use as much information from the partial parses in the Structured Language Model as possible while avoiding the pitfall of data sparseness. This requires using a different architecture for the SLM internal models than the current deleted interpolation or back-off models that are very vulnerable to data sparsity.

There has been recent promising work in using distributional representation of words and neural networks for Language Modeling [4]. One great advantage of this approach is its ability to fight data sparseness. The model size grows only sub-linearly with the number of predicting features used. It has been shown that this method improves on regular N-gram models in both perplexity and word error rate [4, 5]. The ability of the method to accommodate longer contexts is most appealing to us. In fact, experiments have shown consistent improvements in perplexity and word error rate with increase in the context length.

In this paper we investigate the impact of using a neural network model as the component of the Structured Language Model that predicts the next word, giving it the ability to use many more features, while avoiding data sparseness, than the original SLM.

Section 2 serves as an introduction to the SLM, emphasizing on the parts we want to later modify. In section 3 we give a brief introduction to the neural net model and the distributional representation of words. Section 4 describes how we use the neural network model in the SLM. Finally, results are presented in Section 5.

## 2. 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 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

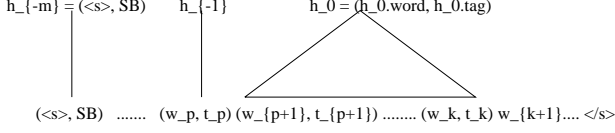


Fig. 1. A word-parse  $k$ -prefix

of length  $n$  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  $W_k = w_0 \dots w_k$  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 shows a word-parse  $k$ -prefix;  $h_0, \dots, h_{-m}$  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. The exposed heads at a given position  $k$  in the input sentence are a function of the word-parse  $k$ -prefix.

### 2.1. Probabilistic Model

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

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

where:

- $W_{k-1} T_{k-1}$  is the word-parse  $(k-1)$ -prefix
- $w_k$  is the word predicted by WORD-PREDICTOR
- $t_k$  is the 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 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  $i - th$  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.

It is worth noting that we can use a finite state machine, as shown in Figure 2, to characterize the operations of the SLM. The SLM starts from the PREDICTOR to predict the start-of-sentence symbol.

The SLM is based on three probabilities,  $P(w_k | W_{k-1} T_{k-1})$ ,  $P(t_k | w_k, W_{k-1} T_{k-1})$  and  $P(p_i^k | W_k T_k)$ . Each of the three probabilities can be parameterized (approximated) in different ways, as we will describe in Section 5.

The *language model* probability assignment for the word at

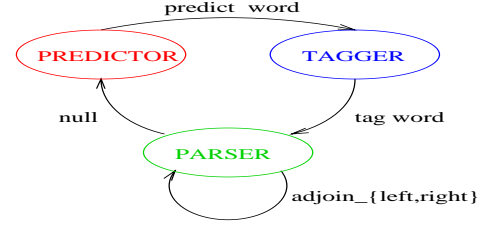


Fig. 2. Finite State Representation of the SLM

position  $k+1$  in the input sentence is made using:

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

$$\rho(W_k, T_k) = P(W_k T_k) / \sum_{T_k \in S_k} P(W_k T_k), \quad (3)$$

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  $k$ .

## 3. NEURAL NETWORK MODEL

Recently a relatively new type of language model has been introduced where words are represented by points in a multi-dimensional feature space and the probability of a sequence of words is computed by means of a neural network. The neural network, having the feature vectors of the preceding words as its input, estimates the probability of the next word [4]. The main idea behind this model is to fight the curse of dimensionality by interpolating the seen sequences in the training data. The generalization this model aims at is to assign to an unseen (in training data) word sequence a probability similar to that of a seen word sequence (sentence prefix) whose words are similar to those of the unseen word sequence. The similarity is defined as being close in the multi-dimensional space mentioned above.

In brief, this model can be described as follows, a *feature vector* is associated with each token in the *input vocabulary*, that is the vocabulary of all the items that can be used for conditioning. Then the conditional probability of the next word is expressed as a function of the input feature vectors by means of a neural network. This probability is produced for every possible next word from the *output vocabulary*. In general there is no relationship between the input and output vocabularies. The feature vectors and the parameters of the neural network are learned simultaneously during training. The number of features is much smaller than the vocabulary size, making it possible to reliably estimate the joint probability function. The input to the neural network are the features vectors for all the inputs concatenated, and the output is the conditional probability distribution over the output vocabulary. The idea here is that the words which are close to each other (close in the sense of their role in predicting words to follow) would have similar (close) feature vectors and since the probability function is a smooth function of these feature values, a small change in the features should only lead to a small change in the probability.

### 3.1. More Detail

The conditional probability function  $P(y | x_1, x_2, \dots, x_{n-1})$  where  $x_i$  and  $y$  are from the input and output vocabularies  $V_i$  and  $V_o$  respectively, is determined in two parts:

1. A mapping that associates with each word in the input vocabulary  $V_i$  a real vector of fixed length

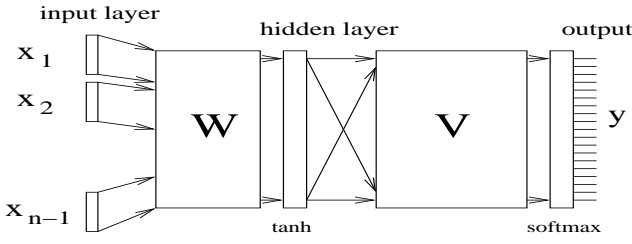


Fig. 3. The neural network architecture

2. A conditional probability function which takes as the input the concatenation of the feature vectors of the input items  $x_1, x_2, \dots, x_{n-1}$ . The function produces a probability distribution (a vector) over  $V_o$ , the  $i - th$  element being the conditional probability of the  $i - th$  member of  $V_o$ . This probability function is realized by a standard multi-layer neural network. A *softmax* function is used at the output of the neural net to make sure probabilities sum up to 1.

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

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

where  $P(y^t | x_1^t, \dots, x_{n-1}^t)$  is the probability of word  $y^t$  (network output at time  $t$ ),  $T$  is the training data size and  $R(\Phi)$  is a regularization term, sum of the parameters' squares in our case.

The model architecture is given in Figure 3. The neural network is a simple fully connected network with one hidden layer and sigmoid transfer functions. The input to the function is the concatenation of the feature vectors of the input items. The output of the output layer is passed through a softmax to make sure that the scores are positive and sum up to one, hence are valid probabilities. More specifically the output of the hidden layer is given by:

$$h_k = \tanh \left( \sum_j f_j W_{kj} + B_k^h \right) \quad k=1, 2, \dots, H$$

where  $h_k$  is the  $k - th$  output of the hidden layer,  $f_j$  is the  $j - th$  input of the network,  $W_{kj}$  and  $B_k^h$  are weight and bias elements for the hidden layer respectively, and  $H$  is the number of hidden units.

Furthermore, the outputs are given by:

$$z_k = \sum_j h_j V_{kj} + B_k^o \quad k=1, 2, \dots, |V_o|$$

$$p_k = \frac{e^{z_k}}{\sum_j e^{z_j}} \quad k=1, 2, \dots, |V_o| \quad (7)$$

The softmax layer (equation 7) ensures that the outputs are positive and sum up to one, hence are valid probabilities.

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, that is  $p_k = P(y^t = y_k | x_1^t, \dots, x_{n-1}^t)$ .

Standard back-propagation is used to train the parameters of the neural network as well as the feature vectors. See [6] for details about neural networks and back-propagation. The function we try to maximize is the log-likelihood of the training data given by equation 4.

We can see from equation 7 that the neural net model is similar in function to the maximum entropy model [7] except that the neural net learns the features by itself from the training data. It is most important to say that one of the great advantages of this model is that the number of inputs can be increased causing only

sub-linear increase in the model size, as opposed to exponential growth in N-gram models. This makes this model more capable in handling longer input spans.

#### 4. NEURAL NETWORK MODEL IN SLM

The standard structured language model suffers from severe data sparseness problems. Recent works [2, 3] have shown that increasing the amount of information available to the SLM components, namely the TAGGER, the CONSTRUCTOR, and the PREDICTOR, leads to a significant improvement on the parsing accuracy as well as a significant reduction in both perplexity and word error rate. However, a severe case of the data sparseness problem was observed in those experiments.

The fact that using longer contexts will improve the performance of the SLM, and the neural network capability in fighting the data sparseness problem, makes the connectionist model a very good candidate to use in any of the SLM internal models.

In this work we investigate the use of a neural net model as the PREDICTOR component of the Structured Language Model. The other components of the SLM remain unchanged. The reason that only PREDICTOR was chosen was that previous experiments with the SLM have shown that the data sparseness problem is much more severe for the PREDICTOR than for the other components and in fact the perplexity of both the TAGGER and the CONSTRUCTOR were found to be less than 2.

Furthermore, because the neural network model is computationally more expensive than the regular interpolated or back-off internal SLM models, we didn't use it for the PREDICTOR in finding the partial parses along with their probabilities (equations 1 and 3). We use the neural PREDICTOR only to estimate the probability of the next word given the already constructed partial parses (equation 2). More precisely, a neural network model will be used for *language model* prediction  $P(w_{k+1} | W_k T_k)$  in equation 2 but everything else remains unchanged in the standard SLM.

#### 5. EXPERIMENTS

The baseline Structured Language Model uses different conditioning contexts for its different components. The PREDICTOR uses 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 and tags of the two previous heads as its context.

We used the neural network model for the *language model* PREDICTOR while keeping the other components unchanged as discussed in section 4. We also increased the information available to the PREDICTOR.

The neural network is a standard multi-layered network exactly as described in section 3. The inputs to the network are a mixture of words and not-terminal tags. At the output layer we have to make sure that the output is still a probability distribution over words only. This means that the input and output vocabularies are different, with the output vocabulary being a subset of the input vocabulary. We used 30 dimensional feature vectors. The network had 100 hidden units with adaptive learning and a starting learning rate of 0.001. We used stochastic gradient descent for training and trained the network for a maximum of 50 iterations.

Table 1 gives the perplexity results on the UPENN section of the Wall Street Journal (WSJ) corpus. The vocabulary size was 10,000 and there were a total of 94 non-terminal tags and part of

		+slm	+3gm	+5gm
SLM	161	161	137	132
2HW	174	137	127	123
3HW	161	132	123	119
HW-OP	155	129	121	<b>117</b>

**Table 1.** UPENN section perplexity

speech tags. The rows denoted by 2HW, 3HW, and HW-OP correspond to contexts consisting of 2 previous heads, 3 previous heads, and 3 previous heads plus the first previous opposite head. The  $n - th$  previous opposite head is the child of the  $n - th$  previous head that is not the head itself. The columns +slm, +3gm, and +5gm denote linear interpolation with the baseline SLM, a 3-gram back-off, and 5-gram back-off models respectively, with weights found on some held-out set. It can be seen that adding more context always helps and the best result on the longest context improves on the best result for the baseline significantly. The interesting point is that the neural network model seems to be much more uncorrelated with the 3-gram or 5-gram models than is the SLM. This can be observed best for the shortest context where the connectionist model performs worse than the SLM baseline, but then does significantly better when interpolated with a regular N-gram model. The SLM model simply can't reproduce the same improvements when combined with the same N-gram models.

With significant improvements gained on the perplexity we carried out some experiments to see how well the model performs its function in speech recognition and in reducing the word error rate. Our model was used to re-rank the output N-best list of a speech recognizer on the Wall Street Journal Corpus. The vocabulary was 19,006 and there were again 94 non-terminal tag and part of speech types. Because of the larger size of the WSJ corpus the training time was considerably longer than for the UPENN corpus. We limited the size of the output vocabulary to 5000 words, reducing computations almost proportionally to the reduction in vocabulary size [5]. The out of vocabulary (OOV) rate for this limited vocabulary is rather low so we are not expecting a major degradation in performance. For the words outside this limited vocabulary we used the regular back-off 5-gram probabilities. Substituting probabilities in this manner causes them not to sum up to one anymore but this is not critical since we are using the probabilities only as scores to re-rank the recognizer's hypotheses. The rest of the network is the same as in the perplexity experiment except that the number of iterations was limited to a maximum of 30. We should also note that the neural network was trained only on half of the available WSJ data, the same amount the baseline SLM is trained on. That is also the case for the 3-gram and 5-gram back-off models we used but the language model of the recognizer is trained on the whole WSJ corpus. The results are given in Table 2. Here the row Lattice denotes the language model from the speech recognizer and the column +l&5gm denotes interpolation with the lattice and back-off 5-gram models. The linear interpolation weights were chosen to give the best performance on the test set itself.

The model gives an improvement of 1.6% relative over the baseline model. The results suggest that the shorter context had almost the same performance as the longest context, but in our experiments in combining all the different models with different weights we found the longest context results to be more consistent in general. However, the results on word error rate don't constitute as good an improvement as we obtained for perplexity and we think that the main reason is that the model is optimized explic-

		+slm	+lattice	+5gm	+l&5gm
Lattice	13.7	12.6	13.7	13.2	13.2
SLM	12.7	12.7	12.6	12.7	12.6
2HW	13.5	12.7	12.7	12.5	<b>12.4</b>
3HW	13.6	12.6	12.8	12.7	12.6
HW-OP	13.2	12.5	12.9	<b>12.4</b>	<b>12.4</b>

**Table 2.** WSJ word error rate

itly for perplexity without any consideration for its performance on word error rate reduction.

## 6. CONCLUSION AND FUTURE WORK

In this paper we presented the integration of a neural network model into the Structured Language Model. The neural network model gives the SLM the power to use more features when predicting the next word in its language model computations. The connectionist model seemed a good choice because of its capability in fighting the data sparseness problem, which is severe in SLM. Experiments showed significant reduction in perplexity. It was also observed that the neural network model produces probabilities which are much less correlated with the regular N-gram models than are those of the baseline SLM. Separate experiments showed a moderate reduction in word error rate. We plan to continue this work by using the neural network model also for the other components of the SLM.

## 7. ACKNOWLEDGEMENT

We would like to thank the Center for Imaging Science at the Johns Hopkins University for the use of the RS/6000 SP machine provided by IBM corporation.

## 8. REFERENCES

- [1] Ciprian Chelba and Frederick Jelinek, "Structured language modeling," *Computer Speech and Language*, vol. 14, no. 4, pp. 283–332, October 2000.
- [2] Ciprian Chelba and Peng Xu, "Richer syntactic dependencies for structured language modeling," in *Proceedings of the Automatic Speech Recognition and Understanding Workshop*, Madonna di Campiglio, Trento-Italy, December 2001.
- [3] Peng Xu, Ciprian Chelba, and Frederick Jelinek, "A study on richer syntactic dependencies for structured language modeling," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, PA, 2002.
- [4] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," in *Advances in Neural Information Processing Systems*, 2001.
- [5] Holger Schwenk and Jean-Luc Gauvain, "Connectionist language modeling for large vocabulary continuous speech recognition," in *Proc. ICASSP*, 2002.
- [6] Simon Haykin, *Neural Networks, A Comprehensive Foundation*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.
- [7] A. L. Berger, S. A. Della Pietra, and V. J. Della Pietra, "A maximum entropy approach to natural language processing," *Computational Linguistics*, vol. 22, no. 1, pp. 39–72, March 1996.